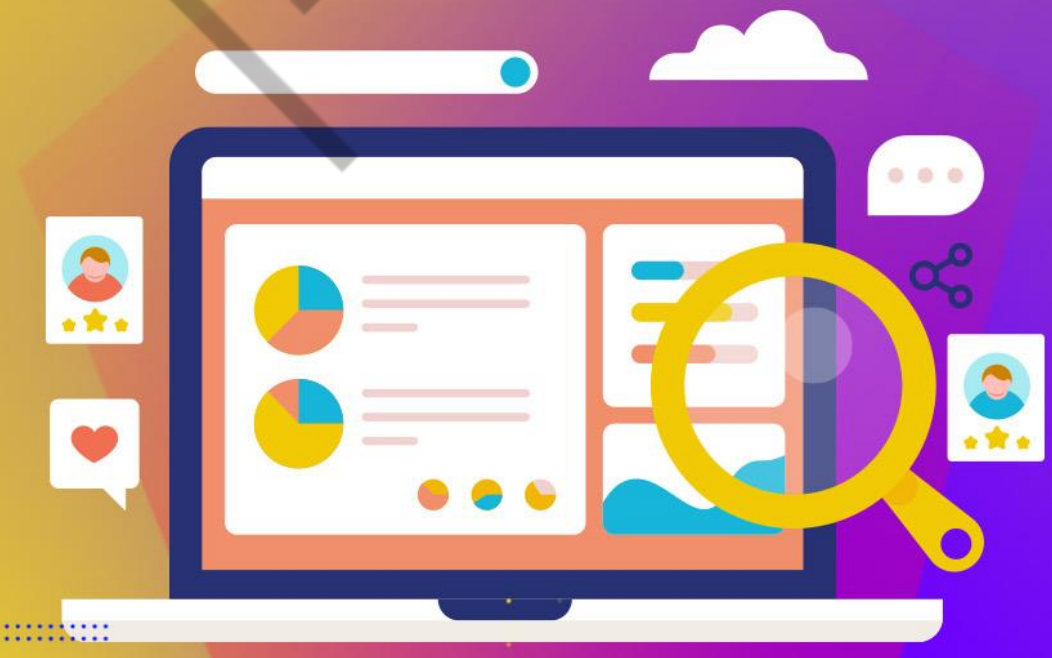
COLHEITA DE DADOS E INSIGHTS - DADOS VALIOSOS E MADUROS

OS GRÁFICOS TAMBÉM FALAM



5

LISTA DE FIGURAS

LISTA DE CÓDIGOS-FONTE

LISTA DE COMANDOS DE PROMPT

SUMÁRIO

1 OS GRÁFICOS TAMBÉM FALAM

1.1 Introdução a visualização de dados

A visualização de dados é um recurso fundamental no campo da ciência e análise de dados, permitindo a conversão de dados brutos em gráficos e figuras informativos e compreensíveis. Essa técnica não apenas facilita a interpretação de grandes volumes de dados, mas também ajuda a identificar padrões, tendências e anomalias que podem não ser imediatamente visíveis em outros formatos de visualização, como listas e tabelas. Por meio dessa abordagem, é possível transformar números abstratos em representações visuais mais fáceis de interpretar.

Painéis bem elaborados podem comunicar informações complexas de forma clara e intuitiva, tornando-se essenciais para a tomada de decisões baseadas em dados. Além disso, eles permitem que analistas e cientistas de dados forneçam *insights* e resultados de maneira acessível a diversas audiências, desde especialistas técnicos até analistas de negócio e *stakeholders* de nível executivo. A capacidade de apresentar dados de forma visual facilita a compreensão das informações, tornando mais eficiente o processo de decisão e ação baseada em dados.

Atualmente, existem aplicações práticas em diversas áreas que fazem uso da visualização de dados. Na área da saúde, pode ser utilizada para monitorar surtos de doenças, rastrear a eficácia de tratamentos e otimizar a alocação de recursos. No setor financeiro, *dashboards* são essenciais para acompanhar o desempenho de investimentos, analisar riscos e identificar oportunidades de mercado. Na educação, as visualizações de dados podem ajudar a monitorar o progresso dos alunos, identificar áreas de melhoria e personalizar a experiência de aprendizado.

No mercado existem diversos softwares proprietários para construção de *dashboards* e visualização de dados, como Tableau, Power BI e Quicksight, cada um com suas próprias características e funcionalidades. No entanto, também existem poderosas bibliotecas open source que oferecem flexibilidade e personalização para os desenvolvedores. No contexto da linguagem Python, podemos destacar entre as mais populares o Matplotlib e o Seaborn, que veremos a seguir.

2 MATPLOTLIB

Matplotlib é uma biblioteca de visualização de dados versátil, amplamente utilizada pela comunidade Python ao redor do mundo. Desenvolvida em 2012, pelo neurobiólogo americano John D. Hunter, ela permite a criação de gráficos 2D e 3D de alta qualidade, com inúmeras possibilidades, desde simples visualizações lineares até complexos *dashboards* personalizados. A estrutura da biblioteca é formada por diversos componentes independentes, incluindo a interface *pyplot*, que fornece uma maneira conveniente de criar gráficos de forma procedural. Além disso, a sua arquitetura orientada a objetos permite um controle refinado sobre todos os aspectos de layout dos painéis, incluindo estilos de linha, cores, rótulos, eixos e muito mais.

No contexto científico, o Matplotlib é essencial para a visualização de dados experimentais, resultados de simulações e análises estatísticas. Na área empresarial, essa biblioteca é frequentemente utilizada para criar *dashboards* que auxiliam na tomada de decisões. Sua flexibilidade é uma característica fundamental para usuários que necessitam de visualizações específicas e a possibilidade de integração com outras bibliotecas, como NumPy e Pandas, tornando-a uma ferramenta indispensável para cientistas, engenheiros e analistas em diversas áreas. Além disso, o Matplotlib é a base para outras bibliotecas de visualização, incluindo o Seaborn (Matplotlib, 2024).

2.1 Instalação

A biblioteca Matplotlib já vem instalada por padrão com a distribuição Anaconda. Porém, você também pode utilizar o gerenciador de pacotes *pip* para realizar uma instalação independente via terminal, com o seguinte comando:

```
pip install matplotlib
```

Comando de prompt 1 - Comando para instalação da biblioteca Matplotlib
Fonte: Elaborado pelo autor (2024)

2.2 Como utilizar

Para começar a utilizar o Matplotlib, é necessário importar a biblioteca em seu script Python. A convenção é importar o módulo *pyplot* seguido pelo codinome “plt”, facilitando o acesso às suas funções e métodos, como especificado no exemplo a seguir:

```
import matplotlib.pyplot as plt
```

Código-fonte 1 - Exemplo de declaração da biblioteca
Fonte: Elaborado pelo autor (2024)

2.3 Tipos de visualizações

O Matplotlib oferece uma ampla gama de visualizações, entre as principais estão: os gráficos de linha, que são ideais para apresentar as tendências ao longo do tempo; os gráficos de barras, úteis para comparar diferentes categorias; os gráficos de dispersão, que revelam a relação entre duas variáveis; e os histogramas para mostrar a distribuição de um conjunto de dados. Além disso, a biblioteca também suporta gráficos de área para representar acúmulo ao longo do tempo, gráficos de pizza para exibir proporções, e gráficos de caixa para descrever a variabilidade dos dados. Cada um desses tipos de visualizações pode ser altamente customizado, permitindo ajustes finos para atender às necessidades específicas dos usuários.

2.3.1 Gráficos de linha

Os gráficos de linha são ferramentas adequadas para visualizar a evolução de dados ao longo do tempo, permitindo identificar tendências e padrões com clareza.

```
# Definindo a área de plotagem
plt.figure(figsize=(10, 6))

# Dados de exemplo
x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
y = [10, 8, 12, 11, 11, 9, 13, 15, 14, 12]

# Criando o gráfico com os eixos x e y
plt.plot(x, y, label='Ações da empresa XYZ')

# Configurando as propriedades do gráfico
plt.xlabel('Dia')
plt.ylabel('Valor')
plt.title('Gráfico de Linha com Matplotlib')
plt.legend()
plt.grid(True)
plt.xticks(x) # Definindo os valores do eixo x

# Exibindo o gráfico
plt.show()
```

Código-fonte 2 - Script para geração de um gráfico de linhas
Fonte: Elaborado pelo autor (2024)

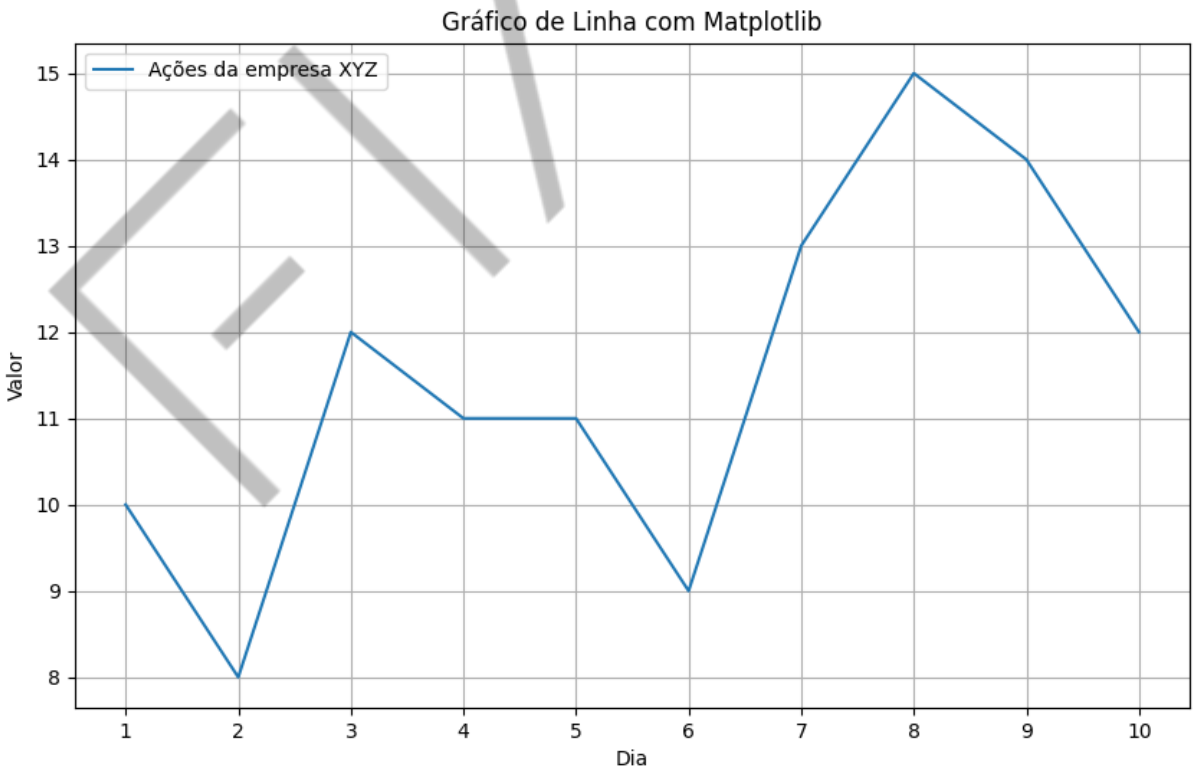


Figura 1 - Gráfico de linha simulando a oscilação no valor de uma ação em 10 dias
Fonte: Elaborado pelo autor (2024)

2.3.2 Gráficos de barra

Gráficos de barra são ideais para comparar valores entre diferentes categorias, proporcionando uma visualização clara e intuitiva das diferenças entre os dados.

```
# Definindo a área de plotagem
plt.figure(figsize=(12, 6))

# Dados de exemplo
categorias = ['A', 'B', 'C', 'D', 'E', 'F']
valores = [10, 20, 15, 30, 25, 22]

# Gráfico de barras
plt.bar(categorias, valores, color='orange')

# Configurando as propriedades do gráfico
plt.xlabel('Categoria')
plt.ylabel('Valor')
plt.title('Gráfico de Barras')

# Exibindo o gráfico
plt.show()
```

Código-fonte 3 - Script para geração de um gráfico de barras
Fonte: Elaborado pelo autor (2024)

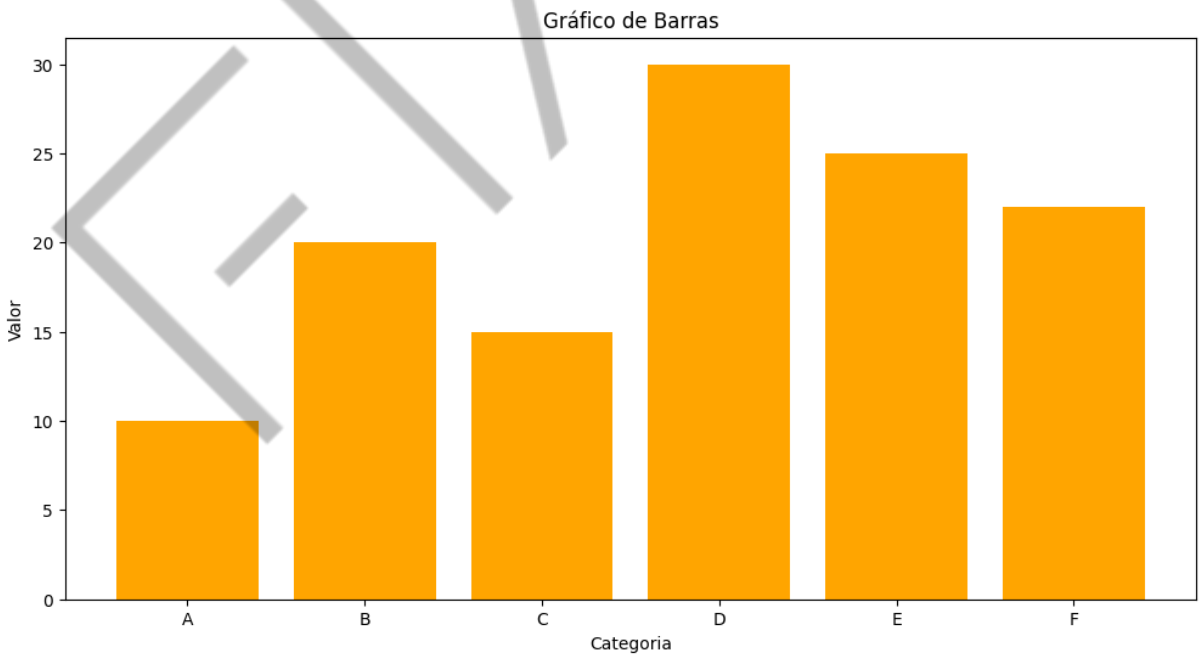


Figura 2 - Gráfico de barras simples
Fonte: Elaborado pelo autor (2024)

2.3.3 Gráficos de dispersão

Os gráficos de dispersão são eficazes para analisar a relação entre duas variáveis, revelando padrões, tendências e possíveis correlações nos dados.

```
import numpy as np

# Definindo a área de plotagem
plt.figure(figsize=(12, 4))

# Fixando a inicialização dos números aleatórios
np.random.seed(0)

# Dados de exemplo
x = np.random.randn(100)
y = np.random.randn(100)

# Criando o gráfico de dispersão
plt.scatter(x, y, color='red')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Gráfico de Dispersão')
plt.grid(True)

# Exibindo o gráfico
plt.show()
```

Código-fonte 4 - Script para geração de um gráfico de dispersão
Fonte: Elaborado pelo autor (2024)

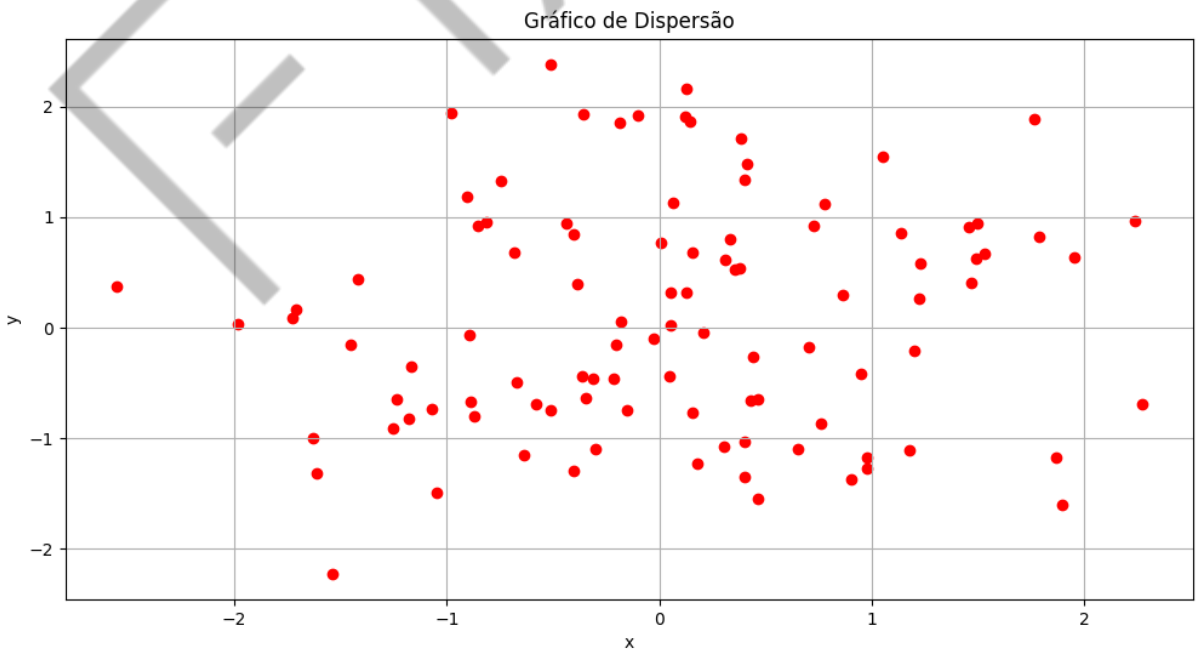


Figura 3 - Gráfico de dispersão sem sinais claros de correlação
Fonte: Elaborado pelo autor (2024)

2.3.4 Histogramas

Os histogramas são excelentes para visualizar a distribuição de um conjunto de dados, destacando a frequência de ocorrências em diferentes intervalos.

```
import numpy as np

# Definindo a área de plotagem
plt.figure(figsize=(12, 6))

# Fixando a inicialização dos números aleatórios
np.random.seed(0)

# Amostras aleatórias com distribuição normal
data = np.random.randn(1000)

# Criando o histograma
plt.hist(data, bins=30, color='cyan')

plt.xlabel('Valor')
plt.ylabel('Frequência')
plt.title('Histograma')
plt.grid(True)

# Exibindo o gráfico
plt.show()
```

Código-fonte 5 - Script para geração de histograma
Fonte: Elaborado pelo autor (2024)

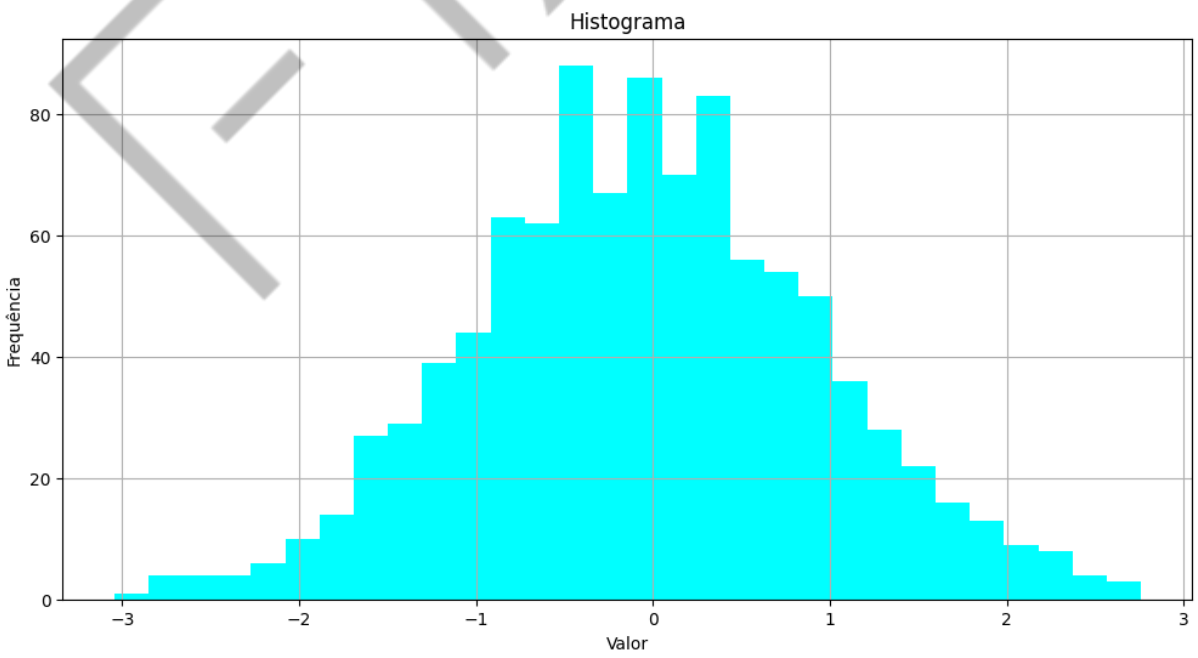


Figura 4 - Histograma de dados aleatórios, com distribuição normal
Fonte: Elaborado pelo autor (2024)

2.3.5 Gráficos de área

Os gráficos de área são úteis para mostrar a contribuição de diferentes componentes ao longo do tempo, enfatizando as variações cumulativas.

```
# Definindo a área de plotagem
plt.figure(figsize=(12, 6))

# Dados de exemplo
anos = [2015, 2016, 2017, 2018, 2019, 2020]
c1 = [3, 4, 6, 8, 7, 9]
c2 = [2, 3, 4, 5, 4, 6]
c3 = [1, 2, 2, 3, 3, 4]
rotulos = ['Categoria 1', 'Categoria 2', 'Categoria 3']

# Criar o gráfico de área
plt.stackplot(anos, c1, c2, c3, labels=rotulos, alpha=0.8)

# Adicionar título e rótulos aos eixos
plt.title('Gráfico de Área Exemplo')
plt.xlabel('Ano')
plt.ylabel('Valores')
plt.legend()

# Mostrar o gráfico
plt.show()
```

Código-fonte 6 - Script para geração de um gráfico de área
Fonte: Elaborado pelo autor (2024)

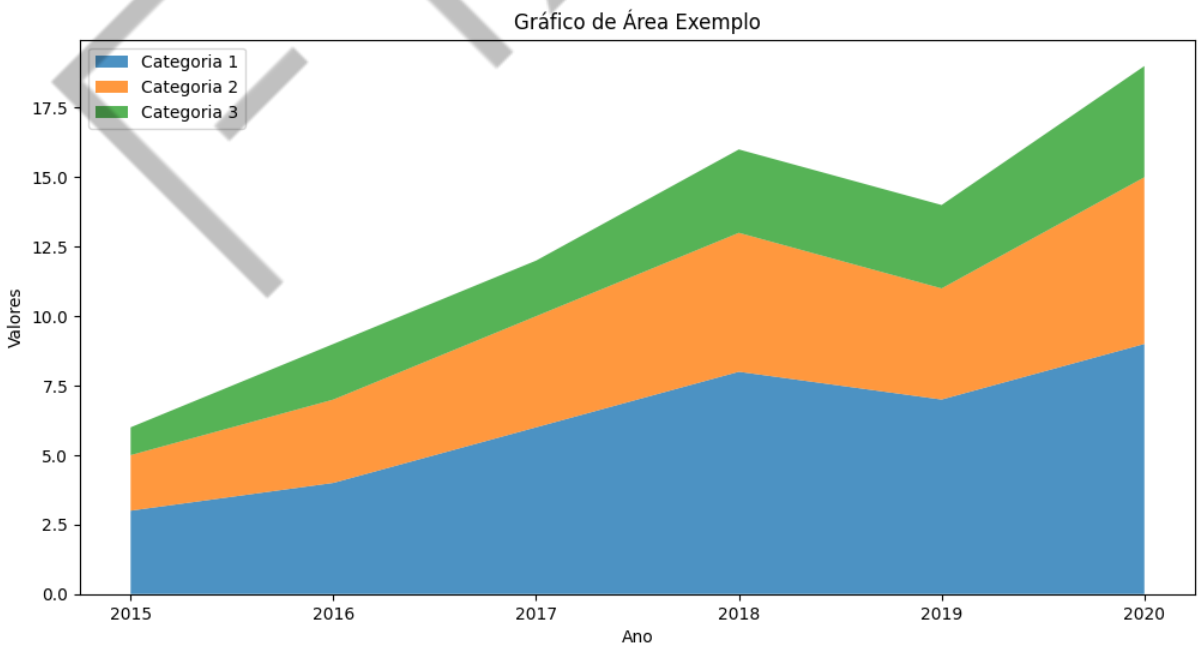


Figura 5 - Gráfico de área sobre valor acumulado em uma distribuição de tempo
Fonte: Elaborado pelo autor (2024)

2.3.6 Gráficos de pizza

Os gráficos de pizza são ideais para visualizar a proporção de diferentes categorias em um conjunto de dados, facilitando a compreensão das partes que compõem o todo. No entanto, esse tipo de gráfico pode ser difícil de interpretar quando há muitas divisões ou quando as diferenças entre os valores são pequenas, ou seja, nessas situações podem ocasionar uma visualização confusa e menos precisa das informações comparadas.

```
# Definindo a área de plotagem
plt.figure(figsize=(4, 4))

# Dados de exemplo
ids = ['A', 'B', 'C', 'D']
valores = [15, 30, 45, 10]
cores = ['gold', 'yellowgreen', 'lightcoral', 'lightskyblue']

# Criar o gráfico de pizza
plt.pie(valores, labels=ids, colors=cores, autopct='%1.1f%%')

# Adicionar título
plt.title('Gráfico de Pizza')

# Mostrar o gráfico
plt.show()
```

Código-fonte 7 - Script para geração de um gráfico de pizza
Fonte: Elaborado pelo autor (2024)

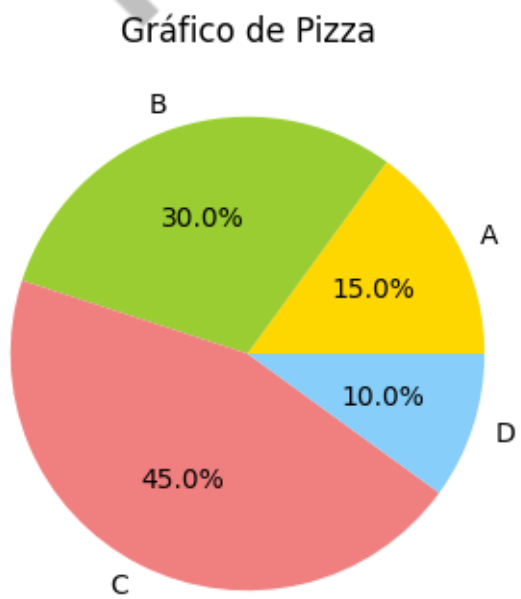


Figura 6 - Gráfico de pizza cheio, com proporção de 4 categorias
Fonte: Elaborado pelo autor (2024)

2.3.7 Gráficos de caixa (*boxplots*)

Os gráficos de caixa, mais conhecidos como *boxplots*, são ferramentas eficazes para visualizar a distribuição, a mediana e a variabilidade dos dados, destacando outliers e proporcionando uma visão clara da dispersão e assimetria dos conjuntos.

```
import numpy as np

# Definindo a área de plotagem
plt.figure(figsize=(12, 4))

# Dados de exemplo
np.random.seed(10)
dados = [np.random.normal(0, std, 100) for std in range(1, 4)]

# Criar o boxplot
plt.boxplot(dados, tick_labels=['A', 'B', 'C'])

# Adicionar título e rótulos aos eixos
plt.title('Exemplo de Boxplot')
plt.xlabel('Grupos')
plt.ylabel('Valores')

# Mostrar o gráfico
plt.show()
```

Código-fonte 8 - Script para geração de um gráfico de caixa (*boxplot*)
Fonte: Elaborado pelo autor (2024)

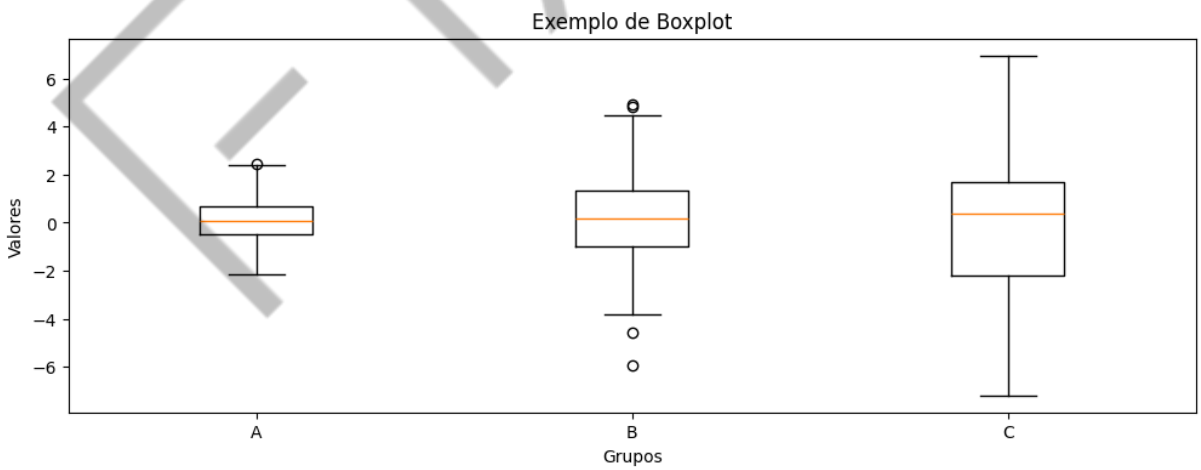


Figura 7 - Gráfico de caixa, ilustrando três conjuntos aleatórios
Fonte: Elaborado pelo autor (2024)

2.3.8 Combinações e outros recursos

Além das opções apresentadas, a flexibilidade da biblioteca Matplotlib permite inúmeras variações por meio da combinação de gráficos distintos, com o intuito de fornecer a visualização mais adequada para o usuário em cada contexto.

```
import numpy as np

fig, axs = plt.subplots(ncols=2, nrows=2)
ax1, ax2, ax3, ax4 = axs.flat

x, y = np.random.normal(size=(2, 200))
ax1.plot(x, y, 'o')

L = 2 * np.pi
x = np.linspace(0, L)
shift = np.linspace(0, L, 10, endpoint=False)

for s in shift:
    ax2.plot(x, np.sin(x + s), '-')

y1 = np.random.randint(1, 25, size=(1, 5))[0]
ax3.bar(np.arange(5), y1, 0.25, color='yellowgreen')

for c in ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728']:
    xy = np.random.normal(size=2)
    ax4.add_patch(plt.Circle(xy, radius=0.3, color=c))

ax4.axis('equal')
plt.show()
```

Código-fonte 9 - Script para geração de gráficos combinados em uma única visualização
Fonte: Elaborado pelo autor (2024)

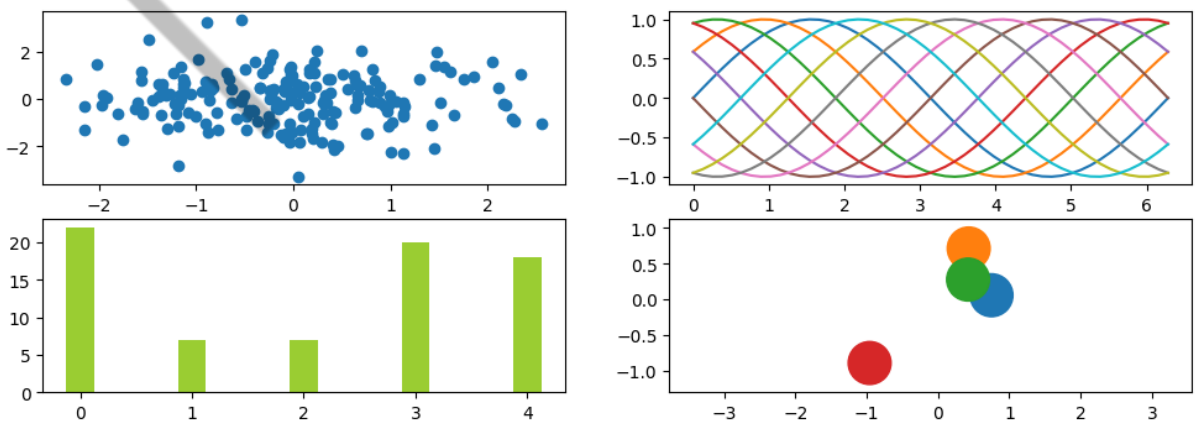


Figura 8 - Opções de gráficos diversas para combinação conforme a demanda
Fonte: Elaborado pelo autor (2024)

3 SEABORN

Seaborn é uma biblioteca de visualização de dados desenvolvida com o objetivo de facilitar a criação de gráficos atraentes e informativos. Desenvolvida a partir do Matplotlib, em 2013, pelo engenheiro de software Michael Waskom, essa biblioteca oferece uma interface de alto nível que simplifica o processo de geração, permitindo a criação de gráficos complexos com menos código e uma estética visual aprimorada. A biblioteca é projetada para trabalhar perfeitamente com DataFrames do Pandas, o que facilita a manipulação de grandes conjuntos de dados (Waskom, 2021).

Além disso, o Seaborn possui uma variedade de estilos de gráficos e temas predefinidos, permitindo a criação rápida de visualizações de dados complexos, como gráficos de regressão, de correlação e de distribuição. A integração nativa com o Matplotlib garante que os usuários possam aproveitar a flexibilidade e a robustez de ambas as bibliotecas, tornando o Seaborn uma ferramenta indispensável para desenvolvedores que buscam criar visualizações de dados claras, informativas e esteticamente agradáveis (Waskom, 2021).

3.1 Instalação

A biblioteca Seaborn já vem instalada por padrão com a distribuição Anaconda. Porém, você também pode utilizar o gerenciador de pacotes *pip* para realizar uma instalação independente via terminal, com o seguinte comando:

```
pip install seaborn
```

Comando de prompt 2 - Comando para instalação da biblioteca Seaborn
Fonte: Elaborado pelo autor (2024)

3.2 Como utilizar

Para começar a utilizar o Seaborn é necessário importar a biblioteca em seu script Python. A convenção é importá-la seguida pelo codinome “sns”, facilitando o acesso posterior às suas funções e métodos, como especificado a seguir:

```
import seaborn as sns
```

Código-fonte 10 - Exemplo de declaração da biblioteca
Fonte: Elaborado pelo autor (2024)

3.3 Tipos de visualizações

O Seaborn é uma biblioteca de visualização de dados em Python que se destaca pela facilidade de uso e pela criação de gráficos estatísticos atraentes e informativos. Entre suas principais visualizações estão: os gráficos de distribuição, excelentes para entender a forma e a dispersão dos dados; os gráficos de regressão, que mostram a relação entre variáveis e incluem a linha de tendência; e os gráficos de categorização, como *boxplots* e *violin plots*, que destacam a variabilidade e a distribuição das categorias analisadas. Além disso, o Seaborn suporta gráficos de calor (*heatmaps*) para visualizar matrizes de correlação, gráficos de pares (*pair plots*) para explorar relações em múltiplas dimensões, e gráficos de enxame (*swarm plots*) para representar dados categóricos com dispersão mínima.

Cada um desses tipos de visualização é projetado para ser altamente personalizável, permitindo ajustes específicos e detalhados, reutilizando a arquitetura orientada a objetos fornecida pelo Matplotlib, visando atender às necessidades específicas dos usuários.

3.3.1 Gráficos de distribuição

Os gráficos de distribuição do Seaborn são ferramentas poderosas para analisar a relação entre duas variáveis, com recursos adicionais como a inclusão de linhas de regressão e a capacidade de diferenciar subgrupos por cor ou marcador.

```
import matplotlib.pyplot as plt

# Dados de exemplo
iris = sns.load_dataset('iris')

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

# Criar o gráfico de dispersão
sns.scatterplot(data=iris, x='sepal_length', y='sepal_width',
hue='species', style='species', palette='deep')

# Adicionar título e rótulos aos eixos
plt.title('Gráfico de Dispersão do Conjunto de Dados Iris')
plt.xlabel('Comprimento da Sépala')
plt.ylabel('Largura da Sépala')

# Mostrar o gráfico
plt.show()
```

Código-fonte 11 - Script para criação de um gráfico de distribuição
Fonte: Elaborado pelo autor (2024)

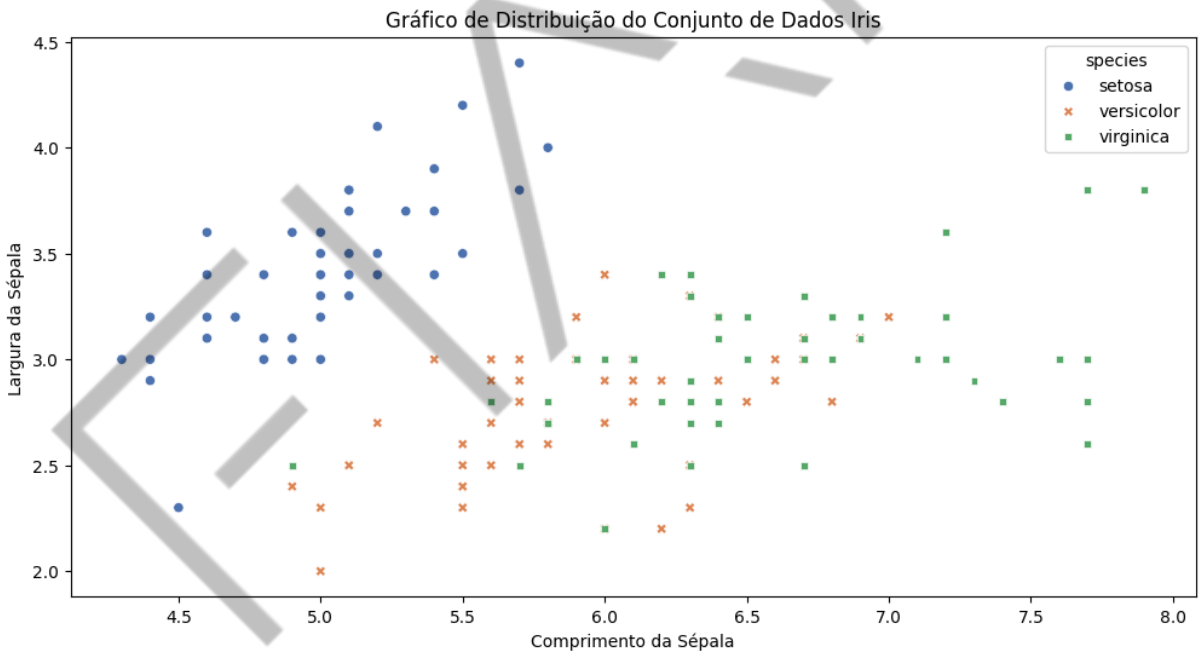


Figura 9 - Gráficos de distribuição, apresentando dados do dataset Iris
Fonte: Elaborado pelo autor (2024)

3.3.2 Gráficos de regressão

Os gráficos de regressão do Seaborn são ideais para visualizar e entender a relação entre duas variáveis com a adição de linhas de tendência e intervalos de confiança, que ajudam a destacar padrões e correlações nos dados.

```
    import matplotlib.pyplot as plt

    # Dados de exemplo
    tips = sns.load_dataset('tips')

    # Definindo a área de plotagem
    plt.figure(figsize=(12, 6))

    # Criar o gráfico de regressão
    sns.regplot(data=tips, x='total_bill', y='tip', ci=95)

    # Adicionar título e rótulos aos eixos
    plt.title('Gráfico de Regressão do Total da Conta vs.
Gorjeta')
    plt.xlabel('Total da Conta')
    plt.ylabel('Gorjeta')

    # Mostrar o gráfico
    plt.show()
```

Código-fonte 12 - Script para criação de um gráfico de regressão
Fonte: Elaborado pelo autor (2024)

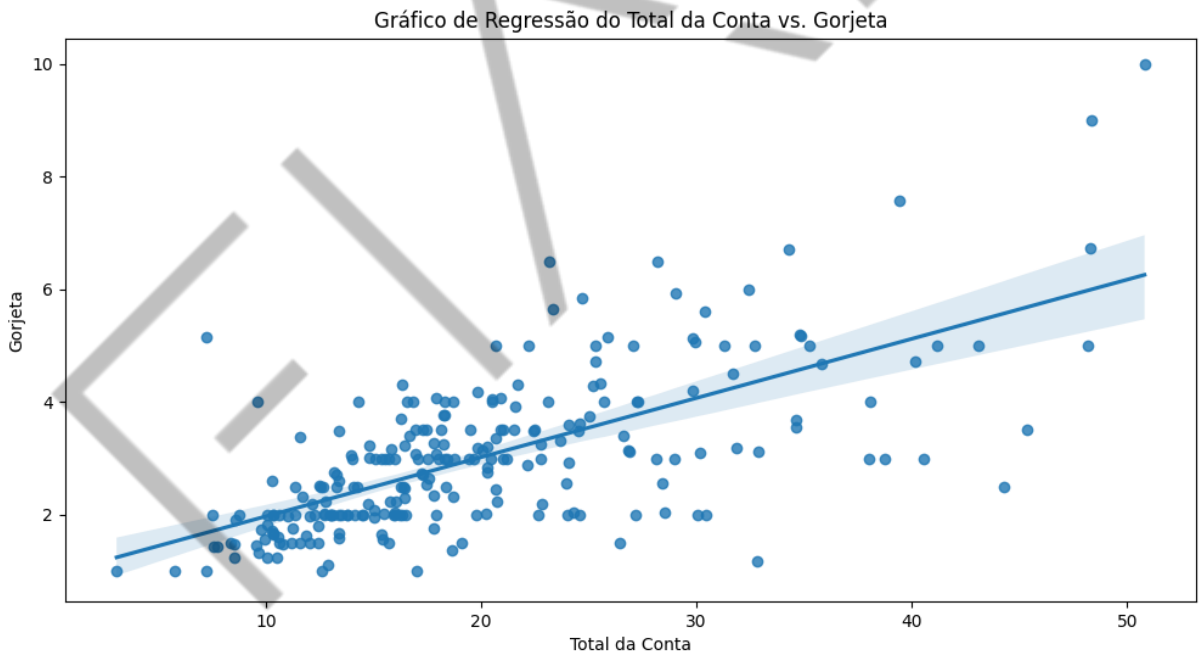


Figura 10 - Gráficos de regressão, apresentando dados do dataset Tips
Fonte: Elaborado pelo autor (2024)

3.3.3 Gráficos de categorização

Os gráficos de categorização do Seaborn, como *boxplots* e *violin plots*, são eficazes para visualizar a distribuição e a variabilidade dos dados em diferentes categorias, destacando padrões entre grupos de maneira clara e objetiva.

```
import matplotlib.pyplot as plt

# Dados de exemplo
tips = sns.load_dataset('tips')

# Definindo a área de plotagem
plt.figure(figsize=(12, 6))

# Criar o boxplot
plt.subplot(1, 2, 1)
sns.boxplot(data=tips, x='day', y='total_bill')

plt.title('Boxplot do Total da Conta por Dia')
plt.xlabel('Dia da Semana')
plt.ylabel('Total da Conta')

# Criar o violin plot
plt.subplot(1, 2, 2)
sns.violinplot(data=tips, x='day', y='total_bill')

plt.title('Violin Plot do Total da Conta por Dia')
plt.xlabel('Dia da Semana')
plt.ylabel('Total da Conta')

# Ajustar layout e mostrar o gráfico
plt.tight_layout()
plt.show()
```

Código-fonte 13 - Script para criação de gráficos de categorização
Fonte: Elaborado pelo autor (2024)

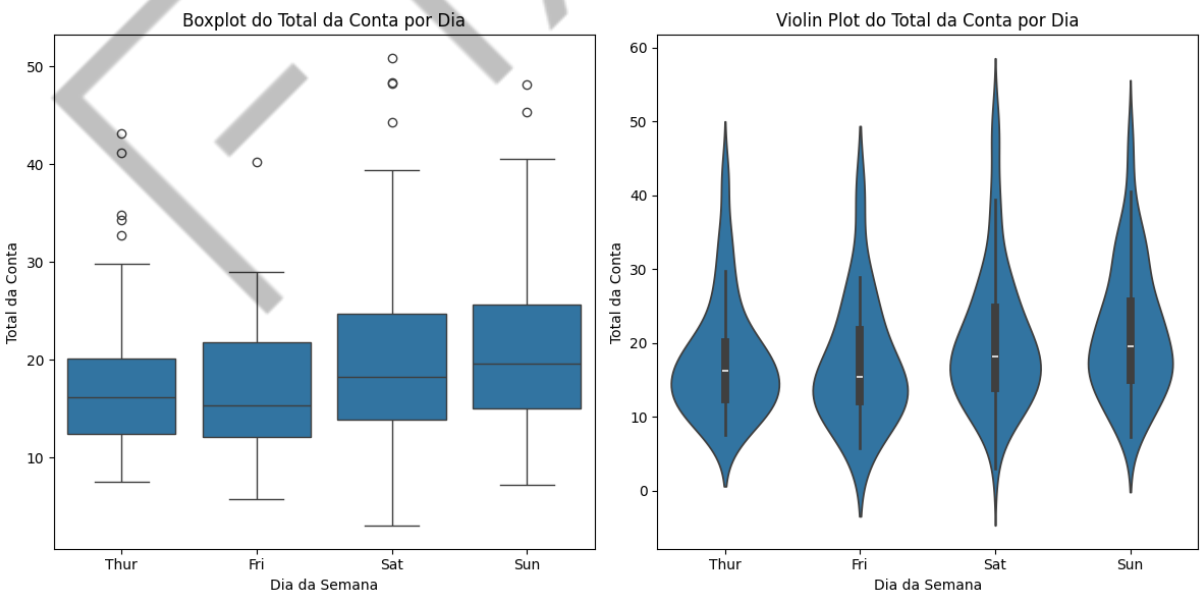


Figura 11 - Gráficos de categorização, apresentando dados do dataset Tips
Fonte: Elaborado pelo autor (2024)

3.3.4 Gráficos de calor

Os gráficos de calor do Seaborn são excelentes para visualizar matrizes de dados, facilitando a identificação de padrões, correlações e concentrações de valores por meio de uma representação colorida das intensidades dos dados.

```
import matplotlib.pyplot as plt

# Carregar o conjunto de dados Iris
iris = sns.load_dataset('iris')
numerical_data = iris.select_dtypes(include=['float64'])

# Calcular a matriz de correlação
corr = numerical_data.corr()

# Criar o gráfico de calor
plt.figure(figsize=(10, 8))
sns.heatmap(corr, fmt='.2f', vmin=-1, vmax=1)

# Adicionar título
plt.title('Mapa de Calor da Correlação das Variáveis')

# Mostrar o gráfico
plt.show()
```

Código-fonte 14 - Script para criação de um gráfico de calor (*heatmap*)
Fonte: Elaborado pelo autor (2024)

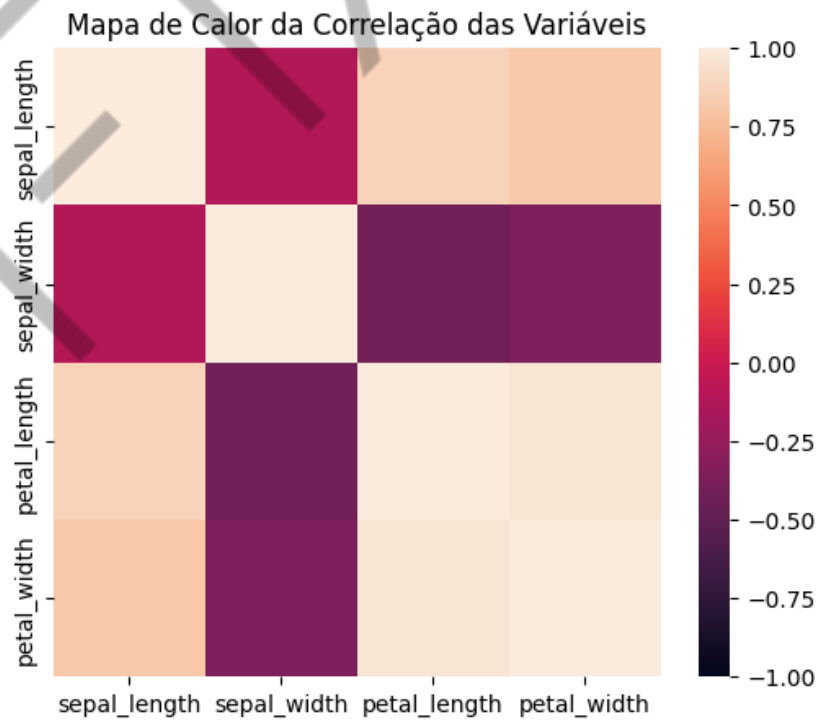


Figura 12 - Gráfico de calor, apresentando a correlação dos dados decimais do dataset Iris
Fonte: Elaborado pelo autor (2024)

3.3.5 Gráficos de pares

Os gráficos de pares, ou *pair plots*, do Seaborn são úteis para explorar relações entre múltiplas variáveis simultaneamente, oferecendo uma visão abrangente das correlações por meio de uma matriz de gráficos de dispersão e histogramas.

```
import matplotlib.pyplot as plt

# Carregar o conjunto de dados Iris
iris = sns.load_dataset('iris')

# Criar o gráfico de pares
plt.figure(figsize=(6, 5))
sns.pairplot(iris, hue='species', diag_kind='kde')

# Adicionar título
plt.suptitle('Gráfico de Pares da Base Iris', y=1.02)

# Mostrar o gráfico
plt.show()
```

Código-fonte 15 - Script para criação de um gráfico de pares (*pair plot*)
Fonte: Elaborado pelo autor (2024)

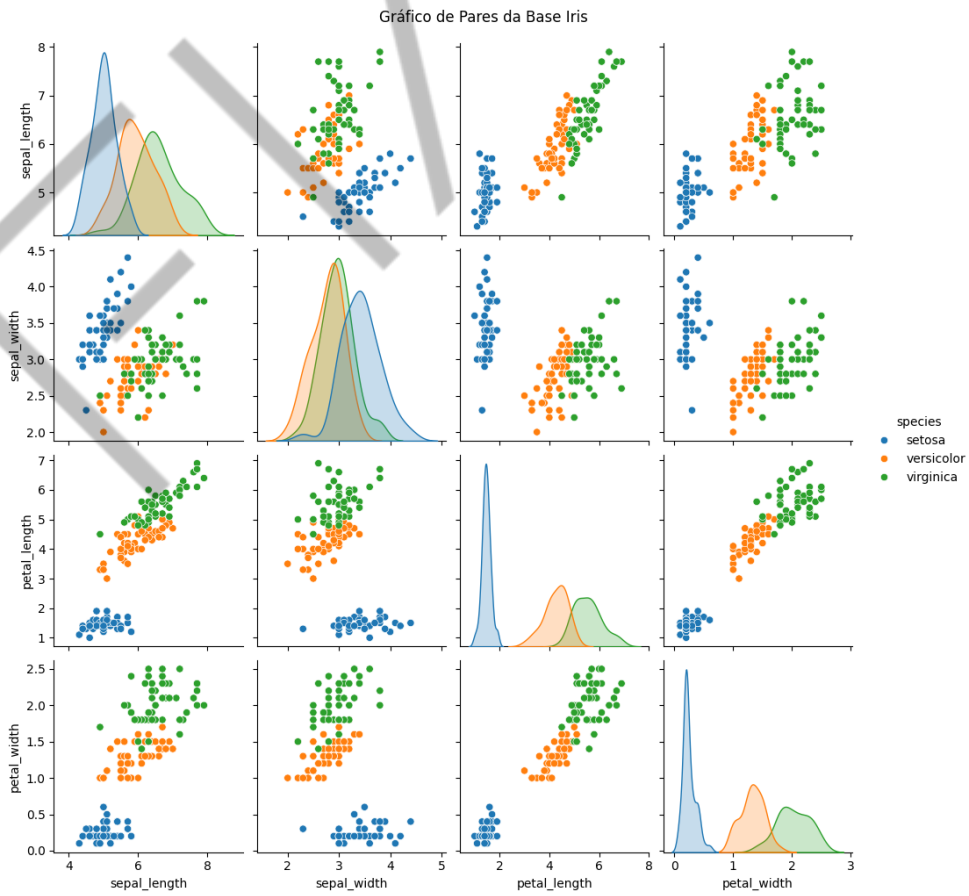


Figura 13 - Gráfico de pares, apresentando correlações e dispersões do dataset Iris
Fonte: Elaborado pelo autor (2024)

3.3.6 Gráficos de enxame

Gráficos de enxame do Seaborn são eficazes para visualizar a distribuição de dados categóricos, exibindo cada ponto de dados individualmente e evitando a sobreposição. Isso facilita a análise de densidade e padrões em pequenos conjuntos.

```
import matplotlib.pyplot as plt

# Carregar o conjunto de dados Tips
tips = sns.load_dataset('tips')

# Criar o gráfico de enxame
plt.figure(figsize=(10, 6))
sns.swarmplot(data=tips, x='day', y='total_bill', hue='time')

# Adicionar título e rótulos aos eixos
plt.title('Gráfico de Enxame do Total da Conta por Dia')
plt.xlabel('Dia da Semana')
plt.ylabel('Total da Conta')

# Mostrar o gráfico
plt.show()
```

Código-fonte 16 - Script para criação de um gráfico de enxame (*swarm plot*)
Fonte: Elaborado pelo autor (2024)

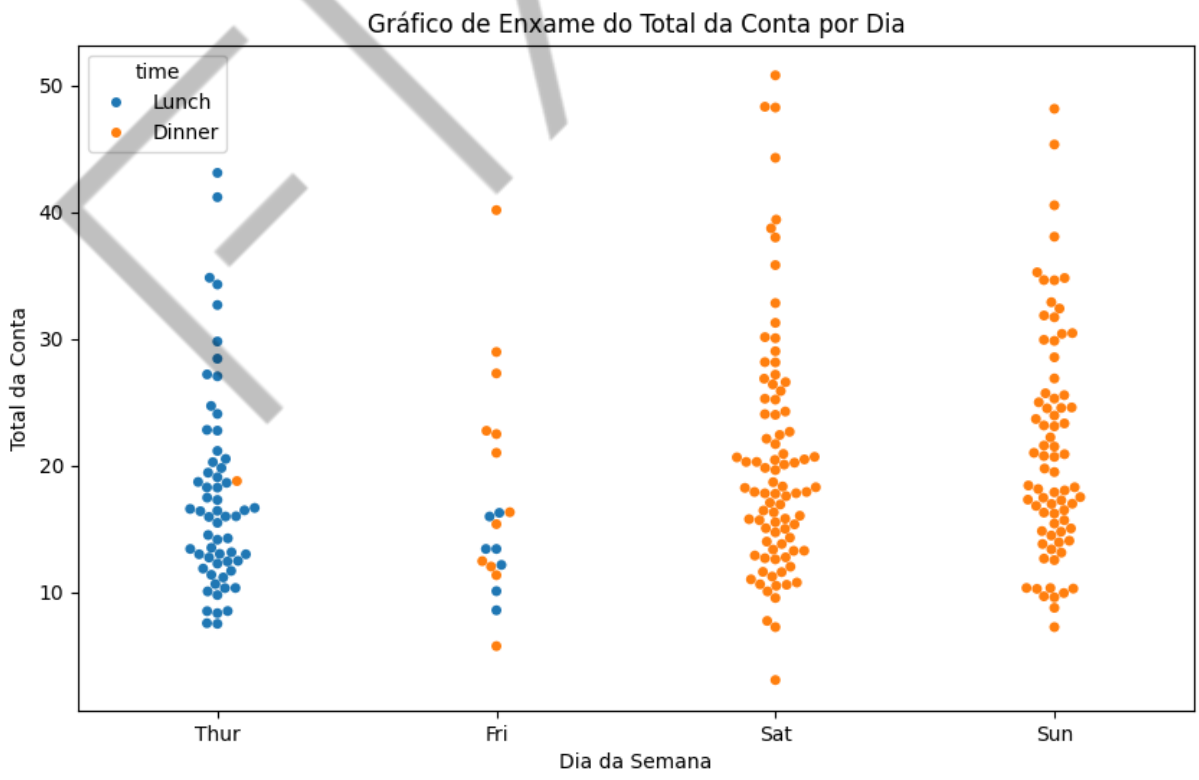


Figura 14 - Gráfico de enxame, apresentando a distribuição de categóricos do dataset Tips
Fonte: Elaborado pelo autor (2024)

3.3.7 Recursos adicionais

Assim como o Matplotlib, a biblioteca Seaborn dispõe de meios para combinar e customizar seus gráficos conforme a necessidade do usuário e de acordo com o contexto da aplicação. Além disso, esse recurso permite a criação de infinitas possibilidades, dependendo apenas da criatividade do desenvolvedor e seu conhecimento sobre as configurações disponíveis na biblioteca.

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

sns.set_theme(style="dark")

# Gaussiana bivariada
n = 10000
mean = [0, 0]
cov = [(2, .4), (.4, .2)]
rng = np.random.RandomState(0)
x, y = rng.multivariate_normal(mean, cov, n).T

# Histograma scatterplot com contorno de densidade
f, ax = plt.subplots(figsize=(12, 6))
sns.scatterplot(x=x, y=y, s=5, color=".15")
sns.histplot(x=x, y=y, bins=50, pthresh=.1, cmap="mako")
sns.kdeplot(x=x, y=y, levels=5, color="w", linewidths=1)
```

Código-fonte 17 - Script para criação de gráficos combinados
Fonte: Elaborado pelo autor (2024)

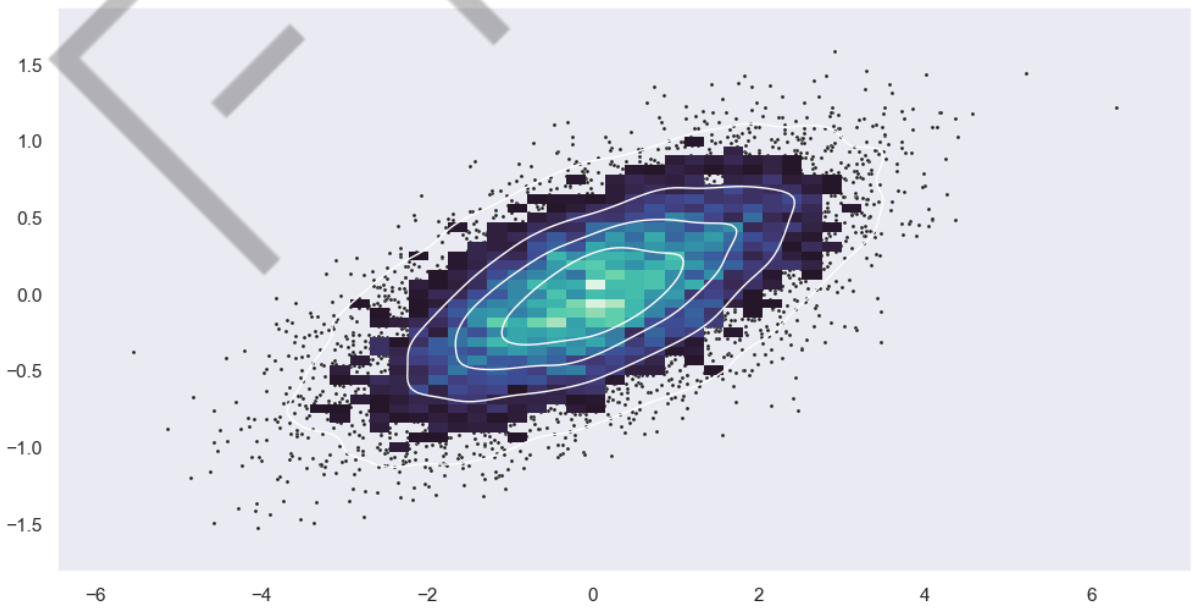


Figura 15 - Histograma scatterplot com contorno de densidade
Fonte: Elaborado pelo autor (2024)

CONCLUSÃO

Os benefícios da visualização de dados incluem a capacidade de detectar rapidamente padrões e anomalias, promover uma compreensão mais profunda do cenário analisado, melhorar a comunicação de *insights* e apoiar a tomada de decisões dinâmicas com embasamento sólido. Ao tornar os dados mais acessíveis e compreensíveis, a visualização de dados empodera os usuários e como resultado, essa prática não apenas facilita a análise, mas também amplifica seu impacto, transformando informações complexas e relevantes em narrativas visuais tangíveis, que podem influenciar o planejamento estratégico em diversos setores.

Nesse contexto, o Matplotlib e o Seaborn são ferramentas poderosas para a construção de visualizações em Python. Enquanto o Matplotlib oferece uma base sólida para a criação de gráficos personalizados e diversos, o Seaborn fornece uma interface mais intuitiva para visualizações e analisar dados com praticidade. Juntas, essas bibliotecas permitem explorar diversas abordagens para realizar a divulgação de dados de maneira eficaz, facilitando a interpretação de informações complexas.

REFERÊNCIAS

MATPLOTLIB: Visualization with Python. **Matplotlib.org**, 2024. Disponível em: <https://matplotlib.org>. Acesso em: 14 ago. 2024.

WASKOM, M. L., Seaborn: Statistical Data Visualization. **Journal of Open Source Software**, 60, 2021, p. 3021-3025.