Apresentação da Biblioteca NumPy

Prof. Dr. Fabiano B. Menegidio

1 Introdução

O **NumPy** (*Numerical Python*) é uma das bibliotecas mais fundamentais para o trabalho com computação científica e análise de dados em Python. Seu principal foco está na manipulação de **arrays** e **matrizes** multidimensionais, além de fornecer uma vasta gama de funções matemáticas de alto desempenho para operações rápidas sobre grandes conjuntos de dados. NumPy é amplamente utilizado em áreas como Inteligência Artificial (IA), *Machine Learning*, processamento de sinais, computação gráfica e em qualquer domínio onde operações numéricas eficientes são necessárias.

2 Por que usar NumPy?

Em Python, o tipo de dado básico para armazenar números é a lista, mas listas não são otimizadas para operações matemáticas complexas, especialmente quando lidamos com grandes volumes de dados. NumPy oferece uma solução a esse problema, fornecendo a estrutura **ndarray**, que permite a criação de arrays e matrizes multidimensionais com uma série de otimizações para armazenamento e processamento eficiente.

O NumPy é escrito em C e Fortran, o que o torna extremamente rápido em comparação com operações matemáticas realizadas diretamente com listas Python. Em problemas de IA e Engenharia de Software, onde o desempenho é crucial, essa biblioteca é muitas vezes a base para outras bibliotecas mais complexas, como *SciPy*, *Pandas* e *TensorFlow*.

3 Principais Utilizações do NumPy

- **Manipulação de Arrays e Matrizes**: A principal estrutura do NumPy é o array multidimensional, que permite operações elementares como adição, subtração e multiplicação de maneira extremamente eficiente.
- **Cálculos Matemáticos e Estatísticos**: NumPy oferece funções avançadas para cálculo de média, desvio padrão, soma, produto, etc.
- **Álgebra Linear**: A biblioteca inclui funções para a multiplicação de matrizes, cálculo de determinantes, inversão de matrizes, e resolução de

sistemas lineares, fundamentais para aprendizado de máquina e análise de dados.

- **Simulação e Geração de Dados Aleatórios**: NumPy também tem uma subbiblioteca para gerar números aleatórios e realizar simulações probabilísticas, muito usada em Monte Carlo e testes estatísticos.
- **Integração com Outras Bibliotecas**: NumPy serve como base para outras bibliotecas como Pandas (para manipulação de dados tabulares), SciPy (para computação científica) e Matplotlib (para visualização de dados), fornecendo o tipo de dados fundamental para operações numéricas.

4 Estrutura Básica do NumPy: O `ndarray`

O principal objeto do NumPy é o **ndarray** (array N-dimensional), que é um array multidimensional de elementos do mesmo tipo de dados. Ao contrário das listas em Python, os arrays NumPy são homogêneos e podem conter apenas dados de um único tipo, o que permite que o NumPy seja mais eficiente no uso de memória e processamento.

```
import numpy as np

# Exemplo de cria  o de um array NumPy
array = np.array([1, 2, 3, 4])
print(array)
```

Listing 1: Exemplo de criação de um array NumPy

Ao criar arrays, você pode definir dimensões de várias formas, desde vetores 1D até tensores de múltiplas dimensões, como matrizes 2D ou tensores 3D.

5 Principais Funções do NumPy

NumPy oferece uma vasta gama de funções e métodos para lidar com operações matemáticas, manipulação de arrays e cálculos de álgebra linear. Abaixo, descrevemos as funções mais essenciais.

5.1 Criação de Arrays

NumPy facilita a criação de arrays de diferentes formas:

- `np.array()`: Cria arrays a partir de listas ou tuplas.
- `np.zeros()`: Cria um array preenchido com zeros.
- `np.ones()`: Cria um array preenchido com uns.
- `np.eye()`: Cria uma matriz identidade.

- `np.linspace()`: Cria um array com valores igualmente espaçados dentro de um intervalo.
- `np.random.rand()`: Gera arrays com valores aleatórios entre 0 e 1.

```
import numpy as np
array_zeros = np.zeros((3, 3))  # Matriz 3x3 com zeros
array_rand = np.random.rand(4)   # Vetor com 4 n meros aleat rios
```

Listing 2: Exemplo de criação de arrays NumPy

5.2 Operações Matemáticas Elementares

NumPy suporta operações aritméticas entre arrays de maneira elementar (elemento por elemento), sem a necessidade de laços explícitos:

- `np.add()`, `np.subtract()`, `np.multiply()`, `np.divide()`: Operações matemáticas básicas.
- `np.dot()`: Produto escalar e produto de matrizes.
- `np.sum()`, `np.prod()`: Soma e produto de todos os elementos.
- `np.exp()`, `np.log()`, `np.sqrt()`: Operações exponenciais, logarítmicas e radicais.

```
import numpy as np

# Arrays para realizar opera  es
array1 = np.array([1, 2, 3])
array2 = np.array([4, 5, 6])

# Opera  es b sicas
soma = np.add(array1, array2)
produto = np.dot(array1, array2)  # Produto escalar
```

Listing 3: Operações básicas com arrays NumPy

5.3 Estatísticas e Funções de Agregação

NumPy tem várias funções de agregação úteis para estatísticas descritivas:

- `np.mean()`: Calcula a média dos elementos do array.
- `np.median()`: Calcula a mediana.
- `np.std()`: Calcula o desvio padrão.
- `np.var()`: Calcula a variância.

```python
import numpy as np

array = np.array([1, 2, 3, 4, 5])
media = np.mean(array)
desvio_padrao = np.std(array)
```

Listing 4: Exemplo de funções de estatísticas

5.4 Manipulação e Fatiamento de Arrays

A manipulação de dados no NumPy é altamente eficiente devido ao suporte para fatiamento e indexação avançada:

- `array[linha, coluna]`: Acessa elementos específicos de um array.
- `array[:, :]`: Fatiamento para acessar partes de um array.
- `np.concatenate()`: Concatena arrays ao longo de eixos especificados.
- `np.reshape()`: Redimensiona o array sem alterar seus dados.

```python
import numpy as np

# Criar um array 3x3
array = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Fatiar a primeira linha
primeira_linha = array[0, :]
# Redimensionar o array 3x3 para 1x9
array_reshaped = np.reshape(array, (1, 9))
```

Listing 5: Exemplo de manipulação de arrays NumPy

5.5 Álgebra Linear

NumPy tem um módulo dedicado à álgebra linear, com funções avançadas:

- `np.linalg.inv()`: Inversão de matrizes.
- `np.linalg.det()`: Determinante de uma matriz.
- `np.linalg.eig()`: Autovalores e autovetores.
- `np.linalg.solve()`: Resolução de sistemas lineares.

```python
import numpy as np

# Criar uma matriz 2x2
A = np.array([[1, 2], [3, 4]])

# Calcular a inversa e o determinante
inversa = np.linalg.inv(A)
determinante = np.linalg.det(A)
```

Listing 6: Exemplo de álgebra linear com NumPy

5.6 Geração de Números Aleatórios

A subbiblioteca `np.random` é responsável pela geração de números aleatórios e é amplamente utilizada em simulações e amostragem:

- `np.random.rand()`: Gera números aleatórios uniformemente distribuídos entre 0 e 1.
- `np.random.randn()`: Gera números aleatórios seguindo uma distribuição normal padrão.
- `np.random.randint()`: Gera inteiros aleatórios em um intervalo.
- `np.random.choice()`: Seleciona elementos aleatórios de um array.

```
import numpy as np

# Gera  o de n meros aleat rios
aleatorios_uniformes = np.random.rand(10) # 10 n meros aleat rios
    entre 0 e 1
aleatorios_normais = np.random.randn(10)  # 10 n meros com
   distribui  o normal
```

Listing 7: Exemplo de geração de números aleatórios com NumPy

6 Conclusão

O NumPy é uma biblioteca extremamente poderosa e eficiente para manipulação de arrays e operações numéricas em Python. Sua capacidade de realizar cálculos rápidos com grandes quantidades de dados torna-o uma ferramenta indispensável para cientistas de dados, engenheiros de software e desenvolvedores de IA. Além de ser a base para muitas outras bibliotecas, suas funções avançadas para álgebra linear, estatística e geração de números aleatórios tornam o NumPy uma escolha ideal para quem lida com computação científica e análise de dados.

Exercícios de NumPy

Prof. Dr. Fabiano B. Menegidio

1 Exercício 1: Criação e Manipulação de Arrays NumPy

Contexto: No desenvolvimento de sistemas de IA, muitas vezes lidamos com grandes volumes de dados numéricos, como matrizes e vetores. NumPy é uma biblioteca otimizada para esse tipo de operação.

Enunciado: Crie um array NumPy 3x3 com números aleatórios e, em seguida, realize as seguintes operações:

1. Encontre o maior valor do array.
2. Encontre o menor valor do array.
3. Calcule a soma de todos os elementos.

Resolução:

```python
import numpy as np

# Criar um array 3x3 com valores aleat rios
array = np.random.random((3, 3))
print("Array Original:")
print(array)

# Encontrar o maior valor
max_value = np.max(array)
print("\nMaior valor:", max_value)

# Encontrar o menor valor
min_value = np.min(array)
print("Menor valor:", min_value)

# Somar todos os elementos do array
sum_value = np.sum(array)
print("Soma dos elementos:", sum_value)
```

Listing 1: Resolução do Exemplo 1

Explicação:

- `np.random.random((3, 3))` gera um array 3x3 com valores aleatórios entre 0 e 1.
- `np.max(array)` retorna o maior valor presente no array.
- `np.min(array)` retorna o menor valor.
- `np.sum(array)` calcula a soma de todos os elementos do array.

Essas operações são comuns em manipulações numéricas em IA, como o ajuste de pesos em redes neurais.

2 Exercício 2: Produto Escalar entre Vetores

Contexto: O produto escalar é frequentemente utilizado em IA para cálculos de similaridade em aprendizado de máquina, como no algoritmo de redes neurais.

Enunciado: Crie dois vetores de tamanho 3 e calcule o produto escalar entre eles.

Resolução:

```
import numpy as np

# Criar dois vetores
vetor1 = np.array([1, 2, 3])
vetor2 = np.array([4, 5, 6])

# Calcular o produto escalar
produto_escalar = np.dot(vetor1, vetor2)
print("Produto escalar:", produto_escalar)
```

Listing 2: Resolução do Exemplo 2

Explicação:

- `np.array([1, 2, 3])` cria um vetor NumPy.
- `np.dot(vetor1, vetor2)` calcula o produto escalar entre dois vetores, que é muito usado para medir a similaridade entre vetores de características em aprendizado de máquina.

3 Exercício 3: Operações Matemáticas Elementares

Contexto: Ao trabalhar com dados numéricos, é essencial realizar operações matemáticas elementares, como adição, subtração e multiplicação.

Enunciado: Crie dois arrays NumPy 2x2 e realize as operações de adição, subtração e multiplicação elemento por elemento.

Resolução:

```
import numpy as np

# Criar dois arrays 2x2
array1 = np.array([[1, 2], [3, 4]])
array2 = np.array([[5, 6], [7, 8]])

# Opera  es
soma = array1 + array2
subtracao = array1 - array2
multiplicacao = array1 * array2

print("Soma:\n", soma)
print("Subtra  o:\n", subtracao)
print("Multiplica  o elemento por elemento:\n", multiplicacao)
```

Listing 3: Resolução do Exemplo 3

Explicação:

- A adição, subtração e multiplicação de arrays em NumPy são feitas elemento por elemento, facilitando operações matriciais em algoritmos de IA.

4 Exercício 4: Fatiamento de Arrays

Contexto: Em muitos casos, é necessário trabalhar com partes específicas de um conjunto de dados, como ao alimentar subconjuntos de dados em um modelo.

Enunciado: Crie um array 4x4 e obtenha o subarray 2x2 que está no canto superior esquerdo.

Resolução:

```
import numpy as np

# Criar um array 4x4
array = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13,
    14, 15, 16]])

# Obter o subarray 2x2
subarray = array[:2, :2]
print("Subarray 2x2:\n", subarray)
```

Listing 4: Resolução do Exemplo 4

Explicação:

- O fatiamento de arrays permite extrair subconjuntos de dados de maneira eficiente. Isso é útil quando se trabalha com grandes matrizes de dados, como em processamento de imagem.

5 Exercício 5: Transposição de Arrays

Contexto: A transposição de matrizes é uma operação comum em algoritmos de aprendizado de máquina, especialmente em redes neurais e álgebra linear.

Enunciado: Crie um array 3x2 e calcule sua transposta.

Resolução:

```
import numpy as np

# Criar um array 3x2
array = np.array([[1, 2], [3, 4], [5, 6]])

# Calcular a transposta
transposta = np.transpose(array)
print("Transposta:\n", transposta)
```

Listing 5: Resolução do Exemplo 5

Explicação:

- `np.transpose(array)` calcula a transposta da matriz, que troca as linhas pelas colunas, sendo essencial em muitos cálculos algébricos.

6 Exercício 6: Geração de Matrizes Identidade

Contexto: Matrizes identidade são fundamentais em álgebra linear, sendo utilizadas em métodos numéricos, como na resolução de sistemas de equações lineares.

Enunciado: Crie uma matriz identidade 4x4 usando NumPy.

Resolução:

```
import numpy as np

# Criar uma matriz identidade 4x4
matriz_identidade = np.eye(4)
print("Matriz Identidade 4x4:\n", matriz_identidade)
```

Listing 6: Resolução do Exemplo 6

Explicação:

- `np.eye(4)` cria uma matriz identidade de dimensão 4x4, onde os elementos da diagonal são 1 e o restante é 0. Essas matrizes são fundamentais para cálculos em redes neurais.

7 Exercício 7: Resolução de Sistemas Lineares

Contexto: Resolver sistemas de equações lineares é uma tarefa comum em otimização, que está presente em muitas áreas da IA.

Enunciado: Dado o sistema de equações:

$$2x + y = 5$$

$$x + 3y = 7$$

Use o NumPy para encontrar os valores de x e y.

Resolução:

```
import numpy as np

# Coeficientes das equa  es
A = np.array([[2, 1], [1, 3]])
# Resultados das equa  es
B = np.array([5, 7])

# Resolver o sistema de equa  es
solucao = np.linalg.solve(A, B)
print("Solu  o [x, y]:", solucao)
```

Listing 7: Resolução do Exemplo 7

Explicação:

- `np.linalg.solve(A, B)` resolve sistemas lineares da forma $Ax = B$. Este método é amplamente utilizado em métodos de otimização e aprendizado de máquina.

8 Exercício 8: Média e Desvio Padrão

Contexto: A análise estatística de dados é importante na preparação de dados para modelos de IA. A média e o desvio padrão são medidas de tendência central e dispersão.

Enunciado: Crie um array de 10 números aleatórios e calcule a média e o desvio padrão.

Resolução:

```
import numpy as np

# Criar um array de 10 n meros aleat rios
array = np.random.rand(10)

# Calcular a m dia
media = np.mean(array)

# Calcular o desvio padr o
desvio_padrao = np.std(array)

print("Array:", array)
print("M dia:", media)
print("Desvio padr o:", desvio_padrao)
```

Listing 8: Resolução do Exemplo 8

Explicação:

- `np.mean(array)` calcula a média dos elementos do array.
- `np.std(array)` retorna o desvio padrão, indicando a dispersão dos dados em torno da média.

9 Exercício 9: Geração de Arrays com `linspace`

Contexto: Muitas vezes, precisamos gerar sequências de números com um determinado intervalo. `linspace` é útil para isso, especialmente ao plotar gráficos.

Enunciado: Gere um array de 50 valores igualmente espaçados entre 0 e 10.

Resolução:

```
import numpy as np

# Gerar um array de 50 valores entre 0 e 10
array = np.linspace(0, 10, 50)
print("Array de 50 valores entre 0 e 10:\n", array)
```

Listing 9: Resolução do Exemplo 9

Explicação:

- `np.linspace(0, 10, 50)` gera 50 valores igualmente espaçados entre 0 e 10, o que é útil para gráficos e cálculos numéricos.

10 Exercício 10: Ordenação de Arrays

Contexto: A ordenação de dados é uma operação comum na preparação de dados para algoritmos de aprendizado de máquina, onde é necessário trabalhar com sequências ordenadas.

Enunciado: Crie um array com 10 números aleatórios e o ordene em ordem crescente.

Resolução:

```
import numpy as np

# Criar um array com 10 n meros aleat rios
array = np.random.rand(10)

# Ordenar o array
array_ordenado = np.sort(array)
print("Array original:", array)
print("Array ordenado:", array_ordenado)
```

Listing 10: Resolução do Exemplo 10

Explicação:

- `np.sort(array)` ordena os elementos do array em ordem crescente, facilitando a análise de dados.

Apresentação da Biblioteca SciPy

Prof. Dr. Fabiano B. Menegidio

1 Introdução

O **SciPy** (Scientific Python) é uma biblioteca de código aberto construída sobre o **NumPy**, que amplia as funcionalidades desta última, fornecendo uma coleção de algoritmos matemáticos e funções científicas avançadas para resolver problemas complexos de ciência de dados, engenharia, estatística e computação científica em geral. A biblioteca SciPy é usada extensivamente em áreas como processamento de sinais, otimização, integração, álgebra linear, equações diferenciais e estatísticas.

2 Por que usar o SciPy?

Enquanto o NumPy oferece uma base sólida para manipulação de arrays e operações matemáticas elementares, o SciPy vai além, fornecendo ferramentas mais avançadas para computação científica. O SciPy é composto de diversos submódulos que permitem resolver problemas complexos de forma eficiente e precisa.

Além de ser muito utilizado em áreas de ciência e engenharia, o SciPy também desempenha um papel importante em *Inteligência Artificial* (IA) e *Machine Learning*, especialmente em problemas que envolvem álgebra linear, otimização de funções, interpolação e estatística. Seu foco em cálculos numéricos de alta performance torna o SciPy uma ferramenta essencial para cientistas de dados, engenheiros e desenvolvedores que lidam com dados e algoritmos avançados.

3 Principais Utilizações do SciPy

- **Otimização de Funções**: O SciPy fornece funções robustas para encontrar mínimos e máximos de funções, essenciais em problemas de aprendizado de máquina e ajuste de modelos.
- **Integração e Equações Diferenciais**: A biblioteca oferece ferramentas para resolver integrais e equações diferenciais ordinárias (EDOs), que são úteis em modelagem de sistemas físicos e dinâmicos.

- **Interpolação de Dados**: O SciPy permite criar funções interpoladoras a partir de pontos de dados discretos, muito útil para preencher valores faltantes ou suavizar dados.
- **Álgebra Linear**: SciPy inclui uma ampla gama de ferramentas para álgebra linear, como decomposições de matrizes e resolução de sistemas lineares.
- **Processamento de Sinais e Transformada de Fourier**: Para análise de sinais, o SciPy inclui transformadas rápidas de Fourier (FFT) e ferramentas para filtrar sinais.
- **Estatística e Distribuições**: A biblioteca fornece métodos avançados para análise estatística e geração de distribuições de probabilidade.

4 Estrutura e Submódulos do SciPy

A estrutura da biblioteca SciPy é organizada em submódulos, cada um responsável por uma área específica da computação científica. Cada um desses submódulos fornece funções especializadas que ampliam o uso de arrays do NumPy em contextos mais complexos.

- `scipy.optimize`: Funções para otimização e ajuste de curvas.
- `scipy.integrate`: Funções para cálculo de integrais e resolução de equações diferenciais.
- `scipy.linalg`: Ferramentas de álgebra linear avançada.
- `scipy.fft`: Transformadas de Fourier rápidas.
- `scipy.stats`: Funções para análise estatística e distribuições de probabilidade.
- `scipy.interpolate`: Funções para interpolação de dados.

5 Principais Funções do SciPy

O SciPy oferece uma ampla gama de funções que são organizadas em seus submódulos. A seguir, discutimos algumas das funções mais utilizadas e importantes em cada uma das áreas oferecidas pela biblioteca.

5.1 1. Otimização de Funções com `scipy.optimize`

O submódulo `scipy.optimize` fornece algoritmos para otimização de funções e ajuste de curvas, incluindo métodos para encontrar mínimos locais e globais de funções. A otimização é amplamente utilizada em *Machine Learning* para minimizar funções de perda ou encontrar parâmetros ideais para modelos.

Exemplo básico de minimização de uma função:

```
from scipy.optimize import minimize

# Definir a fun   o que desejamos minimizar
def funcao(x):
    return (x - 3) ** 2 + 2

# Usar o m todo de minimiza   o partindo de uma estimativa
    inicial
resultado = minimize(funcao, 0)
print("M nimo encontrado em x =", resultado.x)
```

- `minimize(funcao, 0)`: Minimiza a função `funcao` iniciando em 0.
- Otimizações como essa são essenciais em ajuste de parâmetros de modelos em aprendizado de máquina.

5.2 2. Integração e Solução de Equações Diferenciais com `scipy.integrate`

O submódulo `scipy.integrate` inclui funções para resolver integrais numéricas e equações diferenciais. O cálculo de integrais é usado, por exemplo, para calcular probabilidades em distribuições contínuas, enquanto a solução de equações diferenciais é fundamental em sistemas dinâmicos.

Exemplo de cálculo de uma integral:

```
from scipy.integrate import quad

# Definir a fun   o a ser integrada
def f(x):
    return x ** 2

# Calcular a integral de f(x) de 0 a 1
resultado, erro = quad(f, 0, 1)
print("Resultado da integral:", resultado)
```

- `quad(f, 0, 1)`: Calcula a integral da função `f(x) = x`$^2 no intervalo [0, 1]. Esse método é útil em cálculos de áreas$ss

Exemplo de resolução de equações diferenciais:

```
from scipy.integrate import solve_ivp

# Definir a equa   o diferencial
def eq_diferencial(t, y):
    return -2 * y

# Resolver a equa   o com condi   o inicial y(0) = 1
solucao = solve_ivp(eq_diferencial, [0, 5], [1])
print("Solu   o em t=5:", solucao.y[0][-1])
```

- `solve_ivp` resolve equações diferenciais ordinárias, como $\frac{dy}{dt} = -2y$, essencial em modelagem de sistemas físicos e dinâmicos.

5.3 3. Interpolação de Dados com `scipy.interpolate`

A interpolação é a técnica de construir novos pontos a partir de um conjunto de pontos discretos. Ela é usada em análise de dados para preencher valores faltantes ou para suavização de curvas. O submódulo `scipy.interpolate` oferece funções para diversos métodos de interpolação, como linear, spline e polinomial.

Exemplo de interpolação linear:

```
from scipy.interpolate import interp1d
import numpy as np

# Definir os pontos de dados
x = np.array([0, 1, 2, 3, 4])
y = np.array([1, 2, 0, 2, 1])

# Criar a fun   o de interpola   o linear
f = interp1d(x, y, kind='linear')

# Obter valores interpolados para novos pontos
x_novos = np.array([0.5, 1.5, 2.5])
y_novos = f(x_novos)
print("Valores interpolados:", y_novos)
```

- `interp1d(x, y)`: Cria uma função de interpolação a partir dos pontos fornecidos.
- Esse método é útil para preencher dados ausentes e suavizar dados experimentais.

5.4 4. Álgebra Linear com `scipy.linalg`

O submódulo `scipy.linalg` fornece funções para álgebra linear avançada, como decomposições de matrizes, cálculo de determinantes, autovalores e autovetores. Embora o NumPy também tenha funções de álgebra linear, o SciPy oferece versões mais completas e eficientes.

Exemplo de inversão de matriz:

```
from scipy.linalg import inv
import numpy as np

# Criar uma matriz 2x2
A = np.array([[1, 2], [3, 4]])

# Calcular a inversa
inversa = inv(A)
print("Inversa da matriz A:", inversa)
```

- `inv(A)`: Calcula a inversa da matriz `A`.
- Essas operações são fundamentais em redes neurais e regressão linear.

5.5 5. Transformadas de Fourier com `scipy.fft`

O submódulo `scipy.fft` fornece funções para realizar transformadas de Fourier, que são amplamente utilizadas em processamento de sinais e séries temporais. A Transformada de Fourier é usada para converter sinais no domínio do tempo para o domínio da frequência.

Exemplo de transformada de Fourier:

```
from scipy.fft import fft
import numpy as np

# Criar um sinal simples
t = np.linspace(0, 1, 500)
sinal = np.sin(2 * np.pi * t) + 0.5 * np.sin(4 * np.pi * t)

# Calcular a transformada de Fourier
frequencias = fft(sinal)
print("Transformada de Fourier:", frequencias)
```

- `fft(sinal)`: Calcula a transformada de Fourier do sinal `sinal`, útil em análise de sinais e filtragem de ruídos.

5.6 6. Estatística com `scipy.stats`

O submódulo `scipy.stats` fornece uma vasta coleção de funções estatísticas, incluindo testes de hipótese, distribuições de probabilidade e estatísticas descritivas.

Exemplo de geração de números aleatórios de uma distribuição normal:

```
from scipy.stats import norm

# Gerar 10 n meros aleat rios de uma distribui  o normal
dados = norm.rvs(size=10)
print("N meros aleat rios de distribui  o normal:", dados)
```

- `norm.rvs(size=10)`: Gera 10 números aleatórios de uma distribuição normal.

6 Conclusão

O SciPy é uma biblioteca poderosa e essencial para a computação científica em Python. Ele oferece uma vasta gama de ferramentas e funções avançadas para resolver problemas complexos de matemática, álgebra linear, estatística, otimização, integração e muito mais. Sua integração perfeita com o NumPy e outras bibliotecas populares como Pandas e Matplotlib faz do SciPy uma peça central em projetos de ciência de dados, aprendizado de máquina e engenharia.

Para quem trabalha com modelagem, simulações e análise numérica, o SciPy é indispensável, fornecendo as bases para cálculos precisos e eficientes.

Exercícios Resolvidos de SciPy

Prof. Dr. Fabiano B. Menegidio

1 Exercício 1: Cálculo de Integrais Usando SciPy

Contexto: O cálculo de integrais é fundamental em várias áreas da matemática aplicada e física. Em IA, é utilizado, por exemplo, em modelos de probabilidade e cálculos de área sob curvas ROC.

Enunciado: Calcule a integral da função $f(x) = x^2$ no intervalo de 0 a 1 usando SciPy.

Resolução:

```
from scipy import integrate

# Definir a fun   o
def f(x):
    return x**2

# Calcular a integral de 0 a 1
resultado, erro = integrate.quad(f, 0, 1)
print("Resultado da integral:", resultado)
print("Erro estimado:", erro)
```

Listing 1: Resolução do Exemplo 1

Explicação:

- `integrate.quad(f, 0, 1)`: Calcula a integral definida da função $f(x) = x^2$ no intervalo de 0 a 1.
- O primeiro valor retornado é o resultado da integral, e o segundo é o erro estimado do cálculo.

Esse método é amplamente utilizado em IA quando há necessidade de calcular probabilidades contínuas ou áreas sob curvas.

2 Exercício 2: Resolução de Equações Diferenciais Ordinárias (EDOs)

Contexto: Equações diferenciais são comuns em modelagem de sistemas dinâmicos, como controle de robôs e aprendizado profundo em redes neurais dinâmicas.

Enunciado: Resolva a equação diferencial $\frac{dy}{dx} = -2y$ com a condição inicial $y(0) = 1$ no intervalo de 0 a 5.

Resolução:

```
from scipy.integrate import solve_ivp
import numpy as np

# Definir a equa  o diferencial
def eq_diferencial(t, y):
    return -2 * y

# Resolver a EDO com condi  o inicial y(0) = 1
solucao = solve_ivp(eq_diferencial, [0, 5], [1], t_eval=np.linspace
    (0, 5, 100))

# Exibir os resultados
print("Tempo:", solucao.t)
print("Solu  o para y:", solucao.y[0])
```

Listing 2: Resolução do Exemplo 2

Explicação:

- `solve_ivp`: Resolve equações diferenciais ordinárias (EDOs) com condições iniciais.
- Aqui, resolvemos $\frac{dy}{dx} = -2y$, uma equação comum em sistemas dinâmicos.
- `t_eval=np.linspace(0, 5, 100)`: Define 100 pontos no intervalo de tempo de 0 a 5.

Esse método é aplicado em sistemas que simulam mudanças ao longo do tempo, como simulações de IA em ambientes dinâmicos.

3 Exercício 3: Encontrar Raízes de Funções

Contexto: Em IA, encontrar raízes de funções é essencial para otimização de funções e ajuste de modelos. SciPy fornece métodos rápidos para encontrar essas soluções.

Enunciado: Encontre a raiz da função $f(x) = x^3 - 6x^2 + 11x - 6$ que está próxima de $x = 2$.

Resolução:

```
from scipy.optimize import root

# Definir a fun   o
def f(x):
    return x**3 - 6*x**2 + 11*x - 6

# Encontrar a raiz pr xima de x = 2
solucao = root(f, 2)
print("Raiz encontrada:", solucao.x)
```

Listing 3: Resolução do Exemplo 3

Explicação:

- `root(f, 2)`: Encontra a raiz da função $f(x) = 0$ partindo de uma aproximação inicial $x = 2$.

Esse método é utilizado para encontrar pontos críticos em funções de perda ou em algoritmos de otimização.

4 Exercício 4: Otimização de Funções

Contexto: A otimização é a base dos algoritmos de aprendizado de máquina. SciPy fornece métodos eficientes para encontrar mínimos de funções não lineares, essenciais no ajuste de parâmetros de modelos.

Enunciado: Encontre o mínimo da função $f(x) = x^2 + 4x + 4$.

Resolução:

```
from scipy.optimize import minimize

# Definir a fun   o
def f(x):
    return x**2 + 4*x + 4

# Encontrar o m nimo
resultado = minimize(f, 0)
print("M nimo encontrado em x =", resultado.x)
```

Listing 4: Resolução do Exemplo 4

Explicação:

- `minimize(f, 0)`: Busca o valor de x que minimiza a função, partindo de uma estimativa inicial $x = 0$.

Esse tipo de otimização é fundamental em algoritmos de aprendizado de máquina para ajustar parâmetros e minimizar funções de erro.

5 Exercício 5: Transformada de Fourier

Contexto: A transformada de Fourier é usada para decompor sinais no domínio da frequência, sendo fundamental em processamento de sinais, análise de séries temporais e redes neurais convolucionais.

Enunciado: Calcule a transformada de Fourier para o sinal $f(t) = \sin(2\pi t) + 0.5\sin(4\pi t)$ no intervalo de 0 a 1.

Resolução:

```
from scipy.fft import fft
import numpy as np

# Definir o sinal
t = np.linspace(0, 1, 500)
sinal = np.sin(2 * np.pi * t) + 0.5 * np.sin(4 * np.pi * t)

# Calcular a transformada de Fourier
frequencias = fft(sinal)
print("Transformada de Fourier:", frequencias)
```

Listing 5: Resolução do Exemplo 5

Explicação:

- `fft(sinal)`: Calcula a transformada de Fourier do sinal $f(t)$.

Este método é amplamente utilizado em IA para análise de padrões e componentes frequenciais em dados temporais.

6 Exercício 6: Interpolação de Dados

Contexto: Em IA, pode ser necessário reconstruir dados ou gerar novos pontos a partir de amostras conhecidas. SciPy oferece métodos para interpolação de dados, útil em preenchimento de dados ausentes.

Enunciado: Dado o conjunto de pontos $x = [0, 1, 2, 3, 4]$ e $y = [1, 2, 0, 2, 1]$, encontre os valores interpolados para $x = [0.5, 1.5, 2.5, 3.5]$.

Resolução:

```
from scipy import interpolate
import numpy as np

# Definir os pontos
x = np.array([0, 1, 2, 3, 4])
y = np.array([1, 2, 0, 2, 1])

# Criar a fun   o de interpola  o
f_interpolada = interpolate.interp1d(x, y, kind='linear')

# Encontrar os valores interpolados
x_novos = np.array([0.5, 1.5, 2.5, 3.5])
y_novos = f_interpolada(x_novos)
print("Valores interpolados:", y_novos)
```

Listing 6: Resolução do Exemplo 6

Explicação:

- `interpolate.interp1d`: Cria uma função de interpolação linear com base nos pontos dados.

Esse método é útil para preenchimento de dados faltantes ou para suavização de dados.

7 Exercício 7: Resolução de Sistemas de Equações Lineares

Contexto: Resolver sistemas de equações lineares é uma tarefa fundamental em IA, especialmente em métodos de regressão linear e ajuste de modelos.

Enunciado: Resolva o sistema de equações lineares:

$$2x + y = 5$$

$$x + 3y = 7$$

usando a função `linalg.solve` do SciPy.

Resolução:

```
from scipy import linalg
import numpy as np

# Definir os coeficientes das equa  es
A = np.array([[2, 1], [1, 3]])
B = np.array([5, 7])

# Resolver o sistema de equa  es
solucao = linalg.solve(A, B)
print("Solu  o [x, y]:", solucao)
```

Listing 7: Resolução do Exemplo 7

Explicação:

- `linalg.solve(A, B)`: Resolve sistemas lineares da forma $Ax = B$.

Esse método é utilizado em otimização e regressão em IA.

8 Exercício 8: Ajuste de Curvas Usando SciPy

Contexto: O ajuste de curvas é importante para modelagem e regressão em aprendizado de máquina, permitindo encontrar a melhor função que se ajusta a um conjunto de dados.

Enunciado: Encontre a curva que melhor se ajusta aos pontos $x = [0, 1, 2, 3, 4]$ e $y = [1, 2, 0, 2, 1]$, assumindo um ajuste de uma parábola $y = ax^2 + bx + c$.

Resolução:

```
from scipy.optimize import curve_fit
import numpy as np

# Definir os pontos
x = np.array([0, 1, 2, 3, 4])
y = np.array([1, 2, 0, 2, 1])

# Definir a fun   o para ajuste (par bola)
def parabola(x, a, b, c):
    return a * x**2 + b * x + c

# Ajustar a curva
parametros, _ = curve_fit(parabola, x, y)

print("Par metros ajustados [a, b, c]:", parametros)
```

Listing 8: Resolução do Exemplo 8

Explicação:

- `curve_fit`: Ajusta a curva com base nos dados fornecidos e na função modelo. Aqui ajustamos uma parábola.

Esse ajuste é usado em problemas de regressão em IA para encontrar modelos que melhor representem os dados.

9 Exercício 9: Solução de Equações Não Lineares

Contexto: Alguns problemas em IA requerem a resolução de equações não lineares, como em modelos de otimização não convexos.

Enunciado: Encontre a raiz da equação $\cos(x) - x = 0$ usando SciPy.

Resolução:

```
from scipy.optimize import fsolve
import numpy as np

# Definir a equa   o
def equacao(x):
    return np.cos(x) - x

# Encontrar a raiz
raiz = fsolve(equacao, 0.5)
print("Raiz encontrada:", raiz)
```

Listing 9: Resolução do Exemplo 9

Explicação:

- `fsolve`: Resolve equações não lineares como $\cos(x) - x = 0$.

Esse método é útil em problemas de otimização e ajuste em IA.

10 Exercício 10: Minimização de Funções Multivariadas

Contexto: Minimização de funções multivariadas é essencial em aprendizado de máquina para ajuste de parâmetros de redes neurais e modelos de regressão.

Enunciado: Minimize a função $f(x, y) = (x - 2)^2 + (y - 3)^2$.

Resolução:

```
from scipy.optimize import minimize

# Definir a fun   o a ser minimizada
def funcao(v):
    x, y = v
    return (x - 2)**2 + (y - 3)**2

# Encontrar o m nimo
resultado = minimize(funcao, [0, 0])
print("M nimo encontrado em [x, y] =", resultado.x)
```

Listing 10: Resolução do Exemplo 10

Explicação:

- `minimize`: Minimiza funções multivariadas, crucial para encontrar o ajuste ótimo de parâmetros em IA.

Esse método é amplamente utilizado no treinamento de modelos de aprendizado de máquina.

Apresentação da Biblioteca Pandas

Prof. Dr. Fabiano B. Menegidio

1 Introdução

Pandas é uma biblioteca de código aberto para manipulação e análise de dados em Python. Ela fornece estruturas de dados rápidas, flexíveis e expressivas, projetadas para facilitar o trabalho com dados estruturados e rótulados, como tabelas de banco de dados e planilhas. A biblioteca Pandas é amplamente utilizada em ciência de dados, estatística e engenharia de software, especialmente em processos de análise, limpeza e transformação de dados.

Ao oferecer duas estruturas principais – **Series** e **DataFrame** – o Pandas permite manipular e processar grandes volumes de dados de maneira eficiente, oferecendo várias funcionalidades para leitura, filtragem, agregação e manipulação de dados.

2 Por que usar Pandas?

Pandas é uma das bibliotecas mais populares e fundamentais para o trabalho com dados em Python. Sua principal força reside na capacidade de trabalhar com grandes conjuntos de dados de forma eficiente, facilitando operações complexas, como fusão de dados, agrupamento e manipulação de colunas e índices.

Pandas é amplamente usado em áreas como *ciência de dados*, *machine learning*, *análise financeira*, *estatísticas* e até mesmo *visualização de dados*, integrando-se perfeitamente com bibliotecas como *Matplotlib* e *Seaborn*.

3 Principais Utilizações do Pandas

- **Manipulação de Dados Tabulares**: Pandas oferece o DataFrame, uma estrutura de dados tabular, similar a uma tabela SQL ou uma planilha Excel.
- **Leitura e Escrita de Arquivos**: Com Pandas, é possível ler e gravar arquivos de diversos formatos (CSV, Excel, JSON, SQL, etc.).
- **Análise de Dados Estatísticos**: A biblioteca fornece funções para calcular estatísticas descritivas, realizar agrupamentos, pivotagem e outras operações fundamentais.

- **Limpeza de Dados**: Pandas oferece métodos para tratamento de dados ausentes, substituição de valores e manipulação de strings.
- **Integração com Outras Ferramentas**: Pandas pode ser usado junto a bibliotecas como NumPy, Matplotlib e Scikit-learn.

4 Estrutura Básica do Pandas

O Pandas fornece duas estruturas de dados principais:

- **Series**: Um objeto unidimensional semelhante a uma lista ou vetor.
- **DataFrame**: Uma estrutura bidimensional (tabela) que permite armazenar e manipular dados em linhas e colunas com rótulos (índices).

4.1 Series

Uma `Series` é um array unidimensional que pode armazenar qualquer tipo de dado (inteiros, strings, floats, objetos Python). É semelhante a um array do NumPy, mas com um índice associado a cada elemento.

Exemplo de criação de uma Series:

```
import pandas as pd

# Criar uma Series
s = pd.Series([1, 2, 3, 4, 5], index=['a', 'b', 'c', 'd', 'e'])
print(s)
```

4.2 DataFrame

O `DataFrame` é a estrutura de dados mais poderosa do Pandas. Ele é uma tabela bidimensional com rótulos para linhas e colunas. Cada coluna pode conter diferentes tipos de dados (inteiros, floats, strings, etc.).

Exemplo de criação de um DataFrame:

```
import pandas as pd

# Criar um DataFrame
dados = {'Nome': ['Ana', 'Bruno', 'Carlos', 'Diana'],
         'Idade': [23, 35, 45, 28],
         'Cidade': ['S o Paulo', 'Rio de Janeiro', 'Belo Horizonte
    ', 'Curitiba']}
df = pd.DataFrame(dados)
print(df)
```

5 Principais Funções do Pandas

O Pandas oferece uma grande variedade de funções para trabalhar com dados. Abaixo estão algumas das funções mais importantes e amplamente utilizadas.

5.1 1. Leitura de Dados com Pandas

Pandas facilita a leitura de dados a partir de vários formatos de arquivo. Algumas das funções mais utilizadas incluem:

- `pd.read_csv()`: Leitura de arquivos CSV.
- `pd.read_excel()`: Leitura de arquivos Excel.
- `pd.read_json()`: Leitura de arquivos JSON.
- `pd.read_sql()`: Leitura de dados a partir de uma consulta SQL.

Exemplo de leitura de um arquivo CSV:

```
import pandas as pd

# Ler dados de um arquivo CSV
df = pd.read_csv('dados.csv')
print(df.head())  # Exibir as primeiras 5 linhas do DataFrame
```

5.2 2. Seleção e Fatiamento de Dados

Pandas facilita a seleção de linhas e colunas de um DataFrame de forma eficiente.

Exemplo de seleção de dados:

```
# Selecionar a coluna 'Nome'
nomes = df['Nome']

# Selecionar as primeiras duas linhas e a coluna 'Cidade'
primeiras_duas = df.loc[:1, 'Cidade']
```

5.3 3. Manipulação de Colunas e Linhas

Pandas permite adicionar, remover ou modificar colunas e linhas de um DataFrame de maneira fácil.

Exemplo de adição e remoção de colunas:

```
# Adicionar uma nova coluna 'Sal rio'
df['Sal rio'] = [5000, 6000, 7000, 8000]

# Remover a coluna 'Cidade'
df = df.drop('Cidade', axis=1)
```

5.4 4. Agrupamento e Agregação

Pandas oferece funções poderosas para agrupar dados e calcular agregações.

Exemplo de agrupamento:

```
# Agrupar por 'Cidade' e calcular a m dia de 'Idade'
media_idades = df.groupby('Cidade')['Idade'].mean()
print(media_idades)
```

5.5 5. Tratamento de Dados Ausentes

Pandas oferece métodos eficientes para lidar com dados ausentes.

Exemplo de preenchimento de valores ausentes:

```
# Preencher valores ausentes na coluna 'Sal rio' com a m dia da coluna
df['Sal rio'] = df['Sal rio'].fillna(df['Sal rio'].mean())
```

6 Conclusão

O Pandas é uma biblioteca indispensável para quem trabalha com manipulação e análise de dados em Python. Suas estruturas de dados, como Series e DataFrame, permitem a manipulação eficiente de grandes volumes de dados tabulares. Com uma ampla gama de funções para leitura, transformação, limpeza, agrupamento e agregação de dados, o Pandas facilita o trabalho com dados estruturados e é amplamente utilizado em ciência de dados, aprendizado de máquina e análise de negócios.

Exercícios Resolvidos de Pandas

Prof. Dr. Fabiano B. Menegidio

1 Exercício 1: Criação de DataFrames

Contexto: A manipulação de dados tabulares é fundamental para análise de dados em áreas como ciência de dados e aprendizado de máquina. Pandas facilita a criação e manipulação de DataFrames, que são estruturas essenciais para lidar com grandes volumes de dados.

Enunciado: Crie um DataFrame contendo os nomes, idades e cidades de 4 pessoas.

Resolução:

```
import pandas as pd

# Criar um DataFrame
dados = {'Nome': ['Ana', 'Bruno', 'Carlos', 'Diana'],
         'Idade': [23, 35, 45, 28],
         'Cidade': ['S o Paulo', 'Rio de Janeiro', 'Belo Horizonte
    ', 'Curitiba']}
df = pd.DataFrame(dados)
print(df)
```

Listing 1: Resolução do Exemplo 1

Explicação:

- `pd.DataFrame(dados)`: Cria um DataFrame a partir de um dicionário onde as chaves representam os nomes das colunas e os valores são listas de dados correspondentes.

Os DataFrames são amplamente usados em ciência de dados para armazenar, manipular e analisar dados tabulares.

2 Exercício 2: Leitura de Arquivos CSV

Contexto: A leitura de arquivos CSV é uma das operações mais comuns em análise de dados. Pandas oferece métodos eficientes para carregar dados de arquivos CSV em DataFrames.

Enunciado: Leia um arquivo CSV chamado `dados.csv` e exiba as primeiras 5 linhas.

Resolução:

```
import pandas as pd

# Ler dados de um arquivo CSV
df = pd.read_csv('dados.csv')
print(df.head())  # Exibir as primeiras 5 linhas do DataFrame
```

Listing 2: Resolução do Exemplo 2

Explicação:

- `pd.read_csv('dados.csv')`: Lê um arquivo CSV e o carrega em um DataFrame.
- `df.head()`: Exibe as primeiras 5 linhas do DataFrame.

Esse método é amplamente utilizado para carregar conjuntos de dados e realizar análises exploratórias.

3 Exercício 3: Seleção de Colunas e Linhas

Contexto: Uma das operações fundamentais ao trabalhar com DataFrames é selecionar e manipular colunas e linhas específicas. Isso facilita a análise de subsets de dados para avaliações detalhadas.

Enunciado: A partir do DataFrame criado no exercício 1, selecione a coluna `Nome` e as duas primeiras linhas.

Resolução:

```
# Selecionar a coluna 'Nome'
nomes = df['Nome']
print(nomes)

# Selecionar as duas primeiras linhas do DataFrame
primeiras_duas = df.iloc[:2]
print(primeiras_duas)
```

Listing 3: Resolução do Exemplo 3

Explicação:

- `df['Nome']`: Seleciona a coluna `Nome` do DataFrame.
- `df.iloc[:2]`: Seleciona as duas primeiras linhas do DataFrame utilizando a indexação numérica com o `iloc`.

Essas operações são essenciais para filtragem de dados em grandes conjuntos.

4 Exercício 4: Adição de Colunas

Contexto: Em muitos casos, é necessário adicionar novas colunas a um DataFrame para incluir variáveis derivadas ou calculadas com base em outras colunas.

Enunciado: Adicione uma coluna `Salário` ao DataFrame criado no exercício 1, com os valores $[5000, 6000, 7000, 8000]$.

Resolução:

```
# Adicionar uma nova coluna 'Sal rio'
df['Sal rio'] = [5000, 6000, 7000, 8000]
print(df)
```

Listing 4: Resolução do Exemplo 4

Explicação:

- `df['Salário'] = [5000, 6000, 7000, 8000]`: Adiciona uma nova coluna ao DataFrame com os valores especificados.

A adição de colunas é frequentemente usada em processamento de dados para enriquecer os dados com novas informações.

5 Exercício 5: Remoção de Colunas

Contexto: A remoção de colunas desnecessárias é uma tarefa comum na limpeza de dados, especialmente quando se trabalha com grandes conjuntos de dados que contêm informações irrelevantes para a análise.

Enunciado: Remova a coluna `Cidade` do DataFrame.

Resolução:

```
# Remover a coluna 'Cidade'
df = df.drop('Cidade', axis=1)
print(df)
```

Listing 5: Resolução do Exemplo 5

Explicação:

- `df.drop('Cidade', axis=1)`: Remove a coluna `Cidade` do DataFrame. O argumento `axis=1` indica que estamos removendo uma coluna, não uma linha.

A remoção de colunas é importante para otimização de performance e foco nos dados relevantes.

6 Exercício 6: Filtragem de Dados

Contexto: Filtrar dados com base em condições específicas é uma operação essencial na análise de dados, permitindo focar apenas em subconjuntos relevantes do DataFrame.

Enunciado: Filtre o DataFrame para exibir apenas as pessoas com idade maior que 30 anos.

Resolução:

```
# Filtrar o DataFrame para idades maiores que 30
df_filtrado = df[df['Idade'] > 30]
print(df_filtrado)
```

Listing 6: Resolução do Exemplo 6

Explicação:

- `df[df['Idade'] > 30]`: Filtra o DataFrame, retornando apenas as linhas onde o valor da coluna `Idade` é maior que 30.

Essa técnica é amplamente utilizada para realizar análises específicas em subconjuntos de dados.

7 Exercício 7: Agrupamento de Dados

Contexto: Agrupar dados por categorias e calcular agregações é fundamental para relatórios e análises estatísticas em grandes conjuntos de dados.

Enunciado: A partir do DataFrame criado no exercício 1, agrupe os dados por cidade e calcule a média de idade de cada cidade.

Resolução:

```
# Agrupar por 'Cidade' e calcular a m dia de 'Idade'
media_idades = df.groupby('Cidade')['Idade'].mean()
print(media_idades)
```

Listing 7: Resolução do Exemplo 7

Explicação:

- `df.groupby('Cidade')['Idade'].mean()`: Agrupa os dados por `Cidade` e calcula a média da coluna `Idade` para cada grupo.

Essa técnica é útil em análises de dados categóricos e geração de relatórios estatísticos.

8 Exercício 8: Ordenação de Dados

Contexto: A ordenação de dados é uma operação comum para análises, relatórios e visualização de dados, permitindo organizar os dados em ordem crescente ou decrescente.

Enunciado: Ordene o DataFrame criado no exercício 1 pela coluna `Idade` de forma crescente.

Resolução:

```
# Ordenar o DataFrame pela coluna 'Idade'
df_ordenado = df.sort_values(by='Idade')
print(df_ordenado)
```

Listing 8: Resolução do Exemplo 8

Explicação:

- `df.sort_values(by='Idade')`: Ordena o DataFrame pela coluna `Idade` em ordem crescente. O argumento `by` especifica a coluna usada para a ordenação.

A ordenação é frequentemente usada para analisar tendências ou preparar dados para relatórios.

9 Exercício 9: Tratamento de Dados Ausentes

Contexto: O tratamento de dados ausentes (missing data) é uma tarefa crucial na limpeza de dados, pois valores ausentes podem distorcer análises e modelos de aprendizado de máquina.

Enunciado: No DataFrame criado no exercício 1, substitua qualquer valor ausente na coluna `Salário` pela média dessa coluna.

Resolução:

```
# Preencher valores ausentes na coluna 'Sal rio' com a m dia da
    coluna
df['Sal rio'] = df['Sal rio'].fillna(df['Sal rio'].mean())
print(df)
```

Listing 9: Resolução do Exemplo 9

Explicação:

- `df['Salário'].fillna(df['Salário'].mean())`: Substitui os valores ausentes na coluna `Salário` pela média dessa coluna.

Esse método é amplamente utilizado para evitar problemas causados por dados faltantes em modelos de aprendizado de máquina.

10 Exercício 10: Criação de Pivot Table

Contexto: A criação de pivot tables é essencial para sumarização de dados categóricos e quantitativos. Em Pandas, a função `pivot_table` permite gerar essas tabelas facilmente, agregando dados com base em categorias.

Enunciado: Crie uma tabela dinâmica (pivot table) a partir do DataFrame criado no exercício 1, mostrando a média de idade agrupada por cidade.

Resolução:

```
# Criar uma pivot table com a m dia de idade por cidade
pivot_table = df.pivot_table(values='Idade', index='Cidade',
    aggfunc='mean')
print(pivot_table)
```

Listing 10: Resolução do Exemplo 10

Explicação:

- `df.pivot_table(values='Idade', index='Cidade', aggfunc='mean')`: Cria uma tabela dinâmica (pivot table) mostrando a média da coluna `Idade` para cada `Cidade`.

As pivot tables são usadas para gerar relatórios e sumarizar grandes volumes de dados categóricos.

Apresentação da Biblioteca Matplotlib

Prof. Dr. Fabiano B. Menegidio

1 Introdução

Matplotlib é uma biblioteca de visualização de dados em Python, amplamente utilizada para a criação de gráficos 2D. Ela oferece uma interface flexível e rica para criar gráficos interativos e estáticos, sendo uma das ferramentas principais para visualização de dados em *ciência de dados*, *análise de dados* e *engenharia de software*.

Matplotlib permite a criação de uma ampla gama de gráficos, como gráficos de linha, dispersão, barras, histogramas, gráficos de área e muito mais.

2 Por que usar Matplotlib?

Matplotlib é a biblioteca padrão para visualização de dados em Python, devido à sua simplicidade e flexibilidade. Além disso, o Matplotlib se integra perfeitamente com outras bibliotecas, como *NumPy*, *Pandas* e *Seaborn*, permitindo visualizações diretas a partir de estruturas de dados como arrays e DataFrames.

3 Principais Utilizações do Matplotlib

- **Gráficos de Linhas**: Usado para visualizar séries temporais ou qualquer conjunto de dados contínuos.
- **Gráficos de Dispersão**: Utilizados para observar a relação entre duas variáveis em um gráfico de pontos.
- **Histogramas**: Ideal para visualização de distribuições de frequências.
- **Gráficos de Barras**: Usado para comparar diferentes categorias ou grupos.
- **Gráficos de Pizza**: Utilizado para representar proporções de um todo.

4 Estrutura Básica do Matplotlib

A estrutura principal do Matplotlib é baseada no conceito de figuras e eixos.

- **Figura**: É a janela ou página em que os gráficos são desenhados.
- **Eixos**: São os espaços em que os dados são plotados dentro de uma figura.

5 Principais Funções do Matplotlib

O Matplotlib oferece uma ampla gama de funções para a criação de gráficos.

5.1 1. Gráficos de Linhas

Exemplo básico de um gráfico de linha:

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

# Gerar dados
x = np.linspace(0, 10, 100)
y = np.sin(x)

# Criar o gr fico de linha
plt.plot(x, y)
plt.title("Gr fico de Linha")
plt.xlabel("Eixo X")
plt.ylabel("Eixo Y")
plt.show()
```

5.2 2. Gráficos de Dispersão

Exemplo de gráfico de dispersão:

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

# Gerar dados
x = np.random.rand(50)
y = np.random.rand(50)

# Criar o gr fico de dispers o
plt.scatter(x, y)
plt.title("Gr fico de Dispers o")
plt.xlabel("Eixo X")
plt.ylabel("Eixo Y")
plt.show()
```

5.3 3. Histogramas

Exemplo de histograma:

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

# Gerar dados
dados = np.random.randn(1000)
```

```

# Criar o histograma
plt.hist(dados, bins=30)
plt.title("Histograma")
plt.xlabel("Valor")
plt.ylabel("Frequ ncia")
plt.show()
```

5.4 4. Gráficos de Barras

Exemplo de gráfico de barras:

```
import matplotlib.pyplot as plt

# Dados
categorias = ['A', 'B', 'C', 'D']
valores = [3, 7, 5, 9]

# Criar o gr fico de barras
plt.bar(categorias, valores)
plt.title("Gr fico de Barras")
plt.xlabel("Categorias")
plt.ylabel("Valores")
plt.show()
```

5.5 5. Gráficos de Pizza

Exemplo de gráfico de pizza:

```
import matplotlib.pyplot as plt

# Dados
labels = ['A', 'B', 'C', 'D']
sizes = [15, 30, 45, 10]

# Criar o gr fico de pizza
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90)
plt.title("Gr fico de Pizza")
plt.show()
```

6 Personalização de Gráficos

O Matplotlib oferece uma vasta gama de opções para personalizar gráficos.
Exemplo de personalização de um gráfico:

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

# Gerar dados
x = np.linspace(0, 10, 100)
y = np.sin(x)

# Criar o gr fico de linha com personaliza  o
```

```
plt.plot(x, y, color='green', linestyle='--', marker='o')
plt.title("Gr fico Personalizado")
plt.xlabel("Eixo X")
plt.ylabel("Eixo Y")
plt.grid(True)  # Adicionar grade
plt.show()
```

7 Conclusão

O Matplotlib é uma biblioteca essencial para visualização de dados em Python. Ele oferece uma grande flexibilidade para criar gráficos de alta qualidade, que podem ser simples ou altamente personalizados. A capacidade de criar gráficos interativos ou estáticos e a integração com bibliotecas como NumPy e Pandas tornam o Matplotlib uma ferramenta fundamental para cientistas de dados e engenheiros.

Exercícios Resolvidos de Matplotlib

Prof. Dr. Fabiano B. Menegidio

1 Exercício 1: Criação de Gráfico de Linhas Simples

Contexto: Gráficos de linha são usados para exibir dados contínuos ao longo de um intervalo de tempo ou para mostrar a evolução de variáveis em um eixo de coordenadas. É amplamente utilizado em ciência de dados e análise temporal.

Enunciado: Crie um gráfico de linha para a função $y = \sin(x)$, com valores de x variando de 0 a 10.

Resolução:

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

\# Gerar dados
x = np.linspace(0, 10, 100)
y = np.sin(x)

\# Criar o gr fico de linha
plt.plot(x, y)
plt.title("Gr fico de Linha - Fun   o Seno")
plt.xlabel("Eixo X")
plt.ylabel("Eixo Y")
plt.show()
```

Listing 1: Resolução do Exemplo 1

Explicação:

- `plt.plot(x, y)`: Cria um gráfico de linha com os valores de x e y.
- `plt.title()`, `plt.xlabel()`, `plt.ylabel()`: Adicionam título e rótulos aos eixos do gráfico.
- `plt.show()`: Exibe o gráfico gerado.

Gráficos de linha são utilizados para visualizar tendências de dados contínuos, como séries temporais.

2 Exercício 2: Gráfico de Dispersão

Contexto: Gráficos de dispersão (scatter plots) são amplamente usados para mostrar a relação entre duas variáveis, como em problemas de regressão linear ou correlação.

Enunciado: Gere um gráfico de dispersão para 50 pontos aleatórios, com coordenadas x e y geradas aleatoriamente.

Resolução:

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

\# Gerar dados aleat rios
x = np.random.rand(50)
y = np.random.rand(50)

\# Criar o gr fico de dispers o
plt.scatter(x, y)
plt.title("Gr fico de Dispers o")
plt.xlabel("Eixo X")
plt.ylabel("Eixo Y")
plt.show()
```

Listing 2: Resolução do Exemplo 2

Explicação:

- `plt.scatter(x, y)`: Cria um gráfico de dispersão com os pontos x e y.
- Esse tipo de gráfico é útil para observar padrões ou correlações entre variáveis.

Gráficos de dispersão são frequentemente usados para análise de correlação em ciência de dados e aprendizado de máquina.

3 Exercício 3: Criação de Histogramas

Contexto: Histogramas são usados para mostrar a distribuição de uma variável em diferentes intervalos (bins). É uma ferramenta importante na análise de dados estatísticos, como a verificação de normalidade.

Enunciado: Crie um histograma para 1000 números gerados aleatoriamente a partir de uma distribuição normal.

Resolução:

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

\# Gerar dados aleat rios de uma distribui  o normal
dados = np.random.randn(1000)

\# Criar o histograma
plt.hist(dados, bins=30, edgecolor='black')
plt.title("Histograma de Distribui  o Normal")
plt.xlabel("Valores")
plt.ylabel("Frequ ncia")
plt.show()
```

Listing 3: Resolução do Exemplo 3

Explicação:

- `plt.hist(dados, bins=30)`: Cria um histograma com 30 intervalos (bins) a partir dos dados gerados.
- `edgecolor='black'`: Define uma borda preta ao redor das barras do histograma.

Histogramas são essenciais para entender a distribuição de dados em análise exploratória.

4 Exercício 4: Gráfico de Barras

Contexto: Gráficos de barras são úteis para comparar diferentes categorias ou grupos de dados. Eles são amplamente usados em relatórios, comparações de valores categóricos e visualizações de dados discretos.

Enunciado: Crie um gráfico de barras para 4 categorias (`A`, `B`, `C`, `D`) com valores correspondentes $[3, 7, 5, 9]$.

Resolução:

```
import matplotlib.pyplot as plt

\# Dados
categorias = ['A', 'B', 'C', 'D']
valores = [3, 7, 5, 9]

\# Criar o gr fico de barras
plt.bar(categorias, valores, color='blue')
plt.title("Gr fico de Barras")
plt.xlabel("Categorias")
plt.ylabel("Valores")
plt.show()
```

Listing 4: Resolução do Exemplo 4

Explicação:

- `plt.bar(categorias, valores)`: Cria um gráfico de barras com as categorias e seus respectivos valores.
- `color='blue'`: Define a cor das barras.

Gráficos de barras são amplamente utilizados para comparar dados categóricos e discretos.

5 Exercício 5: Gráfico de Pizza

Contexto: Gráficos de pizza são usados para mostrar a proporção de partes de um todo. Eles são comumente utilizados em apresentações e relatórios para representar distribuições proporcionais.

Enunciado: Crie um gráfico de pizza para representar as proporções de quatro categorias: `A`, `B`, `C`, `D`, com tamanhos correspondentes $[15, 30, 45, 10]$.

Resolução:

```
import matplotlib.pyplot as plt

\# Dados
labels = ['A', 'B', 'C', 'D']
sizes = [15, 30, 45, 10]

\# Criar o gr fico de pizza
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90)
plt.title("Gr fico de Pizza")
plt.show()
```

Listing 5: Resolução do Exemplo 5

Explicação:

- `plt.pie(sizes, labels=labels, autopct='%1.1f%')`: Cria um gráfico de pizza com os tamanhos e rótulos das categorias. O argumento `autopct` exibe as porcentagens e `startangle=90` ajusta o ângulo inicial do gráfico.

Gráficos de pizza são usados para representar proporções de categorias em um todo.

6 Exercício 6: Personalização de Gráficos de Linhas

Contexto: A personalização de gráficos é crucial para apresentações e relatórios profissionais. Matplotlib permite personalizar cores, estilos de linha, marcadores e outros aspectos visuais.

Enunciado: Crie um gráfico de linha para a função $y = \cos(x)$, personalizando a cor da linha para verde, o estilo da linha para tracejada, e usando marcadores circulares nos pontos.

Resolução:

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

\# Gerar dados
x = np.linspace(0, 10, 100)
y = np.cos(x)

\# Criar o gr fico de linha personalizado
plt.plot(x, y, color='green', linestyle='--', marker='o')
plt.title("Gr fico Personalizado - Fun   o Cosseno")
plt.xlabel("Eixo X")
plt.ylabel("Eixo Y")
plt.grid(True)  \# Adicionar uma grade
plt.show()
```

Listing 6: Resolução do Exemplo 6

Explicação:

- `plt.plot(x, y, color='green', linestyle='--', marker='o')`: Cria um gráfico de linha com cor verde, linha tracejada e marcadores circulares.
- `plt.grid(True)`: Adiciona uma grade ao gráfico.

A personalização de gráficos é amplamente utilizada em visualizações para destacar padrões e tornar gráficos mais legíveis e esteticamente agradáveis.

7 Exercício 7: Subplots - Múltiplos Gráficos em Uma Figura

Contexto: Subplots são úteis para visualizar múltiplos gráficos em uma única figura, facilitando comparações entre diferentes visualizações de dados.

Enunciado: Crie dois gráficos de linha em uma mesma figura. O primeiro gráfico deve ser para $y = \sin(x)$ e o segundo para $y = \cos(x)$.

Resolução:

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

\# Gerar dados
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)

\# Criar subplots
plt.subplot(2, 1, 1)  \# Primeiro gr fico (2 linhas, 1 coluna, 1
     gr fico)
plt.plot(x, y1)
plt.title("Gr fico de Seno")

plt.subplot(2, 1, 2)  \# Segundo gr fico (2 linhas, 1 coluna, 2
    gr fico)
plt.plot(x, y2)
plt.title("Gr fico de Cosseno")

plt.tight_layout()  \# Ajustar layout para evitar sobreposi  o
plt.show()
```

Listing 7: Resolução do Exemplo 7

Explicação:

- `plt.subplot(2, 1, 1)`: Cria o primeiro gráfico em uma grade de 2 linhas e 1 coluna.
- `plt.subplot(2, 1, 2)`: Cria o segundo gráfico.
- `plt.tight_layout()`: Ajusta automaticamente o layout dos gráficos para evitar sobreposição de títulos e eixos.

Os subplots são amplamente utilizados em comparações de múltiplos conjuntos de dados em uma única visualização.

8 Exercício 8: Gráfico de Linhas com Múltiplas Séries

Contexto: Visualizar múltiplas séries de dados em um mesmo gráfico de linhas é útil para comparar diferentes variáveis em uma mesma escala.

Enunciado: Crie um gráfico de linha exibindo duas séries, $y1 = \sin(x)$ e $y2 = \cos(x)$, no mesmo gráfico.

Resolução:

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

\# Gerar dados
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)

\# Criar o gr fico de linha com duas s ries
plt.plot(x, y1, label='Seno')
plt.plot(x, y2, label='Cosseno')
plt.title("Gr fico de Linhas com M ltiplas S ries")
plt.xlabel("Eixo X")
plt.ylabel("Eixo Y")
plt.legend()  \# Adicionar legenda para identificar as s ries
plt.show()
```

Listing 8: Resolução do Exemplo 8

Explicação:

- `plt.plot(x, y1, label='Seno')`: Plota a primeira série com o rótulo "Seno".
- `plt.plot(x, y2, label='Cosseno')`: Plota a segunda série com o rótulo "Cosseno".
- `plt.legend()`: Adiciona uma legenda ao gráfico para identificar as séries.

Gráficos de linhas com múltiplas séries são comumente usados para comparações diretas entre diferentes conjuntos de dados.

9 Exercício 9: Gráfico de Linhas com Limites de Eixos

Contexto: Ajustar os limites dos eixos é uma funcionalidade importante para focar em uma parte específica dos dados ou melhorar a visualização de gráficos.

Enunciado: Crie um gráfico de linha para $y = \tan(x)$ com x variando de $-\pi$ a π. Defina os limites dos eixos y entre -10 e 10.

Resolução:

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

\# Gerar dados
x = np.linspace(-np.pi, np.pi, 500)
y = np.tan(x)

\# Criar o gr fico de linha
plt.plot(x, y)
plt.title("Gr fico de Linha - Fun   o Tangente")
plt.xlim([-np.pi, np.pi])  \# Definir limites do eixo X
plt.ylim([-10, 10])        \# Definir limites do eixo Y
plt.xlabel("Eixo X")
plt.ylabel("Eixo Y")
plt.grid(True)
plt.show()
```

Listing 9: Resolução do Exemplo 9

Explicação:

- `plt.xlim([-np.pi, np.pi])`: Define os limites do eixo x entre $-\pi$ e π.
- `plt.ylim([-10, 10])`: Define os limites do eixo y entre -10 e 10.

O ajuste dos limites de eixos é importante para focar em áreas de interesse ou para garantir que os dados sejam visualizados corretamente.

10 Exercício 10: Gráfico de Barras Empilhadas

Contexto: Gráficos de barras empilhadas são usados para mostrar a composição de diferentes categorias, acumulando os valores em uma barra única.

Enunciado: Crie um gráfico de barras empilhadas para duas categorias (`A`, `B`) e dois grupos de valores ($[3, 5]$ e $[2, 4]$).

Resolução:

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

\# Dados
categorias = ['A', 'B']
valores1 = [3, 5]
valores2 = [2, 4]

\# Criar o gr fico de barras empilhadas
plt.bar(categorias, valores1, label='Grupo 1')
plt.bar(categorias, valores2, bottom=valores1, label='Grupo 2')

plt.title("Gr fico de Barras Empilhadas")
plt.xlabel("Categorias")
plt.ylabel("Valores")
plt.legend()  \# Adicionar legenda
plt.show()
```

Listing 10: Resolução do Exemplo 10

Explicação:

- `plt.bar(categorias, valores1)`: Plota as barras para o primeiro grupo de valores.
- `plt.bar(categorias, valores2, bottom=valores1)`: Plota as barras empilhadas para o segundo grupo, com base nas barras anteriores.

Gráficos de barras empilhadas são úteis para mostrar composições de dados acumulados por categoria.

Exercícios de NumPy, SciPy, Pandas e Matplotlib

Disciplina de Inteligência Artificial

Contents

1 Exercícios de NumPy **2**

2 Exercícios de SciPy **4**

3 Exercícios de Pandas **5**

4 Exercícios de Matplotlib **7**

1 Exercícios de NumPy

1. Crie um array unidimensional de 50 elementos contendo números inteiros aleatórios entre 10 e 100.
2. Gere uma matriz 4x4 com números inteiros de 1 a 16 e extraia a última linha.
3. Crie um array de zeros com comprimento 15 e substitua os valores da 5ª à 10ª posição por 1.
4. Gere um array de 100 números inteiros aleatórios e encontre a mediana e a moda dos valores.
5. Crie um array de 20 elementos contendo números entre 1 e 100 e substitua todos os valores múltiplos de 3 por -3.
6. Gere uma matriz 5x5 de números aleatórios entre 0 e 1 e encontre a soma de cada coluna.
7. Crie dois arrays de números aleatórios e encontre a correlação entre eles.
8. Gere uma matriz 3x3 com números inteiros aleatórios e eleve todos os seus elementos ao quadrado.
9. Crie um array com 10 números aleatórios e normalize seus valores entre 0 e 1.
10. Crie uma matriz identidade 6x6 e substitua a diagonal principal por valores crescentes de 1 a 6.
11. Gere um array de 100 números inteiros e encontre o valor que mais se repete no array.
12. Crie uma matriz 4x4 e substitua os valores da diagonal principal por 7.
13. Crie um array contendo os 10 primeiros números pares e calcule sua soma cumulativa.
14. Calcule a diferença entre dois arrays de números aleatórios de mesmo comprimento e encontre a média das diferenças.
15. Gere um array contendo 20 números e substitua todos os valores maiores que 50 por 50.
16. Crie um array unidimensional e reorganize seus valores em uma matriz 3x4.
17. Gere uma matriz 4x4 de números inteiros aleatórios e multiplique cada valor por sua posição na matriz.

18. Crie um array de 10 elementos contendo números entre 1 e 100 e substitua todos os valores múltiplos de 5 por 0.

19. Crie dois arrays de números inteiros aleatórios de tamanho 20 e calcule o ângulo entre eles (produto escalar normalizado).

20. Crie uma matriz de números inteiros e encontre o valor mínimo de cada linha.

2 Exercícios de SciPy

1. Calcule a integral da função $f(x) = x^2 \sin(x)$ no intervalo de 0 a π.
2. Resolva a equação diferencial $\frac{dy}{dx} = x - y$, com $y(0) = 1$, no intervalo de 0 a 10.
3. Encontre as raízes da função $f(x) = x^4 - 3x^3 + 2$ utilizando SciPy.
4. Use a função de otimização de SciPy para maximizar $f(x) = -x^2 + 4x + 1$.
5. Calcule a transformada de Fourier de um sinal composto de $\sin(2\pi t)$ e $\cos(4\pi t)$.
6. Encontre a matriz inversa de uma matriz 5x5 de números aleatórios.
7. Use a função de interpolação cúbica de SciPy para interpolar os dados $(0, 0), (1, 2), (2, 3), (3, 5), (4, 4)$.
8. Resolva o sistema de equações $3x + 2y = 5$ e $x + 4y = 6$.
9. Ajuste uma curva quadrática aos pontos $(1, 2), (2, 4), (3, 5), (4, 7), (5, 8)$.
10. Calcule a integral dupla da função $f(x, y) = xy + x^2$ no intervalo $[0, 2]$ para x e $[0, 1]$ para y.
11. Encontre os autovalores e autovetores de uma matriz 4x4 de números inteiros aleatórios.
12. Resolva a equação $e^x - x^2 = 0$ para encontrar a raiz mais próxima de $x = 1$.
13. Calcule o determinante de uma matriz 6x6 de números aleatórios entre -10 e 10.
14. Utilize a função de otimização multivariada para minimizar $f(x, y) = (x - 2)^2 + (y - 3)^2$.
15. Calcule a transformada de Fourier inversa de um sinal complexo gerado aleatoriamente.
16. Resolva a equação diferencial de segunda ordem $\frac{d^2y}{dx^2} + y = 0$, com condições iniciais $y(0) = 1$ e $y'(0) = 0$.
17. Encontre a raiz cúbica de 27 utilizando SciPy.
18. Interpole os valores de uma função $f(x) = x^2$ nos pontos $[1, 2, 3, 4, 5]$ utilizando spline cúbica.
19. Realize a transformada de Fourier de um sinal discreto composto de senos e cossenos.
20. Resolva um sistema de equações lineares com três variáveis utilizando SciPy.

3 Exercícios de Pandas

1. Crie um DataFrame contendo as colunas `Produto`, `Preço` e `Quantidade` com 5 produtos.
2. Leia um arquivo Excel chamado `vendas.xlsx` e exiba as 10 primeiras linhas.
3. Filtre um DataFrame para exibir apenas os valores onde a coluna `Preço` é maior que 100.
4. Adicione uma coluna chamada `Desconto` ao DataFrame com 10% de desconto para todos os produtos.
5. Remova as linhas do DataFrame que contenham valores nulos.
6. Ordene o DataFrame pela coluna `Produto` em ordem alfabética.
7. Crie uma nova coluna chamada `Total` que seja o produto entre as colunas `Preço` e `Quantidade`.
8. Agrupe o DataFrame pela coluna `Categoria` e calcule o total de produtos por categoria.
9. Exporte o DataFrame resultante para um arquivo CSV chamado `resultados.csv`.
10. Leia um arquivo CSV, renomeie as colunas e exiba as 5 últimas linhas do DataFrame.
11. Preencha os valores nulos de uma coluna numérica com a mediana dessa coluna.
12. Verifique quais colunas de um DataFrame contêm valores duplicados.
13. Crie uma tabela dinâmica a partir do DataFrame, calculando a soma de `Quantidade` por `Categoria`.
14. Mescle dois DataFrames baseados em uma coluna comum chamada `ID` utilizando um join.
15. Crie um DataFrame com dados faltantes e use a interpolação linear para preenchê-los.
16. Crie um gráfico de barras a partir de um DataFrame usando a biblioteca Pandas.
17. Adicione uma coluna ao DataFrame que calcule o imposto (5%) sobre o valor da coluna `Total`.
18. Exiba apenas as linhas de um DataFrame onde o nome do produto começa com a letra "B".

19. Crie um DataFrame a partir de um dicionário de listas e calcule a média de cada coluna numérica.

20. Agrupe os dados por uma coluna categórica e calcule a mediana de uma coluna numérica.

4 Exercícios de Matplotlib

1. Crie um gráfico de linha para a função $y = x^3$ no intervalo de $x = -10$ até $x = 10$.
2. Plote um gráfico de dispersão para 100 pontos aleatórios com uma paleta de cores baseada nos valores de y.
3. Gere um histograma para 500 números inteiros gerados aleatoriamente entre 0 e 100.
4. Crie um gráfico de barras para 5 categorias com valores diferentes, colorindo cada barra de uma cor diferente.
5. Faça um gráfico de pizza para representar as porcentagens de quatro setores de mercado com rótulos personalizados.
6. Plote dois gráficos de linha em subplots diferentes: $y_1 = e^x$ e $y_2 = \log(x)$.
7. Crie um gráfico de dispersão com tamanho dos pontos variando de acordo com os valores de y.
8. Plote um gráfico de linha com intervalos de confiança (sombramento) para a função $y = \sin(x)$.
9. Adicione título, rótulos e uma grade personalizada a um gráfico de linha.
10. Plote três gráficos de linha em um único gráfico, cada um com uma cor e estilo de linha diferentes.
11. Crie um gráfico de barras empilhadas para três categorias com três grupos de valores.
12. Plote um gráfico de linha com uma função quadrática e ajuste os limites do eixo para focar nos valores $x = -5$ a $x = 5$.
13. Gere um gráfico de dispersão 3D para 100 pontos aleatórios utilizando Matplotlib.
14. Crie um gráfico de barras horizontal para 4 categorias com valores gerados aleatoriamente.
15. Plote a função $y = \frac{1}{x}$ com x variando de 1 a 100.
16. Crie um gráfico de linha com títulos para cada eixo e adicione um rótulo de texto no ponto $x = 5$.
17. Plote um gráfico de linha e salve-o como um arquivo PDF.
18. Crie um gráfico de barras com valores positivos e negativos e use diferentes cores para cada tipo de valor.

19. Plote um gráfico de linha com duas séries e adicione uma legenda para diferenciá-las.

20. Adicione uma grade com espaçamento personalizado a um gráfico de linha para $y = x^2$.