|  |
| --- |
| **INFO 02** |
| **Curso**: UFCD 10793 |
| **UFCD/Módulo/Temática:**  UFCD 10793 **- Fundamentos de Python** |
| **Ação: 10793\_05/AG** |
| **Formador/a**: Sandra Liliana Meira de Oliveira |
| **Data**: |
| **Nome do Formando/a**: |

MatPlotLib

[1. Visualização de Dados com Matplotlib 4](#_Toc185612544)

[Capítulo 1: Fundamentos da Comunicação Visual – Cor, Gestalt, Percepção e Storytelling com Dados 4](#_Toc185612545)

[1.1 Teoria da Cor e Paletas Cromáticas 4](#_Toc185612546)

[1.2 Princípios da Gestalt Aplicados à Visualização de Dados 5](#_Toc185612547)

[1.3 Percepção Visual e Limites Humanos 5](#_Toc185612548)

[1.4 Storytelling com Dados 5](#_Toc185612549)

[1.5 Aprofundar o Conhecimento sobre Design de Informação, Análise Perceptual e Escolha de Paletas de Cor 5](#_Toc185612550)

[Capítulo 2: Tipos de Gráfico e a Sua Aplicação 5](#_Toc185612551)

[2.1 Gráficos de Linhas 6](#_Toc185612552)

[2.2 Gráficos de Barras 6](#_Toc185612553)

[2.3 Histogramas 6](#_Toc185612554)

[2.4 Gráficos de Pizza 6](#_Toc185612555)

[2.5 Gráficos de Dispersão (Scatter Plots) 6](#_Toc185612556)

[2.6 Boxplots e Violinos 6](#_Toc185612557)

[2.7 Mapas de Calor e Contornos 6](#_Toc185612558)

[2.8 Gráficos 3D 7](#_Toc185612559)

[Capítulo 3: Introdução ao Matplotlib 7](#_Toc185612560)

[3.1 O que é o Matplotlib? 7](#_Toc185612561)

[3.2 Instalação e Ambiente de Trabalho 7](#_Toc185612562)

[3.3 Figura, Eixos e Objectos Básicos 7](#_Toc185612563)

[3.4 Criar o Primeiro Gráfico 9](#_Toc185612564)

[3.5 Controlar Tamanho e Resolução 9](#_Toc185612565)

[Capítulo 4: Gráficos de Linhas, Pontos e Estilos 9](#_Toc185612566)

[4.1 Cores e Estilos de Linha 9](#_Toc185612567)

[4.2 Marcadores e Espessura de Linha 9](#_Toc185612568)

[4.3 Múltiplas Séries no Mesmo Gráfico 12](#_Toc185612569)

[4.4 Subplots 13](#_Toc185612570)

[Capítulo 5: Gráficos de Barras, Histogramas e Setores 16](#_Toc185612571)

[5.1 Gráficos de Barras 16](#_Toc185612572)

[5.2 Histogramas 17](#_Toc185612573)

[5.3 Gráficos Circulares 18](#_Toc185612574)

[Capítulo 6: Gráficos de Dispersão, Boxplots e Violino 20](#_Toc185612575)

[6.1 Gráficos de Dispersão (Scatter Plot) 20](#_Toc185612576)

[6.2 Boxplots 21](#_Toc185612577)

[6.3 Gráficos de Violino 22](#_Toc185612578)

[Capítulo 7: Personalização Avançada 23](#_Toc185612579)

[7.1 Anotações e Texto 23](#_Toc185612580)

[7.2 Formatação Avançada dos Eixos 24](#_Toc185612581)

[7.3 Estilos Temáticos 27](#_Toc185612582)

[7.4 Legendas, Títulos e Labels Personalizados 29](#_Toc185612583)

[Capítulo 8: Visualização em 3D, Contornos e Mapas de Calor 30](#_Toc185612584)

[8.1 Gráficos em 3D 30](#_Toc185612585)

[8.2 Gráficos de Contorno 32](#_Toc185612586)

[8.3 Mapas de Calor (Heatmaps) 33](#_Toc185612587)

[Capítulo 9: Interactividade, Animações e Integração com Pandas 34](#_Toc185612588)

[9.1 Ferramentas Interativas 35](#_Toc185612589)

[**%matplotlib notebook** 35](#_Toc185612590)

[**%matplotlib widget** 35](#_Toc185612591)

[**Diferenças e Considerações** 36](#_Toc185612592)

[**Exemplo de Utilização** 36](#_Toc185612593)

[9.2 Animações com FuncAnimation 37](#_Toc185612594)

[9.3 Integração com Pandas 38](#_Toc185612595)

[Capítulo 10: Ajuste de Layout, Combinação de Gráficos e Exportação 40](#_Toc185612596)

[10.1 Ajustar Layout 40](#_Toc185612597)

[10.2 Combinação de Diferentes Tipos de Gráficos 42](#_Toc185612598)

[10.3 Exportação 43](#_Toc185612599)

[Capítulo 11: Integração dos Gráficos em Relatórios Automatizados e Aplicações Web 45](#_Toc185612600)

[11.1 Relatórios Automatizados com LaTeX e Pandoc 45](#_Toc185612601)

[11.2 Integração em Aplicações Web (Dash, Voila) 45](#_Toc185612602)

# Visualização de Dados com Matplotlib

A visualização de dados desempenha um papel crucial na análise e comunicação de informação. Desde o contexto académico ao mundo empresarial, gráficos bem concebidos e adequadamente contextualizados ajudam a transformar números em insights claros, apoiando a tomada de decisão. Este documento guia-te desde os princípios fundamentais de comunicação visual, passando pela teoria da cor, princípios da gestalt e percepção humana, até às capacidades técnicas do Matplotlib. Apresenta também vários tipos de gráficos, como personalizá-los e integrá-los em relatórios e aplicações interactivas. No final, terás adquirido um conjunto de ferramentas e conhecimentos que lhe permitirão criar visualizações de alta qualidade, esteticamente apelativas e informativamente ricas.

## Capítulo 1: Fundamentos da Comunicação Visual – Cor, Gestalt, Percepção e Storytelling com Dados

Antes de nos debruçarmos sobre o Matplotlib, é essencial compreender princípios fundamentais da comunicação visual. Uma visualização eficaz não depende apenas do código: envolve conhecimentos de percepção humana, selecção criteriosa de cores, aplicação de princípios da gestalt e a capacidade de contar histórias com dados.

### 1.1 Teoria da Cor e Paletas Cromáticas

A cor é um dos aspectos mais poderosos na comunicação visual, ajudando a destacar padrões, classificar categorias e criar hierarquias de informação. Contudo, o uso descuidado da cor pode confundir o leitor e distorcer a mensagem.

* **Cores Primárias, Secundárias e Complementares:**  
  O círculo cromático ajuda a compreender a relação entre cores. Por exemplo, cores complementares (azul e laranja) criam contraste forte, enquanto cores análogas (azul e verde) produzem harmonia suave.
* **Cores para Dados Categóricos, Contínuos e Divergentes:**
* **Categóricas:** Paletas que atribuem cores claramente distintas a cada categoria.
* **Contínuas:** Gradientes que mostram a progressão de um valor, por exemplo, do claro para o escuro.
* **Divergentes:** Indicadas para dados com um ponto médio, utilizando cores que divergem a partir de um tom neutro (ex.: azul para valores negativos e vermelho para positivos, com branco ao centro).
* **Acessibilidade e Daltonismo:**  
  É importante garantir que a paleta seja distinguível por pessoas com diferentes tipos de visão. Paletas “colorblind friendly” reduzem a probabilidade de confusão.

### 1.2 Princípios da Gestalt Aplicados à Visualização de Dados

A Psicologia da Gestalt estuda como o nosso cérebro organiza a informação visual. Aplicar estes princípios ao design de gráficos ajuda a criar visualizações mais intuitivas:

* **Proximidade:** Elementos próximos tendem a ser percepcionados como um grupo.
* **Semelhança:** Semelhanças em cor, forma ou tamanho indicam relação entre elementos.
* **Continuidade:** O olho segue linhas e curvas suaves, tornando gráficos de linha mais fáceis de ler.
* **Figura-Fundo:** Destacar o objeto principal do fundo, evitando ruído visual.

### 1.3 Percepção Visual e Limites Humanos

O olho humano é sensível a certos padrões, mas tem limitações:

* Diferenças muito subtis de cor ou textura podem passar despercebidas.
* O excesso de informação visual pode levar a sobrecarga cognitiva.
* É importante simplificar, rotular corretamente e apresentar a informação de forma clara.

### 1.4 Storytelling com Dados

A visualização é comunicação. O storytelling com dados significa estruturar a informação de modo a conduzir o leitor a um insight:

* **Contexto:** Títulos descritivos, legendas, anotações e referências temporais tornam o gráfico mais compreensível.
* **Narrativa:** Apresentar os dados numa sequência lógica.
* **Destaques:** Chamar a atenção para pontos-chave, como picos, variações súbitas ou tendências relevantes.

### 1.5 Aprofundar o Conhecimento sobre Design de Informação, Análise Perceptual e Escolha de Paletas de Cor

Ir além do básico implica compreender design de informação (estudar obras de Edward Tufte ou Colin Ware), psicofísica da percepção (o que o olho distingue melhor ou pior) e ferramentas para escolha de paletas (como “ColorBrewer”). Assim, poderás tomar decisões informadas, escolhendo o tipo de gráfico, paletas cromáticas e níveis de detalhe ideais para o contexto e o público-alvo.

## Capítulo 2: Tipos de Gráfico e a Sua Aplicação

Antes de aprofundarmos o uso do Matplotlib, é útil conhecer os tipos de gráficos mais comuns, bem como as suas aplicações.

### 2.1 Gráficos de Linhas

* **Aplicação:** Mostrar tendências ao longo do tempo ou de um eixo contínuo (ex.: vendas mensais, evolução de temperatura).
* **Melhor Prática:** Usar marcadores nos pontos-chave se necessário, cores contrastantes para múltiplas séries e legendas claras.

### 2.2 Gráficos de Barras

* **Aplicação:** Comparar categorias discretas (ex.: vendas por produto, número de queixas por tipo).
* **Melhor Prática:** Ordenar as categorias de forma lógica, usar cores distintas, adicionar rótulos nos eixos.

### 2.3 Histogramas

* **Aplicação:** Mostrar a distribuição de uma variável contínua (ex.: alturas de uma população, notas de um exame).
* **Melhor Prática:** Escolher um número de “bins” adequado, usar cores suaves e adicionar grelha se necessário.

### 2.4 Gráficos de Pizza

* **Aplicação:** Mostrar proporções de um todo.
* **Melhor Prática:** Usar com moderação, realçar apenas uma fatia importante, incluir percentagens.

### 2.5 Gráficos de Dispersão (Scatter Plots)

* **Aplicação:** Visualizar a relação entre duas variáveis (ex.: altura vs peso, preço vs procura).
* **Melhor Prática:** Ajustar a transparência se houver sobreposição, usar cores ou tamanhos diferenciados para mostrar uma terceira dimensão.

### 2.6 Boxplots e Violinos

* **Aplicação:** Comparar distribuições estatísticas entre grupos (ex.: rendas em diferentes cidades).
* **Melhor Prática:** Legendar cada grupo, cores neutras, destacar medianas e outliers.

### 2.7 Mapas de Calor e Contornos

* **Aplicação:** Visualizar valores numa grelha (ex.: matriz de correlações, intensidade numa imagem).
* **Melhor Prática:** Usar paleta contínua ou divergente adequada, adicionar colorbar e rótulos claros nos eixos.

### 2.8 Gráficos 3D

* **Aplicação:** Dados espaciais, topografias, superfícies matemáticas complexas.
* **Melhor Prática:** Evitar ângulos de visualização pouco claros, usar cores adequadas e considerar se uma projecção 2D não seria mais eficaz.

## Capítulo 3: Introdução ao Matplotlib

**Objectivo:** Familiarizar-se com o Matplotlib, instalação, conceitos fundamentais, estrutura interna e primeiros gráficos.

### 3.1 O que é o Matplotlib?

O Matplotlib é uma biblioteca de visualização de dados para Python, inspirada no MATLAB, mas altamente flexível. É a base de muitas outras bibliotecas (Seaborn, pandas.plot).

* **História:** Criada por John D. Hunter para trazer as capacidades gráficas do MATLAB para o Python.
* **Filosofia:** Oferece uma interface estilo MATLAB (pyplot) e uma abordagem orientada a objetos.

### 3.2 Instalação e Ambiente de Trabalho

* Instale via pip install matplotlib.
* Use ambientes virtuais e ferramentas interactivas como Jupyter Notebook ou Google Colab.
* IDEs: VSCode, PyCharm, Spyder.

### 3.3 Figura, Eixos e Objectos Básicos

* **Figure:** A “folha” onde o gráfico é desenhado.
* **Axes:** A área onde os dados são representados, contendo os eixos (Axis).
* **Axis:** Eixos X, Y (e Z no 3D), com marcas (ticks) e rótulos.
* **Figure:**  
  A Figure é o contorno maior, o “papel” virtual onde o gráfico é desenhado. Pense numa folha de papel A4. Dentro dela podemos ter um ou vários Axes.
* **Axes:**  
  Dentro da Figure, desenhamos uma ou mais áreas retangulares chamadas Axes, que contêm efectivamente o gráfico. É dentro dos Axes que vemos as escalas, rótulos e os dados. Cada Axes pode ter os seus próprios eixos X e Y. Podemos imaginar Axes como a “janela” onde o nosso gráfico aparece.
* **Axis (X, Y, Z):**  
  Dentro de cada Axes temos os Axis, que são os próprios eixos do gráfico (normalmente o eixo X na horizontal e o eixo Y na vertical, e em 3D também o Z). Os Axis têm marcas (chamadas ticks), rótulos numéricos, e um título do eixo (por exemplo, “Tempo (s)” para o eixo X, “Temperatura (°C)” para o eixo Y).

**Exemplo de uma figure com dois axes e um gráfico distinto em cada axe. Código abaixo.**

A graph of a function

Description automatically generated

import matplotlib.pyplot as plt

import numpy as np

# Dados simulados

x = np.linspace(0, 10, 50)

y1 = np.sin(x)

y2 = np.cos(x)

# Criação da figura e dois axes lado a lado (2 colunas, 1 linha)

fig, axes = plt.subplots(1, 2, figsize=(10, 4))

# Primeiro Axes: gráfico da função seno

axes[0].plot(x, y1, color='blue', marker='o')

axes[0].set\_title("Função Seno")

axes[0].set\_xlabel("Eixo X")

axes[0].set\_ylabel("Amplitude")

# Segundo Axes: gráfico da função cosseno

axes[1].plot(x, y2, color='red', linestyle='--')

axes[1].set\_title("Função Cosseno")

axes[1].set\_xlabel("Eixo X")

axes[1].set\_ylabel("Amplitude")

plt.tight\_layout()

plt.show()

### 3.4 Criar o Primeiro Gráfico

Exemplo simples:

import matplotlib.pyplot as plt

x = [1,2,3,4,5]

y = [2,4,1,8,7]

plt.plot(x, y)

plt.title("Meu Primeiro Gráfico")

plt.xlabel("Eixo X")

plt.ylabel("Eixo Y")

plt.show()

**A line graph with numbers and lines

Description automatically generated**

### 3.5 Controlar Tamanho e Resolução

* plt.figure(figsize=(8,6)) ajusta o tamanho.
* plt.savefig("fig.png", dpi=300) salva com alta resolução.

## Capítulo 4: Gráficos de Linhas, Pontos e Estilos

**Objectivo:** Dominar personalização de linhas e pontos, incluindo cores, marcadores, estilos de linha e legendas.

### 4.1 Cores e Estilos de Linha

* Cores por nome (“red”, “blue”) ou hex (“#1f77b4”).
* Estilos: sólido (-), tracejado (--), pontilhado (:), pontilhado-tracejado (-.).

### 4.2 Marcadores e Espessura de Linha

* Marcadores: o, ^, s etc.
* Espessura da linha: linewidth=2.
* Tamanho do marcador: markersize=8.

Exemplo de aplicação dos conceitos anteriores:

import matplotlib.pyplot as plt

import numpy as np

# Dados de exemplo

x = np.linspace(0, 10, 50)

y1 = np.sin(x)

y2 = np.cos(x)

# Figura e axes

fig, ax = plt.subplots(figsize=(8,5))

# Exemplo 1: Linha azul sólida, sem marcadores, espessura padrão

ax.plot(x, y1, color='blue', linestyle='-', label='Seno')

# Exemplo 2: Linha vermelha tracejada com marcadores circulares, espessura e tamanho do marcador personalizados

ax.plot(x, y2, color='red', linestyle='--', marker='o', linewidth=2, markersize=8, label='Cosseno')

# Personalizações do gráfico

ax.set\_title("Exemplo de Personalização de Linhas e Marcadores")

ax.set\_xlabel("Eixo X")

ax.set\_ylabel("Eixo Y")

ax.grid(True)

ax.legend()

plt.tight\_layout()

plt.show()

A graph with red and blue lines

Description automatically generated

**Explicações dos parâmetros utilizados:**

* **Cores:**
  + color='blue' ou color='red' para usar nomes de cores.
  + Poderia usar também um código hex, por ex.: color='#1f77b4'.
* **Estilos de Linha:**
  + linestyle='-' linha sólida.
  + linestyle='--' linha tracejada.
  + Poderia utilizar ':' para pontilhada ou '-.' para pontilhado-tracejado.
* **Marcadores:**
  + marker='o' marcador circular.
  + Poderia usar '^' para marcador triangular, 's' para quadrado, entre outros.
* **Espessura da Linha (linewidth):**
  + linewidth=2 define a espessura da linha em pontos. O padrão é geralmente 1.0.
* **Tamanho do Marcador (markersize):**
  + markersize=8 controla o tamanho do marcador. O padrão costuma ser menor, tipicamente 6.
* **Legenda (label e legend):**
  + label='Seno' e label='Cosseno' definem o texto que aparecerá na legenda.
  + ax.legend() apresenta a legenda no gráfico.

Ao executar o código, obtém-se um gráfico com duas linhas: a primeira azul sólida (função seno) e a segunda vermelha tracejada com marcadores circulares (função cosseno), demonstrando os conceitos mencionados.

### 4.3 Múltiplas Séries no Mesmo Gráfico

Chamar plt.plot() várias vezes e depois plt.legend() para criar legenda.

A graph with green and blue lines

Description automatically generated

import matplotlib.pyplot as plt

import numpy as np

# Dados de exemplo

x = np.linspace(0, 10, 50)

y1 = np.sin(x)

y2 = np.cos(x)

y3 = np.tan(x/2)

# Criação da figura e do axes

fig, ax = plt.subplots(figsize=(8,5))

# Série 1: Linha sólida azul com marcadores circulares

ax.plot(x, y1, color='blue', linestyle='-', marker='o',

        linewidth=2, markersize=8, label='Seno')

# Série 2: Linha tracejada vermelha com marcadores triangulares

ax.plot(x, y2, color='red', linestyle='--', marker='^',

        linewidth=2, markersize=8, label='Cosseno')

# Série 3: Linha pontilhada em verde com marcadores quadrados

ax.plot(x, y3, color='green', linestyle=':', marker='s',

        linewidth=2, markersize=8, label='Tangente (x/2)')

# Personalização do gráfico

ax.set\_title("Múltiplas Séries no Mesmo Gráfico")

ax.set\_xlabel("Eixo X")

ax.set\_ylabel("Valor da Função")

ax.grid(True)

# Exibição da legenda

ax.legend()

# Ajuste automático do layout

plt.tight\_layout()

plt.show()

**O que este exemplo demonstra:**

* **Múltiplas séries:** Três chamadas a ax.plot(), cada uma com seus próprios parâmetros visuais, resultando em três linhas distintas no mesmo axes.
* **Cores e estilos variados:**
  + Seno: linha sólida azul, marcadores 'o'
  + Cosseno: linha tracejada vermelha, marcadores '^'
  + Tangente: linha pontilhada verde, marcadores 's'
* **Espessura e tamanho dos marcadores ajustados:** linewidth=2, markersize=8.
* **Legenda:** Cada série tem um label, exibido após ax.legend().
* **Grid e títulos:** ax.grid(True) para mostrar a grelha, ax.set\_title(), ax.set\_xlabel(), e ax.set\_ylabel() para rotular o gráfico.

Ao executar este código, obtém-se um gráfico único com três funções matemáticas distintas, cada uma com um estilo visual próprio, legendas, e rótulos claros, ilustrando perfeitamente o conceito de múltiplas séries no mesmo gráfico.

### 4.4 Subplots

* fig, ax = plt.subplots() cria um subplot.
* fig, axes = plt.subplots(2,2) cria uma grelha 2x2.
* A abordagem orientada a objectos permite axes[i,j].plot(...).

A screen shot of a graph

Description automatically generated

A group of graphs with lines

Description automatically generated

import matplotlib.pyplot as plt

import numpy as np

# Dados simulados

x = np.linspace(0, 10, 50)

y\_sin = np.sin(x)

y\_cos = np.cos(x)

y\_lin = x

y\_quad = x\*\*2

# Exemplo 1: Um único subplot

fig, ax = plt.subplots(figsize=(6,4))

ax.plot(x, y\_sin, color='blue', label='Seno')

ax.set\_title("Um único subplot")

ax.set\_xlabel("Eixo X")

ax.set\_ylabel("Valor")

ax.legend()

plt.tight\_layout()

plt.show()

# Exemplo 2: Uma grelha 2x2 de subplots

fig, axes = plt.subplots(2, 2, figsize=(10,8))

# axes[0,0] - Gráfico do seno

axes[0,0].plot(x, y\_sin, color='green')

axes[0,0].set\_title("Seno")

axes[0,0].set\_xlabel("X")

axes[0,0].set\_ylabel("sin(x)")

# axes[0,1] - Gráfico do cosseno

axes[0,1].plot(x, y\_cos, color='red', linestyle='--')

axes[0,1].set\_title("Cosseno")

axes[0,1].set\_xlabel("X")

axes[0,1].set\_ylabel("cos(x)")

# axes[1,0] - Gráfico linear

axes[1,0].plot(x, y\_lin, color='blue')

axes[1,0].set\_title("Linear")

axes[1,0].set\_xlabel("X")

axes[1,0].set\_ylabel("x")

# axes[1,1] - Gráfico quadrático

axes[1,1].plot(x, y\_quad, color='magenta', linestyle='-.')

axes[1,1].set\_title("Quadrático")

axes[1,1].set\_xlabel("X")

axes[1,1].set\_ylabel("x²")

plt.tight\_layout()

plt.show()

**O que este exemplo demonstra:**

* **fig, ax = plt.subplots():** Cria uma figura com um único Axes. O ax é um objeto que pode ser usado com a abordagem orientada a objetos, chamando ax.plot(), ax.set\_title(), etc.
* **fig, axes = plt.subplots(2,2):** Cria uma figura com uma grelha 2x2 de Axes, retornando axes como um array bidimensional. Assim, axes[0,0], axes[0,1], axes[1,0] e axes[1,1] referem-se a cada subplot individual.
* **Abordagem Orientada a Objetos:** Ao invés de usar plt.plot(), que funciona de forma mais procedimental, aqui utilizamos axes[i,j].plot(...) e outros métodos do objeto Axes para personalizar cada subplot individualmente (títulos, rótulos, cores, estilos de linha, etc.).

Ao executar o código, primeiro surge um gráfico com um único subplot. De seguida surge um segundo gráfico com uma grelha 2x2, cada secção das 4 secções desenham uma função matemática diferente, ilustrando claramente o uso da abordagem orientada a objetos e do array axes.

## Capítulo 5: Gráficos de Barras, Histogramas e Setores

**Objectivo:** Criar gráficos de barras, histogramas e pizzas, adequados para comparações, distribuições e proporções.

### 5.1 Gráficos de Barras

import matplotlib.pyplot as plt

import numpy as np

categorias = ['A', 'B', 'C', 'D']

valores = [10,15,7,12]

plt.bar(categorias, valores, color='skyblue')

plt.title("Vendas por Categoria")

plt.xlabel("Categoria")

plt.ylabel("Quantidade")

plt.show()

**import matplotlib.pyplot as plt**: Importa o módulo pyplot da biblioteca Matplotlib. O pyplot fornece uma interface simples, estilo MATLAB, para criar gráficos. A convenção é usar plt como apelido.

**import numpy as np**: Importa a biblioteca NumPy, usada para operações numéricas e criação de arrays. Neste caso, a importação de NumPy não é estritamente necessária para o gráfico de barras simples, mas é comum tê-la disponível quando se trabalha com dados numéricos.

Definem-se duas listas:

* categorias: Contém as etiquetas (rótulos) de cada barra. São quatro categorias: 'A', 'B', 'C' e 'D'.
* valores: Contém os valores associados a cada categoria. Por exemplo, a categoria 'A' tem valor 10, 'B' tem 15, 'C' tem 7 e 'D' tem 12.
* **plt.bar()** cria um gráfico de barras. Recebe como primeiro argumento as **categorias (o eixo X)** e como segundo argumento **os valores (a altura de cada barra)**.
* **color='skyblue'** atribui uma cor às barras (um azul claro), ajudando na estética e leitura do gráfico.
* **plt.title("Vendas por Categoria"):** Define o título do gráfico, aparecendo no topo da figura. Aqui o título é "Vendas por Categoria".
* **plt.xlabel("Categoria"):** Define a etiqueta do eixo X como "Categoria".
* **plt.ylabel("Quantidade"):** Define a etiqueta do eixo Y como "Quantidade".
* plt.show() exibe a janela com o gráfico criado.
* Sem esta instrução, em alguns ambientes (como scripts Python normais), o gráfico pode não aparecer. Em ambientes interativos (como notebooks), show() pode não ser obrigatório, mas é uma boa prática chamá-lo para garantir que o gráfico seja exibido.

A graph of blue bars

Description automatically generated with medium confidence

### 5.2 Histogramas

import matplotlib.pyplot as plt

import numpy as np

dados = np.random.randn(1000)

plt.hist(dados, bins=30, color='green', alpha=0.7)

plt.title("Histograma de Dados Aleatórios")

plt.xlabel("Valor")

plt.ylabel("Frequência")

plt.show()

A green graph with black text

Description automatically generated

O código apresentado gera e visualiza um histograma a partir de uma amostra de dados aleatórios.

Inicialmente, importa-se o matplotlib.pyplot como plt, que fornece funções para criar e personalizar gráficos, e o numpy como np, uma biblioteca para cálculos numéricos.

De seguida, cria-se uma variável dados que contém 1000 valores gerados aleatoriamente a partir de uma distribuição normal padrão, através do comando np.random.randn(1000).

Ao chamar plt.hist(dados, bins=30, color='green', alpha=0.7), cria-se um histograma que mostra a distribuição desses valores, dividindo-os em 30 intervalos (bins), pintando as barras de verde (color='green') e ajustando a transparência para 0.7 (alpha=0.7).

As funções plt.title("Histograma de Dados Aleatórios"), plt.xlabel("Valor") e plt.ylabel("Frequência") adicionam título e rótulos aos eixos, tornando o gráfico mais claro e informativo.

Finalmente, plt.show() exibe a figura, apresentando ao utilizador um histograma que ajuda a entender visualmente a distribuição dos 1000 valores gerados aleatoriamente.

### 5.3 Gráficos Circulares

import matplotlib.pyplot as plt

import numpy as np

fatias = [40,30,20,10]

labels = ['Produto A','Produto B','Produto C','Produto D']

plt.pie(fatias, labels=labels, autopct='%1.1f%%', startangle=90, explode=(0,0.1,0,0))

plt.title("Participação de Mercado")

plt.show()

A pie chart with different colored circles

Description automatically generated

O código apresentado cria um gráfico de pizza (gráfico circular) (pie chart) para visualizar a participação de mercado de quatro produtos. A lista fatias contém os valores numéricos que correspondem à proporção de cada categoria (Produto A, Produto B, Produto C e Produto D). A lista labels define os rótulos textuais associados a cada fatia, tornando o gráfico mais legível ao identificar diretamente cada produto no círculo.

A função plt.pie() recebe os valores das fatias e as etiquetas, além de outros parâmetros para personalizar a aparência. O argumento autopct='%1.1f%%' adiciona automaticamente o valor percentual de cada fatia no gráfico, formatando-o com uma casa decimal. O parâmetro startangle=90 faz com que o gráfico de pizza comece o seu primeiro segmento a partir do ângulo de 90 graus, o que pode melhorar a estética ou a legibilidade em certas situações. Já a opção explode=(0,0.1,0,0) destaca a segunda fatia (relativa ao Produto B) ligeiramente afastando-a do centro do gráfico, chamando a atenção para essa categoria específica.

Por fim, plt.title("Participação de Mercado") adiciona um título ao gráfico, contextualizando a informação apresentada, e plt.show() exibe o resultado. Assim, ao executar este código, obtém-se um gráfico de pizza claro e informativo, onde as proporções de mercado de cada produto podem ser visualizadas de forma imediata e intuitiva.

## Capítulo 6: Gráficos de Dispersão, Boxplots e Violino

**Objectivo:** Visualizar relações entre variáveis e distribuições estatísticas detalhadas.

### 6.1 Gráficos de Dispersão (Scatter Plot)

import matplotlib.pyplot as plt

import numpy as np

x = np.random.rand(100)

y = np.random.rand(100)

cores = np.random.rand(100)

tamanhos = (np.random.rand(100)\*100)+50

plt.scatter(x, y, c=cores, s=tamanhos, alpha=0.7, cmap='viridis')

plt.colorbar(label='Cor Aleatória')

plt.title("Gráfico de Dispersão")

plt.xlabel("X")

plt.ylabel("Y")

plt.show()

O código cria um gráfico de dispersão (scatter plot) a partir de conjuntos de valores aleatórios. Primeiro, são gerados 100 valores aleatórios para as coordenadas X e Y, usando np.random.rand(100) que produz números entre 0 e 1, de forma uniforme. Assim, cada ponto terá uma posição no intervalo [0,1] para ambos os eixos.

Em seguida, a variável cores também recebe 100 valores aleatórios, que serão utilizados para definir a cor de cada ponto no scatter plot. Já a variável tamanhos é calculada a partir de valores aleatórios multiplicados por 100 e somando 50, resultando em tamanhos de marcador entre 50 e 150, conferindo uma variação visível no tamanho de cada ponto.

Ao chamar plt.scatter(x, y, c=cores, s=tamanhos, alpha=0.7, cmap='viridis'), cria-se o gráfico de dispersão. O parâmetro c=cores define a cor de cada ponto de acordo com a escala fornecida pelo cmap='viridis', um mapa de cor contínuo e agradável à vista. O s=tamanhos ajusta o tamanho dos marcadores, alpha=0.7 adiciona transparência, ajudando na visualização de pontos sobrepostos, e cmap='viridis' atribui um gradiente de cores aos pontos, mapeando os valores contidos em cores.

plt.colorbar(label='Cor Aleatória') acrescenta uma barra de cor ao lado do gráfico, permitindo interpretar os valores numéricos associados às cores. Por fim, o título e os rótulos dos eixos, definidos por plt.title("Gráfico de Dispersão"), plt.xlabel("X") e plt.ylabel("Y"), fornecem o contexto necessário ao leitor. O comando plt.show() exibe finalmente a figura.

Em resumo, este código demonstra como criar um scatter plot em que a posição, a cor e o tamanho dos pontos variam, resultando numa representação visual dinâmica e informativa de dados

A diagram of different colored circles

Description automatically generated

### 6.2 Boxplots

dados = [np.random.randn(100)+i for i in range(5)]

plt.boxplot(dados, labels=["G1","G2","G3","G4","G5"])

plt.title("Boxplot de Vários Grupos")

plt.ylabel("Valor")

plt.show()

Este código cria um boxplot para comparar a distribuição de dados entre vários grupos. Primeiramente, a variável dados é construída como uma lista de cinco conjuntos de dados, cada um contendo 100 valores. Cada conjunto é gerado por np.random.randn(100)+i, o que significa que partimos de uma distribuição normal padrão (média 0, desvio padrão 1) e adicionamos um valor i diferente a cada grupo. Assim, o primeiro grupo (i=0) será centrado próximo de 0, o segundo (i=1) próximo de 1, o terceiro (i=2) próximo de 2, e assim sucessivamente, criando cinco grupos de dados com médias crescentes.

A função plt.boxplot(dados, labels=["G1","G2","G3","G4","G5"]) gera o boxplot, mostrando a mediana, quartis e possíveis outliers para cada um dos cinco grupos. Os rótulos “G1”, “G2”, “G3”, “G4” e “G5” são usados para identificar cada caixa no gráfico, facilitando a comparação visual entre as distribuições.

Em seguida, plt.title("Boxplot de Vários Grupos") adiciona um título ao gráfico, e plt.ylabel("Valor") atribui um rótulo ao eixo vertical, ajudando a contextualizar os valores representados. Por fim, plt.show() exibe o gráfico na tela.

Em suma, este código produz um boxplot simples que ajuda a visualizar e comparar a distribuição, mediana, dispersão e possíveis valores atípicos de cinco grupos de dados gerados aleatoriamente.

A diagram of a group of people

Description automatically generated with medium confidence

### 6.3 Gráficos de Violino

plt.violinplot(dados, showmeans=True)

plt.title("Violin Plot")

plt.ylabel("Valor")

plt.show()

Este código gera um gráfico do tipo "violin plot", uma variação do boxplot que mostra não apenas estatísticas descritivas, mas também a densidade da distribuição dos dados. Para isso, utiliza-se a função plt.violinplot(dados, showmeans=True).

O parâmetro showmeans=True faz com que o gráfico exiba uma linha ou marcador indicando a média de cada conjunto de dados. Assim, cada "violino" fornece, ao mesmo tempo, informações sobre a forma, dispersão e tendência central dos dados. O formato do "violino" é obtido espelhando a densidade da distribuição ao redor do eixo vertical, criando uma figura semelhante ao instrumento musical.

O comando plt.title("Violin Plot") insere um título no gráfico, enquanto plt.ylabel("Valor") adiciona um rótulo ao eixo vertical, tornando o gráfico mais claro e interpretável. Por fim, plt.show() exibe a figura.

Em suma, este código cria um violin plot que ajuda a visualizar não apenas quartis e medianas (como num boxplot), mas também a densidade da distribuição, apresentando uma visão mais completa dos dados.

A diagram of a violin plot

Description automatically generated

## Capítulo 7: Personalização Avançada

**Objectivo:** Aprender técnicas mais sofisticadas de personalização, anotando gráficos, formatando eixos, usando estilos temáticos.

### 7.1 Anotações e Texto

import numpy as np

import matplotlib.pyplot as plt

# Definir x como um conjunto de pontos entre 0 e 2π

x = np.linspace(0, 2\*np.pi, 100)  # 100 pontos igualmente espaçados

max\_x = np.pi/2

max\_y = 1

plt.plot(x, np.sin(x))

plt.annotate("Máximo do Seno", xy=(max\_x,max\_y), xytext=(max\_x+0.5,max\_y+0.5),

             arrowprops=dict(facecolor='black', shrink=0.05),

             bbox=dict(boxstyle='round,pad=0.3', fc='yellow', alpha=0.5))

plt.xlabel('x')

plt.ylabel('sin(x)')

plt.title('Gráfico do Seno')

plt.grid(True)

plt.show()

O código começa por importar as bibliotecas necessárias: numpy é importada como np para facilitar operações matemáticas e criação de arrays, e matplotlib.pyplot como plt para a geração de gráficos. De seguida, define-se x como um conjunto de 100 valores igualmente espaçados entre 0 e 2π, através da função np.linspace(0, 2\*np.pi, 100), garantindo assim uma amostra de pontos adequada para desenhar a função seno ao longo de um ciclo completo.

Em seguida, definem-se max\_x e max\_y como a coordenada correspondente ao ponto máximo do seno, que ocorre em x = π/2 e y = 1. Ao chamar plt.plot(x, np.sin(x)), desenha-se a curva da função sin(x) utilizando os valores definidos anteriormente. Desta forma, obtém-se um gráfico suave e contínuo da função trignométrica seno.

Para destacar o ponto máximo da função, o código usa plt.annotate() com o texto "Máximo do Seno". O argumento xy=(max\_x,max\_y) indica a posição do ponto de interesse no gráfico. Por sua vez, xytext=(max\_x+0.5, max\_y+0.5) especifica a posição onde o texto da anotação aparecerá, ligeiramente afastado do ponto, facilitando a sua leitura. Os parâmetros arrowprops e bbox personalizam a aparência da seta e da caixa de texto, respetivamente. Neste exemplo, a seta tem a cor preta e a caixa amarela em forma arredondada, o que melhora a clareza visual da anotação.

Por fim, plt.xlabel('x') e plt.ylabel('sin(x)') rotulam os eixos X e Y, ao passo que plt.title('Gráfico do Seno') atribui um título ao gráfico, contextualizando a informação apresentada. A chamada plt.grid(True) adiciona uma grelha que facilita a leitura dos valores ao longo dos eixos, e plt.show() apresenta finalmente o gráfico na tela. Assim, o resultado é um gráfico bem estruturado, com o ponto máximo da função seno destacado, rótulos claros e uma grelha útil para análise visual.

A graph with a line

Description automatically generated

### 7.2 Formatação Avançada dos Eixos

* Rotação de rótulos: plt.xticks(rotation=45)
* Limites dos eixos: plt.xlim(), plt.ylim()
* Uso de Matplotlib.ticker para formatação personalizada.

A graph with a line

Description automatically generated

import matplotlib.pyplot as plt

import numpy as np

from matplotlib.ticker import FuncFormatter

# Dados de exemplo

x = np.linspace(0, 10, 11)

y = x\*\*2

fig, ax = plt.subplots(figsize=(8,5))

ax.plot(x, y, marker='o', color='blue')

# Título e rótulos de eixos

ax.set\_title("Exemplo de Formatação Avançada dos Eixos")

ax.set\_xlabel("Eixo X (passos)")

ax.set\_ylabel("Eixo Y (x²)")

# Rotação de rótulos do eixo X

plt.xticks(rotation=45)

# Ajuste dos limites dos eixos

# Queremos focar apenas na parte entre x=2 e x=8, e y=0 até 70

plt.xlim(2, 8)

plt.ylim(0, 70)

# Uso de Matplotlib.ticker para formatação personalizada dos ticks

# Por exemplo, formatar valores do eixo Y como valores monetários (R$)

def real\_format(x, pos):

    return f"R${x:,.0f}"

formatter = FuncFormatter(real\_format)

ax.yaxis.set\_major\_formatter(formatter)

# Grid e layout

ax.grid(True)

plt.tight\_layout()

plt.show()

Este código demonstra diferentes técnicas de formatação avançada dos eixos num gráfico produzido com Matplotlib, incluindo a rotação de rótulos, a definição manual de limites para os eixos, e a personalização do formato dos valores apresentados nos ticks (marcas dos eixos).

No início, são importadas as bibliotecas necessárias. O matplotlib.pyplot é importado como plt para lidar com a criação e controlo de gráficos, numpy como np para gerar dados numéricos, e FuncFormatter da classe matplotlib.ticker para criar um formatador de valores personalizado.

Em seguida, definem-se os dados. A variável x é criada com np.linspace(0, 10, 11), o que gera 11 pontos igualmente espaçados entre 0 e 10. A variável y é definida como x\*\*2, produzindo valores quadráticos, criando assim uma relação não linear entre x e y.

Ao criar a figura e o Axes com fig, ax = plt.subplots(figsize=(8,5)), obtém-se um espaço para o gráfico com dimensões de 8 por 5 polegadas. O método ax.plot(x, y, marker='o', color='blue') desenha os pontos (marcadores circulares em azul) que correspondem à função y = x².

Depois, definem-se o título do gráfico, bem como os rótulos dos eixos X e Y, usando ax.set\_title(), ax.set\_xlabel() e ax.set\_ylabel(). Isto torna a visualização mais clara, indicando o propósito do gráfico e o significado dos eixos.

Para melhorar a legibilidade dos rótulos do eixo X, chama-se plt.xticks(rotation=45), o que roda os valores do eixo X em 45 graus. Esta rotação é útil quando há valores muito próximos ou longos, evitando sobreposição e facilitando a leitura.

Em seguida, ajustam-se os limites visíveis dos eixos. plt.xlim(2, 8) restringe o eixo X ao intervalo entre 2 e 8, enquanto plt.ylim(0, 70) limita o eixo Y entre 0 e 70. Isto permite focar apenas na parte do gráfico mais relevante, ignorando dados fora desse intervalo.

A parte mais avançada do código ocorre ao utilizar o FuncFormatter. Primeiro, define-se a função real\_format(x, pos) que recebe o valor x (o valor do tick) e pos (a posição do tick), retornando uma string formatada no estilo monetário, por exemplo "R$1,000" ao invés de "1000". Com formatter = FuncFormatter(real\_format) é criado o formatador personalizado e, posteriormente, ax.yaxis.set\_major\_formatter(formatter) aplica este formatador ao eixo Y, alterando a forma como os valores do eixo Y são apresentados. Agora, em vez de simples números, aparecerão valores prefixados por "R$" e com vírgulas nos milhares.

Por fim, ax.grid(True) ativa uma grelha auxiliar no fundo do gráfico, ajudando na leitura dos valores, e plt.tight\_layout() ajusta automaticamente o espaçamento interno da figura, garantindo que todos os rótulos e títulos fiquem visíveis e não colidam com as margens. Ao chamar plt.show() o gráfico final é exibido, mostrando um exemplo de formatação avançada dos eixos.

Em suma, este código exemplifica como rodar rótulos no eixo X, alterar manualmente os limites de visualização dos eixos, e usar formatadores personalizados para apresentar valores nos ticks de forma mais compreensível e contextualmente relevante.

### 7.3 Estilos Temáticos

plt.style.use(‘ggplot')

A graph with red and blue lines

Description automatically generated

import matplotlib.pyplot as plt

import numpy as np

# Aplicar o estilo "seaborn-darkgrid"

plt.style.use('ggplot')

# Gerar alguns dados de exemplo

x = np.linspace(0, 10, 100)

y1 = np.sin(x)

y2 = np.cos(x)

# Criar a figura e o Axes

fig, ax = plt.subplots(figsize=(8,5))

# Plotar duas séries de dados

ax.plot(x, y1, color='blue', linewidth=2, label='Seno')

ax.plot(x, y2, color='red', linestyle='--', linewidth=2, label='Cosseno')

# Título e rótulos dos eixos

ax.set\_title("Exemplo com Estilo 'ggplot'")

ax.set\_xlabel("Eixo X")

ax.set\_ylabel("Valor da Função")

# Legenda

ax.legend(loc='upper right')

# Mostrar o gráfico

plt.tight\_layout()

plt.show()

Este código cria um gráfico em Python utilizando a biblioteca Matplotlib, aplicando um estilo pré-definido chamado "ggplot", inspirando-se no visual tradicional do software R ggplot2. A cada etapa, diferentes aspectos do gráfico são configurados, resultando numa apresentação clara, coerente e esteticamente agradável.

Num primeiro momento, são importados os módulos matplotlib.pyplot como plt e numpy como np. O NumPy é utilizado para gerar dados numéricos e o Matplotlib para a criação de gráficos. Ao chamar plt.style.use('ggplot'), seleciona-se o estilo "ggplot", que altera a paleta de cores, as fontes, o fundo e outros parâmetros visuais do gráfico, conferindo-lhe uma aparência mais suave e moderna.

A seguir, define-se o array x com np.linspace(0, 10, 100), o que gera 100 pontos igualmente espaçados entre 0 e 10. As variáveis y1 e y2 são calculadas usando funções trigonométricas: y1 = np.sin(x) representa a função seno e y2 = np.cos(x) a função cosseno. Assim, obtemos duas séries de dados matemáticos para representação gráfica.

Com fig, ax = plt.subplots(figsize=(8,5)) cria-se uma figura de 8 polegadas de largura por 5 de altura, e um único objeto de eixos (ax), no qual o gráfico será desenhado. A seguir, ax.plot(x, y1, color='blue', linewidth=2, label='Seno') desenha a primeira série (seno) a azul, com linha espessa. Logo depois, ax.plot(x, y2, color='red', linestyle='--', linewidth=2, label='Cosseno') adiciona a segunda série (cosseno) a vermelho, usando uma linha tracejada para distinguir visualmente as duas funções.

O título do gráfico é definido com ax.set\_title("Exemplo com Estilo 'ggplot'"), enquanto ax.set\_xlabel("Eixo X") e ax.set\_ylabel("Valor da Função") adicionam rótulos aos eixos horizontal e vertical, respectivamente. Estes rótulos e títulos ajudam o leitor a compreender o que está a ser apresentado. Em seguida, ax.legend(loc='upper right') chama a legenda do gráfico, posicionando-a no canto superior direito, de forma a identificar facilmente qual linha corresponde ao seno e qual corresponde ao cosseno.

Finalmente, plt.tight\_layout() ajusta automaticamente o espaçamento interno da figura, evitando que os rótulos e legendas se sobreponham ou fiquem cortados. Ao chamar plt.show(), o gráfico é exibido, mostrando as duas funções sobre o fundo suave e agradável do estilo "ggplot".

Em suma, o código demonstra como combinar dados gerados pelo NumPy, configurações estéticas do Matplotlib e um estilo pré-definido, resultando num gráfico de alta qualidade, claro e equilibrado.

### 7.4 Legendas, Títulos e Labels Personalizados

* Ajustar a posição da legenda: plt.legend(loc='upper right')
* Títulos multi-linha: plt.title("Linha 1\nLinha 2")

**O que o exemplo seguinte faz:**

* **Título multi-linha:**  
  ax.set\_title("Gráfico de Funções Trigonométricas\nSeno e Cosseno") insere um título com uma quebra de linha entre “Gráfico de Funções Trigonométricas” e “Seno e Cosseno”. Assim, o título aparece distribuído em duas linhas.
* **Posição da legenda:**  
  ax.legend(loc='upper right') posiciona a legenda no canto superior direito do Axes, identificando claramente cada linha (seno e cosseno).

Ao executar este código, será exibido um gráfico limpo, com o título em duas linhas, legendas bem posicionadas e dois conjuntos de dados distintos apresentados de forma clara.

import matplotlib.pyplot as plt

import numpy as np

# Gerar dados de exemplo

x = np.linspace(0, 10, 100)

y1 = np.sin(x)

y2 = np.cos(x)

# Criar a figura e o Axes

fig, ax = plt.subplots(figsize=(8,5))

# Plotar duas séries de dados, cada uma com um label

ax.plot(x, y1, color='blue', linewidth=2, label='Seno')

ax.plot(x, y2, color='red', linestyle='--', linewidth=2, label='Cosseno')

# Título com múltiplas linhas

ax.set\_title("Gráfico de Funções Trigonométricas\nSeno e Cosseno", fontsize=14)

# Rótulos dos eixos

ax.set\_xlabel("Eixo X")

ax.set\_ylabel("Valor")

# Ajustar a posição da legenda para o canto superior direito

ax.legend(loc='upper right')

# Ajuste de layout

plt.tight\_layout()

# Mostrar o gráfico

plt.show()

A graph of a function

Description automatically generated

## Capítulo 8: Visualização em 3D, Contornos e Mapas de Calor

**Objectivo:** Explorar visualizações tridimensionais, superfícies, contornos e mapas de calor.

### 8.1 Gráficos em 3D

import matplotlib.pyplot as plt

import numpy as np

from mpl\_toolkits.mplot3d import Axes3D

X = np.linspace(-5,5,50)

Y = np.linspace(-5,5,50)

X,Y = np.meshgrid(X,Y)

Z = np.sin(np.sqrt(X\*\*2+Y\*\*2))

fig = plt.figure(figsize=(8,6))

ax = fig.add\_subplot(111, projection='3d')

ax.plot\_surface(X, Y, Z, cmap='viridis')

ax.set\_title("Superfície 3D")

plt.show()

A graph of a colorful cone

Description automatically generated with medium confidence

Este código cria uma visualização tridimensional de uma superfície definida por uma função matemática. Inicialmente, são importadas as bibliotecas necessárias: matplotlib.pyplot como plt para manipular gráficos, numpy como np para operações numéricas e criação de arrays, e Axes3D de mpl\_toolkits.mplot3d para permitir a geração de gráficos 3D.

Em seguida, definem-se os arrays X e Y utilizando np.linspace(-5,5,50), o que gera 50 pontos entre -5 e 5, igualmente espaçados. Estas variáveis servem como eixo horizontal e eixo vertical na malha de pontos da superfície. Ao chamar X,Y = np.meshgrid(X,Y), cria-se uma malha bidimensional, combinando todos os pontos de X com todos os pontos de Y, resultando num conjunto de coordenadas (X,Y) sobre o qual a função será avaliada.

A variável Z é calculada como np.sin(np.sqrt(X\*\*2+Y\*\*2)). Esta expressão avalia o valor da função sin(√(X²+Y²)) para cada ponto da malha. Assim, Z contém a altura da superfície em cada posição (X,Y), resultando numa superfície ondulada.

A seguir, é criada uma figura com dimensões (8,6) polegadas. O comando fig.add\_subplot(111, projection='3d') adiciona ao fig um conjunto de eixos tridimensionais, armazenando este objeto em ax. Ao chamar ax.plot\_surface(X, Y, Z, cmap='viridis'), desenha-se a superfície 3D resultante dos valores de Z sobre a grelha definida por X e Y, aplicando o mapa de cores 'viridis', que varia suavemente do azul ao verde, melhorando a perceção da profundidade e relevo.

Com ax.set\_title("Superfície 3D"), atribui-se um título ao gráfico, tornando mais explícita a natureza da figura. Por fim, plt.show() exibe a janela com o gráfico, permitindo ao utilizador interagir com a visualização, rodar a perspectiva 3D (quando possível) e analisar a forma da superfície.

Em resumo, este código gera uma superfície tridimensional a partir de uma função matemática, ilustra a utilização de NumPy para criar malhas de pontos e mostra como usar o Matplotlib, juntamente com as extensões 3D, para criar visualizações tridimensionais.

### 8.2 Gráficos de Contorno

import matplotlib.pyplot as plt

import numpy as np

from mpl\_toolkits.mplot3d import Axes3D

X = np.linspace(-5,5,50)

Y = np.linspace(-5,5,50)

X,Y = np.meshgrid(X,Y)

Z = np.sin(np.sqrt(X\*\*2+Y\*\*2))

plt.contourf(X, Y, Z, cmap='viridis')

plt.colorbar(label='Valor')

plt.title("Mapa de Contorno")

plt.xlabel("X")

plt.ylabel("Y")

plt.show()

Este código cria um mapa de contorno bidimensional a partir de valores calculados numa grelha de pontos (X,Y). Primeiramente, são importadas as bibliotecas necessárias: matplotlib.pyplot como plt para gerar e manipular gráficos, numpy como np para lidar com operações numéricas e arrays, e Axes3D de mpl\_toolkits.mplot3d (embora neste exemplo 3D não seja necessário, pode ser útil em cenários futuros).

A variável X é definida como um conjunto de 50 valores igualmente espaçados entre -5 e 5, e o mesmo acontece com Y. Em seguida, X, Y = np.meshgrid(X, Y) transforma os arrays unidimensionais em matrizes bidimensionais, criando uma malha de coordenadas. Isto significa que, para cada ponto no plano, agora existe um par (X[i,j], Y[i,j]) representando cada célula da grelha.

A variável Z é definida como np.sin(np.sqrt(X\*\*2+Y\*\*2)). Esta expressão calcula o valor da função sinusoidal da distância do ponto (X,Y) à origem. Como resultado, o campo escalar Z formará um padrão ondulatório, à semelhança de círculos de onda cada vez maiores à medida que nos afastamos do centro.

A chamada plt.contourf(X, Y, Z, cmap='viridis') gera um mapa de contorno preenchido, onde as linhas de contorno delimitam regiões de valores similares de Z, e o atributo cmap='viridis' aplica um mapa de cores moderno e suave, tornando a transição entre diferentes valores facilmente perceptível.

plt.colorbar(label='Valor') adiciona uma barra lateral de cores, o que auxilia a interpretar numericamente cada cor do mapa de contorno. Por sua vez, plt.title("Mapa de Contorno"), plt.xlabel("X") e plt.ylabel("Y") atribuem um título ao gráfico e rótulos aos eixos, tornando a figura mais compreensível. Ao executar plt.show(), é exibido o mapa de contorno colorido, que representa de forma intuitiva e visual os valores da função sinusoidal sobre a área definida por X e Y.

A chart of a concentric circle

Description automatically generated

### 8.3 Mapas de Calor (Heatmaps)

import matplotlib.pyplot as plt

import numpy as np

matriz = np.random.rand(10,10)

plt.imshow(matriz, cmap='hot', interpolation='nearest')

plt.colorbar(label='Intensidade')

plt.title("Heatmap")

plt.show()

Este código gera um mapa de calor (heatmap) a partir de uma matriz aleatória de 10 por 10, ilustrando valores numéricos através de cores. Primeiro, são importados o matplotlib.pyplot como plt para criar e gerir gráficos, e o numpy como np para operações numéricas e geração de dados.

A linha matriz = np.random.rand(10,10) cria uma matriz de 10 linhas e 10 colunas, cujos elementos são valores aleatórios entre 0 e 1, distribuídos uniformemente. Estes valores podem ser interpretados como intensidades ou quantidades a representar visualmente.

Em seguida, plt.imshow(matriz, cmap='hot', interpolation='nearest') exibe a matriz sob a forma de uma imagem, onde cada célula é convertida num pequeno quadrado colorido. O parâmetro cmap='hot' aplica um mapa de cores do tipo gradiente de calor, tipicamente passando do preto e vermelho até ao amarelo, de acordo com o valor numérico. Valores mais próximos de 0 tenderão para cores mais escuras, enquanto valores próximos de 1 serão representados por cores mais claras e intensas. O parâmetro interpolation='nearest' evita a suavização entre os píxeis, resultando numa imagem com fronteiras bem definidas entre as células.

A chamada plt.colorbar(label='Intensidade') adiciona uma barra lateral de referência às cores, indicando numericamente a correspondência entre as cores e os valores. Desta forma, o leitor pode interpretar corretamente as tonalidades apresentadas.

Por fim, plt.title("Heatmap") atribui um título ao gráfico, fornecendo contexto, e plt.show() apresenta a janela gráfica. O resultado é um mapa de calor simples, mas ilustrativo, onde cada célula da matriz aleatória é convertida numa cor, tornando mais intuitiva a compreensão da distribuição dos valores numéricos.

A heat map with different colored squares

Description automatically generated

## Capítulo 9: Interactividade, Animações e Integração com Pandas

**Objectivo:** Criar visualizações dinâmicas, animações e integrar com pandas.

### 9.1 Ferramentas Interativas

No Jupyter: %matplotlib notebook ou %matplotlib widget para zoom e pan interativo.

A utilização de ferramentas interativas no Jupyter Notebook é um recurso valioso para a análise exploratória de dados, permitindo que o utilizador não se limite a visualizações estáticas. Ao contrário do ambiente padrão, em que os gráficos são apresentados como imagens estáticas (normalmente através do comando %matplotlib inline), as modalidades %matplotlib notebook ou %matplotlib widget transformam os gráficos em elementos interativos dentro da célula do notebook, oferecendo funcionalidades adicionais como zoom, pan, rotação (em gráficos 3D), e até a exportação direta da figura.

**%matplotlib notebook**

Quando se executa o comando mágico %matplotlib notebook numa célula do Jupyter, o backend do Matplotlib é alterado para produzir gráficos interativos baseados em JavaScript e HTML, integrados no próprio notebook. Esta abordagem confere uma barra de ferramentas interativa logo acima ou abaixo do gráfico. Esta barra permite:

* **Zoom:** É possível ampliar uma determinada área do gráfico. Basta clicar no ícone da lupa e depois desenhar uma caixa sobre a região de interesse, permitindo analisar detalhes finos nos dados.
* **Pan:** Permite deslocar a área de visualização do gráfico, arrastando-o para explorar outras partes dos dados, sem alterar o nível de zoom.
* **Restaurar Visualização:** Caso o utilizador se perca ao fazer zoom ou pan, existe um botão para restaurar a vista original do gráfico.
* **Exportar a Figura:** Botões para guardar a figura num ficheiro de imagem (normalmente PNG) sem precisar de comandos adicionais.

Esta interatividade é especialmente útil quando se lida com grandes conjuntos de dados ou quando se quer inspecionar determinados pontos, sobreposições ou áreas do gráfico de modo a ter uma melhor perceção antes de tirar conclusões.

**%matplotlib widget**

Já o comando %matplotlib widget faz uso de uma integração ainda mais profunda com o ecossistema Jupyter, usando o framework ipympl (uma extensão do Jupyter para o Matplotlib). Aqui, os gráficos são renderizados como widgets interativos, implementados com a biblioteca ipywidgets. Isso traz não apenas as funcionalidades de zoom e pan descritas acima, mas também permite uma integração mais rica com outros widgets interativos do Jupyter, como seletores, sliders, botões e menus drop-down.

* **Combinação com ipywidgets:** Ao integrar com sliders ou caixas de seleção, é possível atualizar o gráfico em tempo real, por exemplo, alterando parâmetros de uma função, filtrando dados ou mudando o tipo de gráfico. Essa abordagem torna o notebook não apenas um relatório, mas também uma ferramenta exploratória, permitindo que o utilizador brinque com os dados e veja imediatamente o impacto das alterações nos gráficos.
* **Experiência mais fluida:** Como os gráficos são widgets, podem ser interligados a outros widgets, compondo pequenas aplicações interativas dentro do notebook, algo bastante útil em data storytelling, ensino, ou no refinamento de modelos, visualizações e parâmetros.

**Diferenças e Considerações**

* **Performance:** Enquanto %matplotlib inline simplesmente gera imagens estáticas, o modo interativo pode ser mais pesado em termos de recursos, dependendo do tamanho dos dados e da complexidade do gráfico. Usuários com notebooks menos potentes ou com dados muito extensos podem notar alguma lentidão.
* **Ambiente:** Estes modos interativos funcionam tipicamente em Jupyter Notebooks e JupyterLab. Ferramentas como Google Colab podem não suportar integralmente %matplotlib notebook ou %matplotlib widget. Nesse caso, é necessário verificar a documentação do ambiente de execução para saber que backends interativos estão disponíveis.
* **Manutenção do Estado Interativo:** Se o notebook for guardado e posteriormente reaberto, os gráficos podem requerer nova execução para recuperar a interatividade. Além disso, partilhar o notebook com terceiros fará com que eles também precisem executar as células adequadas para experimentar as funcionalidades interativas.

**Exemplo de Utilização**

Imagine um notebook Jupyter onde se pretende analisar um conjunto de dados. Com %matplotlib notebook, basta:

%matplotlib notebook

import matplotlib.pyplot as plt

import numpy as np

x = np.linspace(0, 10, 100)

y = np.sin(x)

plt.figure()

plt.plot(x, y, marker='o')

plt.title("Seno de x")

plt.xlabel("x")

plt.ylabel("sin(x)")

plt.show()

Ao correr este código, surge um gráfico com uma barra de ferramentas. O utilizador pode fazer zoom em áreas específicas, deslocar o gráfico e até guardá-lo facilmente.

Para %matplotlib widget, assume-se que a extensão ipympl está instalada:

%matplotlib widget

import matplotlib.pyplot as plt

import numpy as np

x = np.linspace(0, 10, 100)

y = np.sin(x)

plt.figure()

plt.plot(x, y, marker='o')

plt.title("Seno de x")

plt.xlabel("x")

plt.ylabel("sin(x)")

plt.show()

Aqui, terá não só a barra interativa como também a possibilidade de, numa célula seguinte, criar sliders e outros controlos com ipywidgets, interligando-os com o gráfico.

As ferramentas interativas no Jupyter tornam a análise visual mais dinâmica e envolvente. Em vez de se limitar a inspecionar gráficos estáticos, o utilizador pode explorar os dados em tempo real, ajustando a visualização e, com a ajuda de widgets interativos, manipulando parâmetros que influenciam o gráfico. Isto contribui para um processo de análise de dados mais rico, intuitivo e eficiente.

### 9.2 Animações com FuncAnimation

import matplotlib.pyplot as plt

import numpy as np

from matplotlib.animation import FuncAnimation

fig, ax = plt.subplots()

xdata, ydata = [], []

ln, = plt.plot([], [], 'ro', animated=True)

def init():

    ax.set\_xlim(0, 2\*np.pi)

    ax.set\_ylim(-1,1)

    return ln,

def update(frame):

    xdata.append(frame)

    ydata.append(np.sin(frame))

    ln.set\_data(xdata,ydata)

    return ln,

ani = FuncAnimation(fig, update, frames=np.linspace(0,2\*np.pi,128), init\_func=init, blit=True)

plt.show()

Este código demonstra a criação de uma animação simples utilizando o Matplotlib, mostrando um ponto (ou uma série de pontos) movendo-se ao longo da curva da função seno.

Primeiro, são importados os módulos necessários: matplotlib.pyplot para criação de gráficos, numpy para operações numéricas, e FuncAnimation de matplotlib.animation para gerar a animação. Em seguida, cria-se uma figura (fig) e um conjunto de eixos (ax) com fig, ax = plt.subplots(). As variáveis xdata e ydata começam vazias, destinadas a guardar progressivamente os valores do eixo X e do eixo Y ao longo dos frames da animação. A linha ln, = plt.plot([], [], 'ro', animated=True) cria uma linha (na verdade, apenas pontos, neste caso marcadores 'o' de cor vermelha) sem dados iniciais, que será atualizada a cada frame.

A função init() é definida para configurar o estado inicial da animação. Aqui, ax.set\_xlim(0, 2\*np.pi) e ax.set\_ylim(-1,1) determinam os limites iniciais dos eixos X e Y, garantindo que a área exibida acomoda o gráfico do seno entre 0 e 2π e valores entre -1 e 1. Esta função devolve ln, para indicar ao FuncAnimation qual objeto gráfico vai ser atualizado.

A função update(frame) é chamada para cada frame da animação, recebendo um valor da lista de frames definida posteriormente. Neste caso, frame assumirá valores de 0 a 2π, divididos em 128 passos (frames=np.linspace(0,2\*np.pi,128)). A cada chamada, frame é adicionado a xdata, e np.sin(frame) é adicionado a ydata. Desta forma, o conjunto (xdata, ydata) vai crescendo, representando mais pontos da curva senoidal. Em seguida, ln.set\_data(xdata, ydata) atualiza a linha no gráfico com os novos valores. Isto faz com que o ponto vermelho se desloque progressivamente, traçando o caminho da função seno. Novamente, a função retorna ln, para indicar ao FuncAnimation qual elemento do gráfico foi atualizado.

Por fim, a linha ani = FuncAnimation(fig, update, frames=np.linspace(0,2\*np.pi,128), init\_func=init, blit=True) cria a animação. O parâmetro frames especifica a sequência de valores que frame assumirá. O init\_func=init determina a função que configura o estado inicial, enquanto blit=True otimiza a animação, desenhando apenas os elementos que mudam entre frames, melhorando o desempenho.

Por último, plt.show() exibe a janela do gráfico. Executando o código, surge uma figura onde um ponto vermelho se move ao longo da curva do seno de x, começando em x=0 e avançando até x=2π, criando assim uma animação suave e contínua.

### 9.3 Integração com Pandas

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

df = pd.DataFrame({"Mês":[1,2,3], "Vendas":[230,250,270]})

df.plot(x="Mês", y="Vendas", marker='o', title="Vendas Mensais")

plt.show()

Este código exemplifica como integrar o Matplotlib com o Pandas para gerar gráficos a partir de dados contidos num DataFrame.

Inicialmente, importam-se as bibliotecas necessárias:

* matplotlib.pyplot como plt para gerar e manipular gráficos.
* numpy como np (embora não seja usado diretamente neste exemplo, é comum tê-lo importado).
* pandas como pd para lidar facilmente com dados tabulares e criar DataFrames.

Em seguida, cria-se um DataFrame df através do pd.DataFrame(), fornecendo um dicionário onde as chaves são os nomes das colunas ("Mês" e "Vendas") e os valores são listas com dados correspondentes. Desta forma, o DataFrame df contém três meses (1, 2 e 3) e as vendas correspondentes (230, 250 e 270).

A linha df.plot(x="Mês", y="Vendas", marker='o', title="Vendas Mensais") utiliza o método plot do objeto DataFrame do Pandas. Este método integra-se com o Matplotlib para produzir um gráfico sem que seja necessário criar arrays separados.

* x="Mês" especifica que a coluna "Mês" será utilizada no eixo horizontal (X).
* y="Vendas" indica que a coluna "Vendas" será usada para o eixo vertical (Y).
* marker='o' adiciona marcadores em forma de círculo nos pontos da linha, ajudando a destacar cada valor de vendas mensais.
* title="Vendas Mensais" atribui um título ao gráfico, facilitando a interpretação do que está a ser mostrado.

Ao chamar plt.show(), o gráfico é exibido numa nova janela ou inline, caso esteja a usar um ambiente como o Jupyter Notebook. O resultado é um gráfico de linhas simples, mas informativo, que mostra as vendas ao longo dos três meses, com pontos marcados para cada observação e um título descrevendo o conteúdo do gráfico.

A graph with a line and a point

Description automatically generated

## Capítulo 10: Ajuste de Layout, Combinação de Gráficos e Exportação

**Objectivo:** Melhorar a apresentação final, ajustar layouts, combinar gráficos e exportar.

### 10.1 Ajustar Layout

* plt.tight\_layout() para evitar sobreposições.
* fig.subplots\_adjust(wspace=0.4, hspace=0.4) para ajustes manuais.

Exemplo de aplicação:

import matplotlib.pyplot as plt

import numpy as np

# Gerar alguns dados de exemplo

x = np.linspace(0, 10, 100)

y1 = np.sin(x)

y2 = np.cos(x)

y3 = np.tan(x)

y4 = np.exp(-x)

# Criar uma figura com 4 subplots (2x2)

fig, axes = plt.subplots(2, 2, figsize=(10,8))

# Primeiro subplot (linha 0, coluna 0)

axes[0,0].plot(x, y1, color='blue')

axes[0,0].set\_title("Seno")

# Segundo subplot (linha 0, coluna 1)

axes[0,1].plot(x, y2, color='red')

axes[0,1].set\_title("Cosseno")

# Terceiro subplot (linha 1, coluna 0)

axes[1,0].plot(x, y3, color='green')

axes[1,0].set\_title("Tangente")

# Quarto subplot (linha 1, coluna 1)

axes[1,1].plot(x, y4, color='purple')

axes[1,1].set\_title("Exponencial Decrescente")

# Ajuste automático do layout para evitar sobreposições

plt.tight\_layout()

# Ajuste manual adicional do espaçamento entre subplots

fig.subplots\_adjust(wspace=0.4, hspace=0.4)

# Exibir o gráfico

plt.show()

**O que este código faz:**

1. **Criação de múltiplos subplots:**  
   fig, axes = plt.subplots(2, 2, figsize=(10,8)) cria uma figura com uma grelha de 2 linhas por 2 colunas, resultando em quatro subplots.
2. **Desenho de dados:**  
   Em cada subplot, é desenhada uma função diferente (seno, cosseno, tangente e exponencial decrescente), cada uma com sua própria cor e título.
3. **plt.tight\_layout():**  
   Esta função reorganiza a posição e o espaçamento dos elementos no gráfico, como títulos e rótulos, para evitar que se sobreponham ou fiquem cortados.
4. **fig.subplots\_adjust(wspace=0.4, hspace=0.4):**  
   Caso sejam necessários ajustes mais finos após o tight\_layout(), este comando permite definir manualmente o espaçamento entre subplots.
   * wspace=0.4 aumenta o espaço horizontal entre colunas de subplots.
   * hspace=0.4 aumenta o espaço vertical entre as linhas de subplots.
5. **plt.show():**  
   Exibe a figura final com os quatro subplots, agora bem espaçados e sem sobreposição.

A group of graphs with numbers

Description automatically generated

### 10.2 Combinação de Diferentes Tipos de Gráficos

import matplotlib.pyplot as plt

import numpy as np

fig, ax1 = plt.subplots()

ax2 = ax1.twinx()

meses = np.arange(1,13)

vendas = [230,210,250,260,270,300,310,305,290,280,320,330]

ax1.bar(meses, vendas, color='gray', alpha=0.5, label='Vendas')

ax2.plot(meses, np.log(vendas), color='red', label='Log(Vendas)')

ax1.set\_xlabel("Mês")

ax1.set\_ylabel("Vendas", color='gray')

ax2.set\_ylabel("Log(Vendas)", color='red')

fig.suptitle("Combinação de Gráficos")

fig.tight\_layout()

plt.show()

Este código demonstra como combinar diferentes tipos de gráficos e escalas num mesmo conjunto de eixos, possibilitando comparar informações relacionadas, mas com magnitudes ou interpretações diferentes. A estratégia utilizada é a criação de um segundo eixo Y (um eixo secundário) sobreposto ao primeiro, de forma a apresentar dados em duas escalas distintas, sem perder a clareza na leitura do gráfico.

No início, import matplotlib.pyplot as plt e import numpy as np garantem o acesso às funções de criação de gráficos e aos recursos numéricos do NumPy, respetivamente.

A linha fig, ax1 = plt.subplots() cria uma figura e um eixo (ax1) para o gráfico principal. Logo a seguir, ax2 = ax1.twinx() gera um segundo eixo Y que partilha o mesmo eixo X que ax1, mas mantém uma escala Y independente. Esta abordagem é útil quando se deseja comparar variáveis com ordens de grandeza muito diferentes ou com interpretações diferentes, porém relacionadas ao mesmo eixo X. Assim, ax1 e ax2 referem-se a dois eixos Y sobrepostos, alinhados horizontalmente.

A variável meses é criada através de np.arange(1,13), produzindo um array com os valores de 1 a 12, representando meses do ano. Em vendas, temos uma lista de valores correspondentes às vendas mensais, permitindo visualizar a evolução ao longo desses 12 pontos do tempo.

O comando ax1.bar(meses, vendas, color='gray', alpha=0.5, label='Vendas') desenha um gráfico de barras no eixo principal (ax1). Cada barra representa o valor de vendas num determinado mês. A cor cinzenta (gray) e a transparência (alpha=0.5) ajudam a tornar o gráfico visualmente equilibrado, permitindo a sobreposição de outro gráfico sem perder a legibilidade.

Em seguida, ax2.plot(meses, np.log(vendas), color='red', label='Log(Vendas)') cria uma linha sobre o eixo secundário (ax2), usando o logaritmo das vendas. Este passo é ilustrativo: mostrar o log das vendas pode ser útil para avaliar o crescimento relativo ou para normalizar a distribuição, comparando, por exemplo, o ritmo de crescimento em vez dos valores absolutos. A cor vermelha diferencia claramente esta linha do conjunto de barras cinzentas.

Depois, ajustam-se os rótulos dos eixos. ax1.set\_xlabel("Mês") rotula o eixo X, comum a ambos os gráficos. ax1.set\_ylabel("Vendas", color='gray') atribui o rótulo “Vendas” ao eixo Y principal, e o parâmetro color='gray' sugere ao leitor qual eixo (o da esquerda) está associado às barras cinzentas. Do mesmo modo, ax2.set\_ylabel("Log(Vendas)", color='red') rotula o eixo Y secundário, indicando que o eixo da direita está associado ao gráfico de linha vermelho, mostrando os valores em escala logarítmica.

fig.suptitle("Combinação de Gráficos") adiciona um título geral acima da figura, contextualizando o que está a ser visualizado. Em seguida, fig.tight\_layout() ajusta o layout da figura para evitar que rótulos, títulos ou legendas fiquem fora da área visível ou se sobreponham.

Por fim, plt.show() exibe o resultado. O utilizador vê um gráfico com barras cinzentas representando as vendas mensais no eixo Y da esquerda e uma linha vermelha com o logaritmo das vendas no eixo Y da direita. Esta combinação permite comparar, simultaneamente, o valor absoluto das vendas e a sua escala relativa (via log), mantendo o mesmo eixo temporal (os meses) e tornando a análise mais rica e informativa.

A graph with a line going up

Description automatically generated

### 10.3 Exportação

* plt.savefig("grafico.pdf", dpi=300) para alta qualidade.
* Formatos PDF, SVG para impressão e publicações.

Exemplo de utilização:

import matplotlib.pyplot as plt

import numpy as np

# Gerar dados de exemplo

x = np.linspace(0, 10, 100)

y = np.sin(x)

# Criar o gráfico

plt.figure(figsize=(8,6))

plt.plot(x, y, color='blue', linewidth=2, label='Seno')

plt.title("Função Seno")

plt.xlabel("Eixo X")

plt.ylabel("Amplitude")

plt.legend()

plt.grid(True)

# Ajustar layout para evitar cortes de texto

plt.tight\_layout()

# Salvar o gráfico em formato PDF com alta qualidade (300 DPI)

plt.savefig("grafico.pdf", dpi=300)

# Opcional: Salvar também em formato SVG (vetorial)

plt.savefig("grafico.svg")

# Mostrar o gráfico na tela

plt.show()

**O que este código faz:**

1. **Geração de dados:**  
   x = np.linspace(0, 10, 100) cria 100 pontos igualmente espaçados entre 0 e 10. y = np.sin(x) calcula o valor do seno nesses pontos.
2. **Criação do gráfico:**  
   plt.figure(figsize=(8,6)) define o tamanho da figura em polegadas.  
   plt.plot(x, y, ...) desenha a curva do seno.  
   plt.title(), plt.xlabel(), plt.ylabel(), plt.legend() e plt.grid(True) melhoram a legibilidade e informação do gráfico.
3. **Ajuste do layout:**  
   plt.tight\_layout() garante que nenhum elemento (título, rótulos, legenda) fique cortado ou sobreposto.
4. **Salvar o gráfico em PDF:**  
   plt.savefig("grafico.pdf", dpi=300) grava o gráfico no formato PDF (um formato vetorial), com 300 DPI, alta resolução adequada para publicações impressas. O PDF preserva vetores e texto nítido, mantendo a qualidade independentemente do redimensionamento.
5. **Salvar em SVG (opcional):**  
   plt.savefig("grafico.svg") produz um ficheiro SVG, também um formato vetorial, ideal para integrar em documentos LaTeX, relatórios científicos ou sites, mantendo a qualidade independente da escala.
6. **Exibir o gráfico:**  
   plt.show() permite visualizar o gráfico na janela interativa. O gráfico exibido é idêntico ao que foi salvo, mas a visualização interativa é apenas a parte final do processo.

## Capítulo 11: Integração dos Gráficos em Relatórios Automatizados e Aplicações Web

**Objectivo:** Aprender a incluir gráficos em relatórios (LaTeX, Pandoc) e aplicações web (Dash, Voila).

### 11.1 Relatórios Automatizados com LaTeX e Pandoc

* Gerar o gráfico com Matplotlib (.pdf ou .png).
* Incluir no LaTeX com \includegraphics{figuras/vendas.pdf}.
* Automatizar com scripts para atualizar dados, gráficos e documento final.
* Pandoc: Converter Markdown + imagens em PDF, HTML, etc.

### 11.2 Integração em Aplicações Web (Dash, Voila)

* **Dash:** Integrar gráficos Matplotlib como imagens base64 em dashboards interativos.
* **Voila:** Transformar notebooks Jupyter em aplicações web sem precisar de front-end adicional.
* **Flask/Django:** Gerar gráficos dinamicamente no servidor e servir como imagens para clientes web.

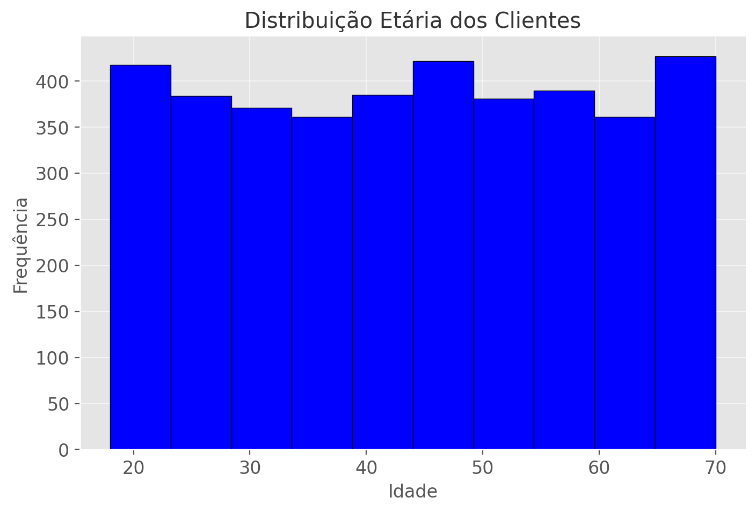
## Capítulo 12: Aplicação Prática com integração com o Pandas e utilização dataset shopping trends

O dataset shopping trends que se encontra no ficheiro shopping\_trends.csv contém informações detalhadas sobre compras, com as seguintes colunas:

1. **Customer ID**: Identificação única do cliente.
2. **Age**: Idade do cliente.
3. **Gender**: Gênero do cliente.
4. **Item Purchased**: Item comprado.
5. **Category**: Categoria do item (ex.: Roupas, Calçados).
6. **Purchase Amount (USD)**: Valor da compra em dólares.
7. **Location**: Localidade do cliente.
8. **Size**: Tamanho do item.
9. **Color**: Cor do item comprado.
10. **Season**: Estação em que a compra foi realizada.
11. **Review Rating**: Avaliação do item por parte do cliente.
12. **Subscription Status**: Se o cliente é assinante (Sim ou Não).
13. **Payment Method**: Método de pagamento utilizado.
14. **Shipping Type**: Tipo de envio escolhido.
15. **Discount Applied**: Indica se houve aplicação de desconto (Sim ou Não).
16. **Promo Code Used**: Indica se foi usado um código promocional (Sim ou Não).
17. **Previous Purchases**: Número de compras anteriores realizadas pelo cliente.
18. **Preferred Payment Method**: Método de pagamento preferido do cliente.
19. **Frequency of Purchases**: Frequência de compras do cliente.

Através da análise dos dados podemos desenhar gráficos que:

* Distribuição etária dos clientes;
* Compras por categoria;
* Média de valor de compras por estação;
* Distribuição das avaliações dos produtos;
* A graph of purple rectangular objects

  Description automatically generated with medium confidenceTop 10 Localidades por número de compras;

A graph of red bars with white text

Description automatically generatedA graph of orange bars

Description automatically generatedA graph with green bars

Description automatically generated

 **Distribuição Etária dos Clientes**:

* Mostra a frequência de clientes em diferentes faixas etárias. É possível identificar grupos etários predominantes, úteis para segmentação de mercado.

 **Compras por Categoria**:

* Um gráfico de barras que destaca quais categorias de produtos (ex.: Roupas, Calçados) são mais compradas. Isso ajuda a identificar os produtos mais populares.

 **Média de Valores de Compras por Estação**:

* Um gráfico de barras que revela a variação no valor médio de compras em diferentes estações do ano (Primavera, Verão, Outono, Inverno). Pode ser usado para planejar estratégias sazonais de vendas.

 **Distribuição das Avaliações dos Produtos**:

* Mostra como os clientes avaliam os produtos. Distribuições inclinadas para valores altos indicam alta satisfação, enquanto valores baixos podem sugerir a necessidade de melhorias.

 **Top 10 Localidades por Número de Compras**:

* Um gráfico de barras que apresenta as 10 localidades com maior número de compras. Essa informação é crucial para otimizar estratégias de marketing e logística.

Segue-se o código Python para gerar os gráficos anteriores:

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

# Carregar os dados do arquivo CSV

data = pd.read\_csv('shopping\_trends.csv')

# Configurar o estilo dos gráficos

plt.style.use('ggplot')

# ---------------- Gráfico 1: Distribuição Etária dos Clientes ----------------

plt.figure(figsize=(8, 5))

data['Age'].hist(bins=10, color='blue', edgecolor='black')

plt.title("Distribuição Etária dos Clientes")

plt.xlabel("Idade")

plt.ylabel("Frequência")

plt.tight\_layout()  # Ajusta o layout para evitar sobreposições

plt.savefig("distribuicao\_etaria.png", dpi=300)  # Salvar o gráfico

plt.show()

# ---------------- Gráfico 2: Compras por Categoria ----------------

plt.figure(figsize=(8, 5))

category\_counts = data['Category'].value\_counts()

category\_counts.plot(kind='bar', color='purple', edgecolor='black')

plt.title("Compras por Categoria")

plt.xlabel("Categoria")

plt.ylabel("Número de Compras")

plt.tight\_layout()

plt.savefig("compras\_por\_categoria.png", dpi=300)

plt.show()

# ---------------- Gráfico 3: Média de Valores de Compra por Estação ----------------

plt.figure(figsize=(8, 5))

season\_avg = data.groupby('Season')['Purchase Amount (USD)'].mean()

season\_avg.plot(kind='bar', color='green', edgecolor='black')

plt.title("Média de Valor de Compras por Estação")

plt.xlabel("Estação")

plt.ylabel("Média de Valor de Compras (USD)")

plt.tight\_layout()

plt.savefig("media\_valor\_compras\_estacao.png", dpi=300)

plt.show()

# ---------------- Gráfico 4: Distribuição das Avaliações dos Produtos ----------------

plt.figure(figsize=(8, 5))

data['Review Rating'].hist(bins=10, color='orange', edgecolor='black')

plt.title("Distribuição das Avaliações dos Produtos")

plt.xlabel("Avaliação")

plt.ylabel("Frequência")

plt.tight\_layout()

plt.savefig("distribuicao\_avaliacoes.png", dpi=300)

plt.show()

# ---------------- Gráfico 5: Top 10 Localidades por Número de Compras ----------------

plt.figure(figsize=(10, 6))

location\_counts = data['Location'].value\_counts().head(10)  # Top 10 localidades

location\_counts.plot(kind='bar', color='red', edgecolor='black')

plt.title("Top 10 Localidades por Número de Compras")

plt.xlabel("Localidade")

plt.ylabel("Número de Compras")

plt.tight\_layout()

plt.savefig("top10\_localidades.png", dpi=300)

plt.show()