Atlântico Avanti

Sejam Bem-vindos ao curso básico em machine learning

Atlântico Avanti

Módulo 2

Visualização e análise de dados

Ementa

- Módulo 2
  - Visualização e análise de dados
    - Unidade 1
      - Visualização de dados com a biblioteca matplotlib
        - Partes de uma Figura
        - Tipos de entradas para funções de plotagem
        - Gráficos repetidos com diferentes conjuntos de dados
        - Estilos e Cores
        - Largura e estilos de linhas e tamanhos de marcadores
        - Anotações e Legendas
        - Escalas dos eixos
        - Trabalhando com várias figuras e eixos

Ementa

- Módulo 2
  - Visualização e análise de dados
    - Unidade 2
      - Análise de dados com a biblioteca pandas
        - Limpeza e pré-processamento de dados
          - ➢ Remoção de dados duplicados
          - ➢ Lidar com valores ausentes (NaN) através de preenchimento ou remoção
          - ➢ Tratamento de outliers
          - ➢ Conversão de tipos de dados
          - ➢ Renomeação de colunas e reindexação de dados
        - Exploração de dados
          - ➢ Estatísticas descritivas, como média, mediana, desvio padrão, mínimo e máximo
          - ➢ Contagem de valores únicos em uma coluna
          - ➢ Agregação de dados
          - ➢ Filtragem e seleção de dados com base em critérios específicos
          - ➢ Amostragem de dados para análise exploratória

Ementa

- Módulo 2
  - Visualização e análise de dados
    - Unidade 2
      - Análise de dados com a biblioteca pandas
        - Manipulação de dados
          - ➢ Fusão (merge) de múltiplos dataframes com base em chaves comuns
          - ➢ Junção (join) de dados com base em colunas compartilhadas
          - ➢ Concatenação de dataframes vertical ou horizontalmente
          - ➢ Divisão de dados em conjuntos de treinamento e teste
        - Visualização de dados
          - ➢ Gráficos de barras, linhas, dispersão, histogramas

Módulo 2: Introdução

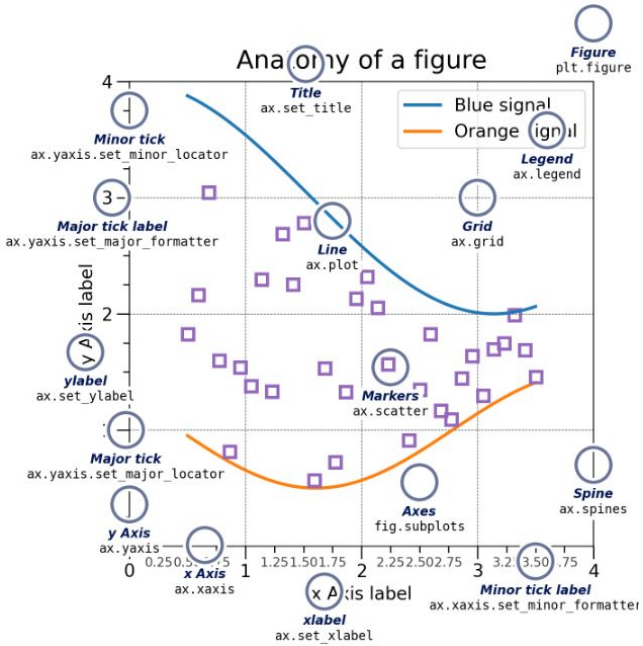
Visualização e análise de dados são práticas fundamentais em várias áreas, especialmente em tomada de decisões baseada em informações. Esses processos envolvem a transformação de dados brutos em gráficos, tabelas ou outras formas de representação visual para extrair informações valiosas, identificar padrões, tendências, insights e tomar decisões mais embasadas.

Módulo 2: Unidade 1 - Visualização de dados com a biblioteca matplotlib

A visualização de dados tem como objetivo representar informações e padrões complexos de forma visual e compreensível. A visualização de dados desempenha um papel crucial na análise de dados, na comunicação de informações e no suporte à tomada de decisões informadas. Através de gráficos, gráficos, mapas, diagramas e outras representações visuais, a visualização de dados permite que as pessoas explorem, entendam e apresentem dados de maneira mais intuitiva e acessível.

Módulo 2: Unidade 1 – Visualização de dados com a biblioteca matplotlib

Partes de uma Figura



Módulo 2: Unidade 1 - Visualização de dados com a biblioteca matplotlib

Construção de figuras

O Matplotlib representa graficamente seus dados em Figures (por exemplo, janelas, widgets Jupyter, etc.), cada um dos quais pode conter um ou mais Axes, uma área onde os pontos podem ser especificados em termos de coordenadas xy (ou theta-r em um gráfico polar, xyz em um gráfico 3D, etc.). A maneira mais simples de criar uma figura com eixos é usando pyplot.subplots. Podemos usar Axes.plot para desenhar alguns dados nos eixos:

Módulo 2: Unidade 1 - Visualização de dados com a biblioteca matplotlib

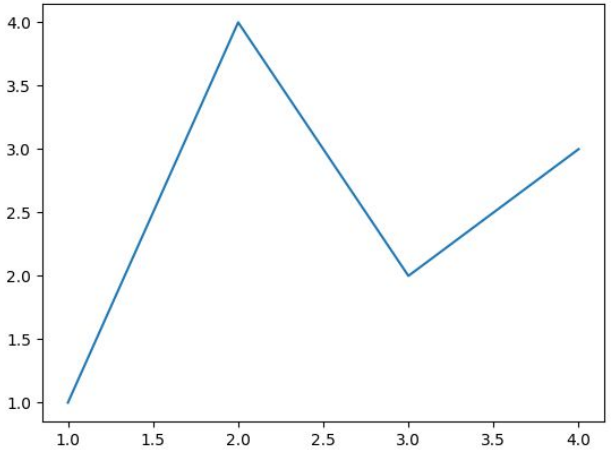
Construção de figuras

O Matplotlib representa graficamente seus dados em Figures (por exemplo, janelas, widgets Jupyter, etc.), cada um dos quais pode conter um ou mais Axes, uma área onde os pontos podem ser especificados em termos de coordenadas xy (ou theta-r em um gráfico polar, xyz em um gráfico 3D, etc.). A maneira mais simples de criar uma figura com eixos é usando pyplot.subplots. Podemos usar Axes.plot para desenhar alguns dados nos eixos:

Módulo 2: Unidade 1 – Visualização de dados com a biblioteca matplotlib

Construção de figuras

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots()  # Create a figure containing a single axes.
ax.plot([1, 2, 3, 4], [1, 4, 2, 3])  # Plot some data on the axes.
plt.show()  # Show the plot.
```



Módulo 2: Unidade 1 - Visualização de dados com a biblioteca matplotlib

Construção de figuras

Tipos de entradas para funções de plotagem

Funções de plotagem esperam **numpy.array** ou **numpy.ma.masked_array** como entrada, ou objetos que podem ser passados para **numpy.asarray**. Classes que são semelhantes a arrays ('semelhantes a arrays'), como pandas objetos de dados, **numpy.matrix** podem não funcionar conforme o esperado. A convenção comum é convertê-los em **numpy.array** objetos antes da plotagem. Por exemplo, para converter um **numpy.matrix**.

```
b = np.matrix([[1, 2], [3, 4]])
b_asarray = np.asarray(b)
```

Módulo 2: Unidade 1 - Visualização de dados com a biblioteca matplotlib

Construção de figuras

Tipos de entradas para funções de plotagem

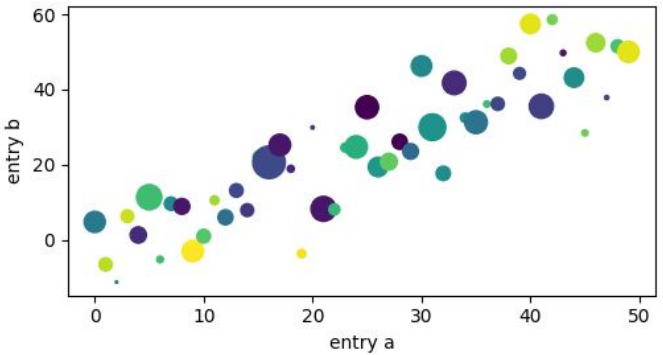
A maioria dos métodos também analisará um objeto endereçável como um dict, a numpy.recarray ou pandas.DataFrame. Matplotlib permite que você forneça o data argumento de palavra-chave e gere gráficos passando as strings correspondentes às variáveis x e y.

Módulo 2: Unidade 1 – Visualização de dados com a biblioteca matplotlib

Construção de figuras

Tipos de entradas para funções de plotagem

```
import matplotlib.pyplot as plt
import numpy as np
np.random.seed(19680801)  # seed the random number generator.
data = {'a': np.arange(50),
        'c': np.random.randint(0, 50, 50),
        'd': np.random.randn(50)}
data['b'] = data['a'] + 10 * np.random.randn(50)
data['d'] = np.abs(data['d']) * 100
fig, ax = plt.subplots(figsize=(5, 2.7))
fig.set_constrained_layout(True)  # Define o layout restrito
ax.scatter('a', 'b', c='c', s='d', data=data)
ax.set_xlabel('entry a')
ax.set_ylabel('entry b')
plt.show()
```

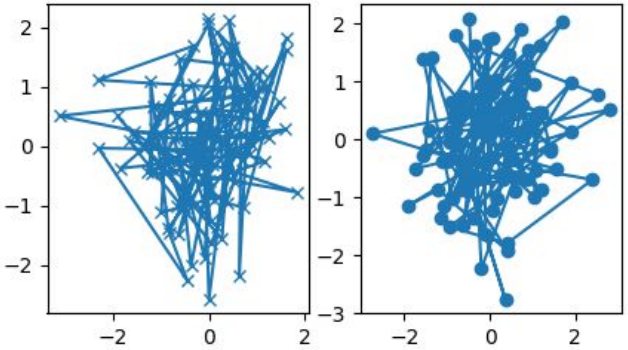


Módulo 2: Unidade 1 – Visualização de dados com a biblioteca matplotlib

Gráficos repetidos com diferentes conjuntos de dados

Se você precisar fazer os mesmos gráficos repetidamente com diferentes conjuntos de dados ou quiser agrupar facilmente os métodos Matplotlib, use a função de assinatura recomendada abaixo.

```
import matplotlib.pyplot as plt
import numpy as np
def my_plotter(ax, data1, data2, param_dict):
    """
    A helper function to make a graph.
    """
    out = ax.plot(data1, data2, **param_dict)
    return out
```




```
data1, data2, data3, data4 = np.random.randn(4, 100)  # make 4 random data sets
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(5, 2.7))
my_plotter(ax1, data1, data2, {'marker': 'x'})
my_plotter(ax2, data3, data4, {'marker': 'o'})
plt.show()
```

Módulo 2: Unidade 1 - Visualização de dados com a biblioteca matplotlib

Estilos e Cores

Estilo

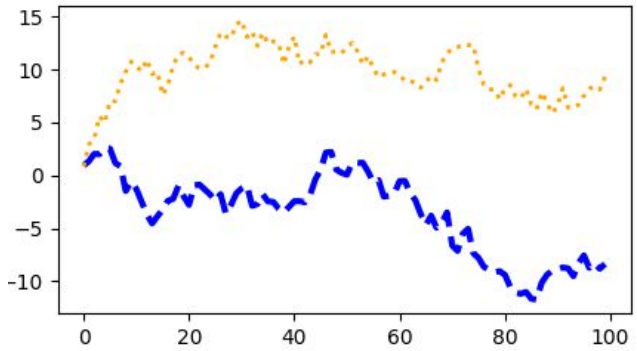
A maioria dos métodos de plotagem tem opções de estilo para os gráricos, acessíveis quando um método de plotagem é chamado ou de um "configurador" no gráfico. No gráfico abaixo, definimos manualmente a cor, a largura da linha e o estilo de linha dos gráficos criados por plot e definimos o estilo de linha da segunda linha após o fato com set_linestyle.

Módulo 2: Unidade 1 – Visualização de dados com a biblioteca matplotlib

Estilos e Cores

Estilo

```
import matplotlib.pyplot as plt
import numpy as np
fig, ax = plt.subplots(figsize=(5, 2.7))
x = np.arange(len(data1))
ax.plot(x, np.cumsum(data1), color='blue', linewidth=3, linestyle='--')
l, = ax.plot(x, np.cumsum(data2), color='orange', linewidth=2)
l.set_linestyle(':')
plt.show()
```



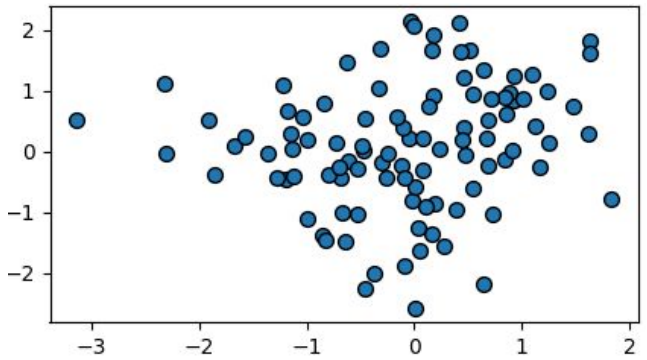
Módulo 2: Unidade 1 – Visualização de dados com a biblioteca matplotlib

Estilos e Cores

Cores

O Matplotlib possui uma variedade muito flexível de cores que são aceitas pela maioria dos gráficos. Para um **scatter** do gráfico, a borda dos marcadores pode ser de cores diferentes do interior:

```
import matplotlib.pyplot as plt
import numpy as np
fig, ax = plt.subplots(figsize=(5, 2.7))
ax.scatter(data1, data2, s=50, facecolor='C0', edgecolor='k')
plt.show()
```



Módulo 2: Unidade 1 - Visualização de dados com a biblioteca matplotlib

Largura e estilos de linhas e tamanhos de marcadores

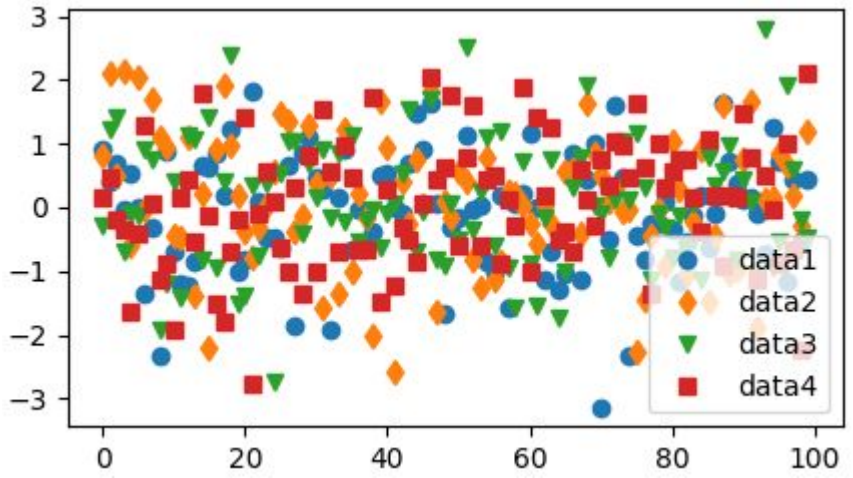
Larguras de linha são tipicamente em pontos tipográficos (1 pt = 1/72 polegadas) e disponíveis para Artistas que possuem linhas traçadas. Da mesma forma, as linhas traçadas podem ter um estilo de linha.

O tamanho do marcador depende do método que está sendo usado. plot especifica o tamanho do marcador em pontos e geralmente é o "diâmetro" ou a largura do marcador. scatter especifica o tamanho do marcador como aproximadamente proporcional à área visual do marcador. Existe uma variedade de estilos de marcadores disponíveis como códigos de string, ou os usuários podem definir seus próprios MarkerStyle.

Módulo 2: Unidade 1 – Visualização de dados com a biblioteca matplotlib

Largura e estilos de linhas e tamanhos de marcadores

```
import matplotlib.pyplot as plt
import numpy as np
fig, ax = plt.subplots(figsize=(5, 2.7))
ax.plot(data1, 'o', label='data1')
ax.plot(data2, 'd', label='data2')
ax.plot(data3, 'v', label='data3')
ax.plot(data4, 's', label='data4')
ax.legend()
plt.show()
```



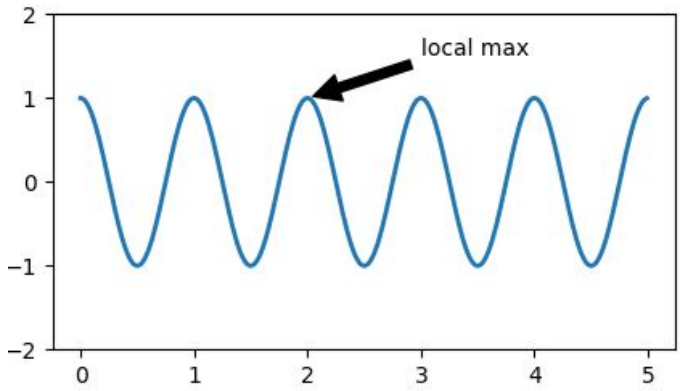
Módulo 2: Unidade 1 – Visualização de dados com a biblioteca matplotlib

Anotações e Legendas

Anotações

Também podemos anotar pontos em um gráfico, geralmente conectando uma seta apontando para xy, com um texto em “xytext”:

```
import matplotlib.pyplot as plt
import numpy as np
fig, ax = plt.subplots(figsize=(5, 2.7))
t = np.arange(0.0, 5.0, 0.01)
s = np.cos(2 * np.pi * t)
line, = ax.plot(t, s, lw=2)
ax.annotate('local max', xy=(2, 1), xytext=(3, 1.5),
            arrowprops=dict(facecolor='black', shrink=0.05))
ax.set_ylim(-2, 2)
plt.show()
```

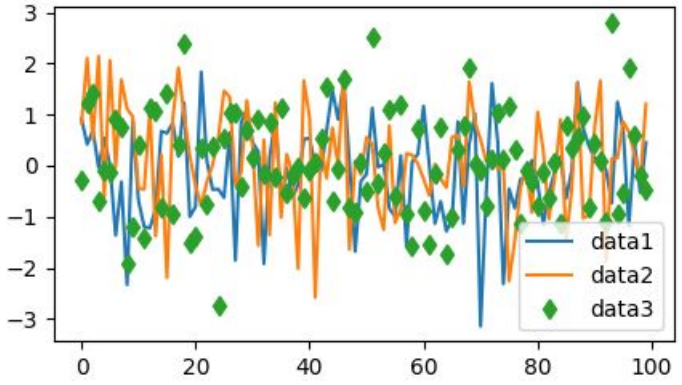


Módulo 2: Unidade 1 – Visualização de dados com a biblioteca matplotlib

Anotações e Legendas

Podemos identificar linhas ou marcadores com Axes.legend:

```
import matplotlib.pyplot as plt
import numpy as np
fig, ax = plt.subplots(figsize=(5, 2.7))
ax.plot(np.arange(len(data1)), data1, label='data1')
ax.plot(np.arange(len(data2)), data2, label='data2')
ax.plot(np.arange(len(data3)), data3, 'd', label='data3')
ax.legend()
plt.show()
```

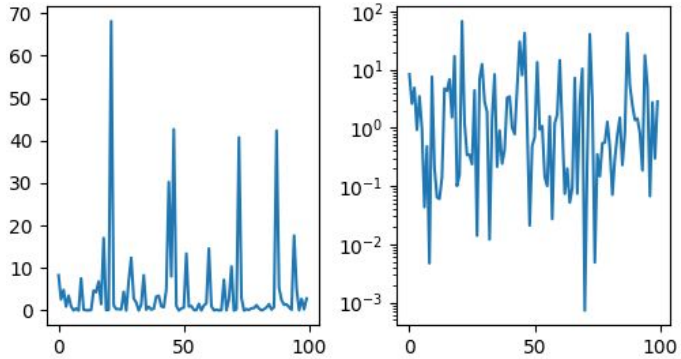


Módulo 2: Unidade 1 – Visualização de dados com a biblioteca matplotlib

Escalas dos eixos

Além da escala linear, o Matplotlib fornece escalas não lineares, como uma escala logarítmica. Como as escalas logarítmicas são muito usadas, também existem métodos diretos como loglog, semilogx e semilogy.

```
import matplotlib.pyplot as plt
import numpy as np
fig, axs = plt.subplots(1, 2, figsize=(5, 2.7))
fig.set_constrained_layout(True)  # Define o layout restrito
xdata = np.arange(len(data1))  # make an ordinal for this
data = 10**data1
axs[0].plot(xdata, data)
axs[1].set_yscale('log')
axs[1].plot(xdata, data)
plt.show()
```

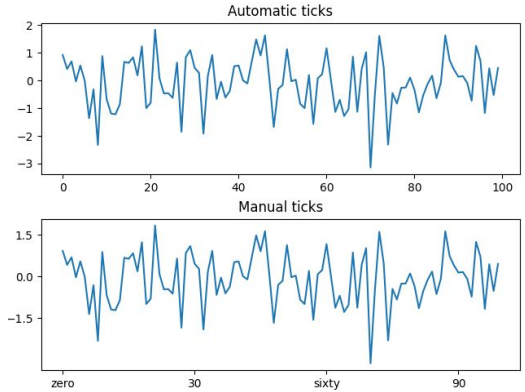


Módulo 2: Unidade 1 – Visualização de dados com a biblioteca matplotlib

Localizadores e formatadores de ticks

O gráfico possui um localizador de marcas e um formatador que escolhe onde colocar as marcas das escalas. Uma interface simples para isso é set_xticks.

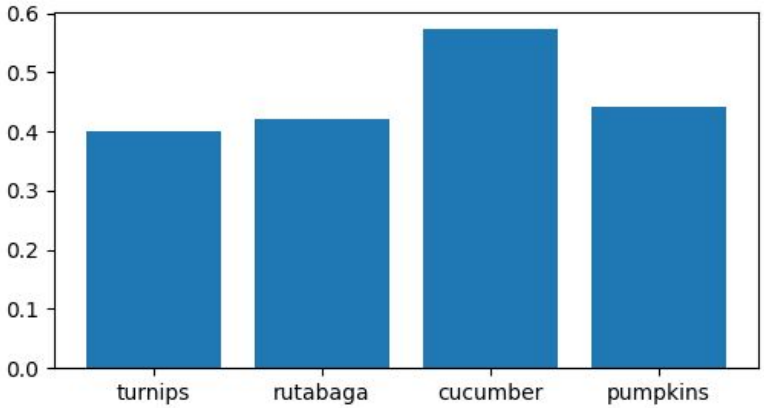
```
import matplotlib.pyplot as plt
import numpy as np
fig, axs = plt.subplots(2, 1)
fig.set_constrained_layout(True)  # Define o layout restrito
xdata = np.arange(len(data1))  # make an ordinal for this
data1 = np.random.randn(100)
axs[0].plot(xdata, data1)
axs[0].set_title('Automatic ticks')
axs[1].plot(xdata, data1)
axs[1].set_xticks(np.arange(0, 100, 30), ['zero', '30', 'sixty', '90'])
axs[1].set_yticks([-1.5, 0, 1.5])
axs[1].set_title('Manual ticks')
plt.show()
```



Módulo 2: Unidade 1 – Visualização de dados com a biblioteca matplotlib

Plotando strings

```
import matplotlib.pyplot as plt
import numpy as np
fig, ax = plt.subplots(figsize=(5, 2.7))
fig.set_constrained_layout(True)  # Define o layout restrito
categories = ['turnips', 'rutabaga', 'cucumber', 'pumpkins']
ax.bar(categories, np.random.rand(len(categories)))
plt.show()
```



Módulo 2: Unidade 1 - Visualização de dados com a biblioteca matplotlib

Trabalhando com várias figuras e eixos

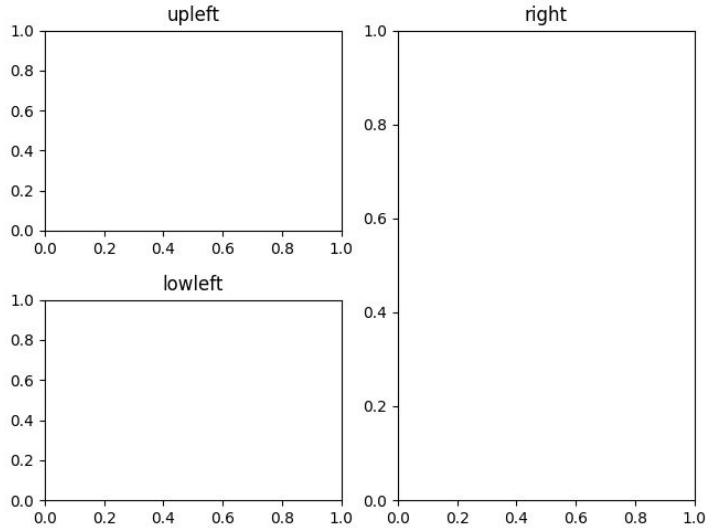
Existe a possibilidade de abrir várias figuras. Ao manter as referências dos objetos, para isso utiliza-se Figura.fig = plt.figure() fig2, ax = plt.subplots().

Vários eixos podem ser adicionados de várias maneiras, a mais básica é plt.subplots(). Pode-se obter layouts mais complexos, com objetos Axes abrangendo colunas ou linhas, usando subplot_mosaic.

Módulo 2: Unidade 1 – Visualização de dados com a biblioteca matplotlib

Trabalhando com várias figuras e eixos

```
fig = plt.figure(constrained_layout=True)  # Define o layout restrito
axd = fig.subplot_mosaic([['upleft', 'right'],
                          ['lowleft', 'right']])
axd['upleft'].set_title('upleft')
axd['lowleft'].set_title('lowleft')
axd['right'].set_title('right')
plt.show()
```



Módulo 2: Unidade 2 - Análise de dados com a biblioteca pandas

A análise de dados envolve tanto habilidades técnicas, como programação, estatísticas e conhecimento de ferramentas de análise, quanto habilidades interpretativas e de comunicação para transformar os resultados em insights valiosos. Com o aumento na disponibilidade de dados e avanços nas tecnologias de análise, a análise de dados tornou-se uma competência fundamental em muitas áreas profissionais.

Módulo 2: Unidade 2 – Análise de dados com a biblioteca pandas

Pandas

Fornece estruturas de dados flexíveis e eficientes para lidar com tabelas e séries temporais, tornando mais fácil o trabalho com conjuntos de dados tabulares.

Recursos e funcionalidades

DataFrame: É uma estrutura tabular bidimensional semelhante a uma planilha do Excel ou uma tabela de banco de dados. Ele é composto por colunas e linhas, permitindo a manipulação e análise de dados de forma intuitiva.

Series: É uma estrutura unidimensional que representa uma sequência de dados, semelhante a uma coluna em um DataFrame ou a um vetor em programação.

Importação e exportação de dados: Permite a importação e exportação de dados de vários formatos, como CSV, Excel, SQL, JSON, HDF5, entre outros.

Módulo 2: Unidade 2 – Análise de dados com a biblioteca pandas

Pandas

Recursos e funcionalidades

Manipulação e transformação de dados: Uma ampla gama de funções para filtrar, ordenar, agrupar, agregar, pivotar, combinar e transformar dados, permitindo a realização de tarefas complexas de limpeza e análise.

Indexação e seleção: Facilita a indexação e seleção de dados, permitindo acessar valores individuais, fatias de dados, colunas específicas e até mesmo executar operações booleanas para filtragem.

Tratamento de valores ausentes: Métodos para lidar com valores ausentes, como preenchimento, remoção ou interpolação.

Visualização de dados: Pandas pode ser usado em conjunto com outras bibliotecas de visualização, como Matplotlib e Seaborn, para criar gráficos e visualizações de dados.

Módulo 2: Unidade 2 – Análise de dados com a biblioteca pandas

Pandas

Recursos e funcionalidades

Integração com numPy: Pandas é construído sobre a biblioteca NumPy e oferece uma integração perfeita com suas estruturas de arrays multidimensionais.

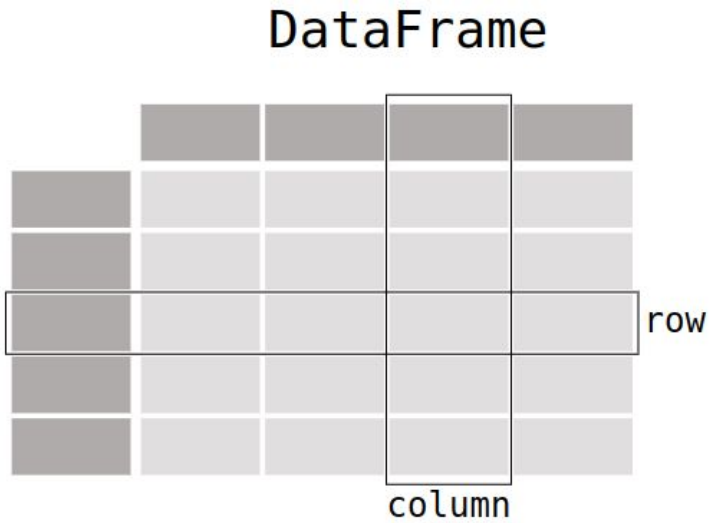
Operações vetorizadas: Pandas suporta operações vetorizadas, o que significa que as operações em colunas inteiras de dados são executadas de forma otimizada, melhorando a eficiência do processamento.

Ferramentas de tempo e séries temporais: Pandas oferece suporte robusto para manipulação de dados de séries temporais, com recursos para resampling, shifting, rolling window, cálculos de média móvel e muito mais.

Agrupamento e agregação: Pandas permite agrupar dados com base em categorias e realizar operações de agregação, como soma, média, contagem, etc., em grupos específicos.

Módulo 2: Unidade 2 – Análise de dados com a biblioteca pandas

Pandas - Iniciando funcionalidades



Módulo 2: Unidade 2 – Análise de dados com a biblioteca pandas

Pandas - Iniciando funcionalidades

Exemplo: Quero armazenar os dados dos passageiros do Titanic. Para um número de passageiros, eu sei os dados de nome (caracteres), idade (números inteiros) e sexo (masculino/feminino).

```
df = pd.DataFrame(
    {
        "Name": [
            "Braund, Mr. Owen Harris",
            "Allen, Mr. William Henry",
            "Bonnell, Miss. Elizabeth",
        ],
        "Age": [22, 35, 58],
        "Sex": ["male", "male", "female"],
    }
)
```

Saída:

	Name	Age	Sex
0	Braund, Mr. Owen Harris	22	male
1	Allen, Mr. William Henry	35	male
2	Bonnell, Miss. Elizabeth	58	female

Módulo 2: Unidade 2 – Análise de dados com a biblioteca pandas

Pandas - Funcionalidades

Exemplo: Como exibir apenas os dados na coluna “Age”?

Ao selecionar uma única coluna de um “DataFrame”, o resultado é uma “Series”.

```
df["Age"]
```

```
Saída:
0    22
1    35
2    58
Name: Age, dtype: int64
```

Exemplo: Como obter a idade máxima das pessoas?

Podemos fazer isso selecionando a coluna “Age” e aplicando “max()”:

```
df["Age"].max()
```

```
Saída:
58
```

Módulo 2: Unidade 2 – Análise de dados com a biblioteca pandas

Pandas - Funcionalidades

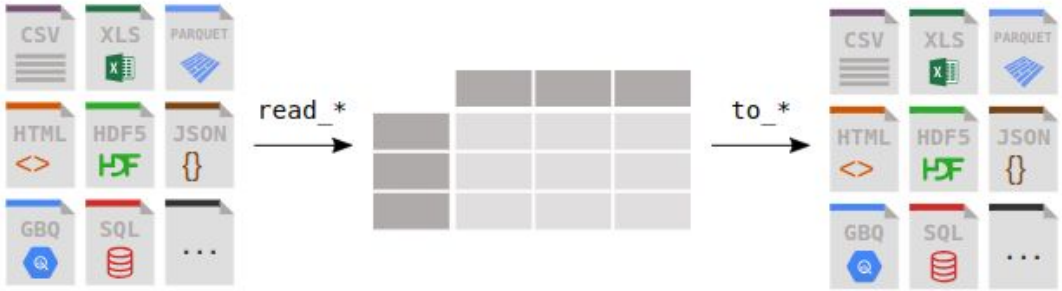
Exemplo: Como existir estatísticas básicas?

```
df.describe()
Saída:
```

	Age
count	3.000000
mean	38.333333
std	18.230012
min	22.000000
25%	28.500000
50%	35.000000
75%	46.500000
max	58.000000

Módulo 2: Unidade 2 – Análise de dados com a biblioteca pandas

Pandas - Leitura e gravação de dados



Módulo 2: Unidade 2 – Análise de dados com a biblioteca pandas

Pandas - Leitura e gravação de dados

Exemplo: Analisando os dados dos passageiros do Titanic, disponíveis como um arquivo CSV.

Pandas fornece a função “read_csv()” para ler dados armazenados como um arquivo csv em “DataFrame”. Pandas suporta muitos formatos de arquivo diferentes ou fontes de dados (csv, excel, sql, json, parquet, ...), cada um com o prefixo “read_”.

Módulo 2: Unidade 2 – Análise de dados com a biblioteca pandas

Pandas - Funcionalidades

Como exibir N primeiras linhas de um “DataFrame”?

Utilizar o método “head()” com o número necessário de linhas (neste caso 8) como argumento.

```
titanic = pd.read_csv("data/titanic.csv")
titanic.head(8)
saída:
```

	PassengerId	Survived	Pclass	...	Fare	Cabin	Embarked
0	1	0	3	...	7.2500	NaN	S
1	2	1	1	...	71.2833	C85	C
2	3	1	3	...	7.9250	NaN	S
3	4	1	1	...	53.1000	C123	S
4	5	0	3	...	8.0500	NaN	S
5	6	0	3	...	8.4583	NaN	Q
6	7	0	1	...	51.8625	E46	S
7	8	0	3	...	21.0750	NaN	S

[8 rows x 12 columns]

Módulo 2: Unidade 2 – Análise de dados com a biblioteca pandas

Pandas - Funcionalidades

Como verificar os tipos de dados ? Pode ser feito por meio do método “dtype”:

```
titanic.dtypes
saída:
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

Módulo 2: Unidade 2 – Análise de dados com a biblioteca pandas

Pandas - Funcionalidades

Como faço para selecionar colunas específicas de um DataFrame?

```
age_sex = titanic[["Age", "Sex"]]
age_sex.head()
saída:
    Age     Sex
0  22.0    male
1  38.0  female
2  26.0  female
3  35.0  female
4  35.0    male
```

Módulo 2: Unidade 2 – Análise de dados com a biblioteca pandas

Pandas - Funcionalidades

Exemplo: Como faço para filtrar linhas específicas de um DataFrame?

A condição dentro dos colchetes verifica em quais linhas a coluna tem um valor maior que 35:titanic["Age"] > 35.

```
titanic["Age"] > 35
saída:
0      False
1       True
2      False
3      False
4      False
      ...
886    False
887    False
888    False
889    False
890    False
Name: Age, Length: 891, dtype: bool
```

Módulo 2: Unidade 2 – Análise de dados com a biblioteca pandas

Pandas - Limpeza e pré-processamento de dados

A limpeza e pré-processamento de dados são etapas essenciais no processo de análise de dados, que envolvem a preparação dos dados brutos para uma análise mais eficaz.

Pandas - Identificando dados duplicados

O método “duplicated()” do pandas pode ser usado para verificar duplicatas em um “DataFrame” ou em colunas específicas. Ele retorna uma série booleana indicando se cada linha é uma duplicata da linha anterior.

```
duplicatas = df.duplicated()  # Verifica duplicatas em todo o DataFrame
coluna_duplicatas = df['coluna'].duplicated()  # Verifica duplicatas em uma coluna específica
```

Módulo 2: Unidade 2 – Análise de dados com a biblioteca pandas

Pandas - Removendo dados duplicados

Algumas abordagens

```
'''
Manter a primeira ocorrência: Isso mantém a primeira ocorrência de cada conjunto de dados duplicados e remove as subsequentes.
'''
df_sem_duplicatas = df.drop_duplicates()
'''
Manter a última ocorrência: Isso mantém a última ocorrência de cada conjunto de dados duplicados e remove as anteriores.
'''
df_sem_duplicatas = df.drop_duplicates(keep='last')
```

Módulo 2: Unidade 2 – Análise de dados com a biblioteca pandas

Pandas - Removendo dados duplicados

Algumas abordagens

```
'''
Remover todas as duplicatas: Isso remove todas as ocorrências de dados duplicados, mantendo apenas a primeira.
'''
df_sem_duplicatas = df.drop_duplicates(keep=False)
'''
Remoção de dados duplicados  com base em colunas específicas
'''
df_sem_duplicatas = df.drop_duplicates(subset=['coluna1', 'coluna2'])
```

IMPORTANTE: Os exemplos citados geram um novo “DataFrame” sem dados duplicados. Se você quiser modificar o “DataFrame” original, pode usar o argumento “inplace=True”:

```
df.drop_duplicates(inplace=True)
```

Módulo 2: Unidade 2 – Análise de dados com a biblioteca pandas

Pandas - Lidar com valores ausentes (NaN) através de preenchimento ou remoção

Identificando valores ausentes

O método “isna()” do pandas pode ser usado para verificar quais valores são ausentes em um “DataFrame” ou em colunas específicas. mó método retorna uma matriz booleana com “True” onde os valores são nulos e “False” onde os valores estão presentes.

```
# Verifica valores ausentes em todo o DataFrame
valores_ausentes = df.isna()

# Verifica valores ausentes em uma coluna específica
coluna_valores_ausentes = df['coluna'].isna()
```

Módulo 2: Unidade 2 – Análise de dados com a biblioteca pandas

Pandas - Lidar com valores ausentes (NaN) através de preenchimento ou remoção

Preenchendo valores ausentes

O método “fillna()” permite preencher os valores ausentes com um valor escolhido.

```
# Preencher valores ausentes em uma coluna com um valor específico

df['coluna'].fillna(valor, inplace=True)
# Preencher valores ausentes em todo o DataFrame
df.fillna(valor, inplace=True)
```

Módulo 2: Unidade 2 – Análise de dados com a biblioteca pandas

Pandas - Lidar com valores ausentes (NaN) através de preenchimento ou remoção

Removendo linhas com valores ausentes

Se os valores ausentes foram encontrados em um número pequeno de linhas, é possível removê-las sem afetar significativamente o conjunto de dados. O método “dropna()” permite remover as linhas com valores ausentes.

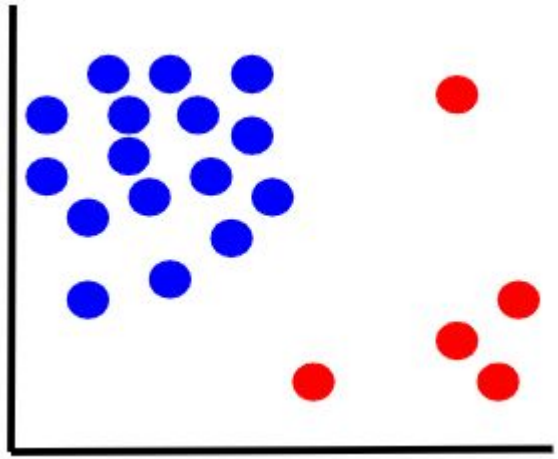
```
# Remover linhas com valores ausentes em qualquer coluna
df.dropna(inplace=True)

# Remover linhas com valores ausentes em colunas específicas
df.dropna(subset=['coluna1', 'coluna2'], inplace=True)
```

Módulo 2: Unidade 2 - Análise de dados com a biblioteca pandas

Pandas - Tratamento de outliers

Outliers (valores extremos ou discrepantes) podem distorcer análises estatísticas e modelagem. Outliers podem ocorrer por diversos motivos, como erros de medição, valores extremos reais ou comportamentos anômalos. O tratamento de outliers visa lidar com esses valores discrepantes de maneira apropriada.



Módulo 2: Unidade 2 – Análise de dados com a biblioteca pandas

Pandas - Tratamento de outliers

```
# Opção 1: Remoção de outliers
df_sem_outliers = df[df['idade'] < limite_superior]

# Opção 2: Transformação
df['idade_transformada'] = df['idade'].apply(lambda x: x ** 0.5)

# Opção 3: Substituição
mediana_idade = df['idade'].median()
df['idade_sem_outliers'] =
df['idade'].apply(lambda x: mediana_idade if x > limite_superior else x)
```

Módulo 2: Unidade 2 – Análise de dados com a biblioteca pandas

Pandas - Conversão de tipos de dados básicos

Você pode usar o método “astype()” para converter as colunas para diferentes tipos de dados básicos, como inteiros, ponto flutuante e string.

```
# Convertendo uma coluna para inteiro
df['coluna_int'] = df['coluna_float'].astype(int)
# Convertendo uma coluna para ponto flutuante
df['coluna_float'] = df['coluna_int'].astype(float)
# Convertendo uma coluna para string
df['coluna_str'] = df['coluna_float'].astype(str)
```

Conversão de tipo durante a leitura

```
df = pd.read_csv('dados.csv', dtype={'coluna_int': int, 'coluna_float': float})
```

Módulo 2: Unidade 2 – Análise de dados com a biblioteca pandas

Pandas - Renomeação de colunas e reindexação de dados

Renomeação de colunas

A renomeação de colunas envolve a alteração dos rótulos (nomes) das colunas em um "DataFrame". Isso pode ser útil para tornar os rótulos mais descritivos, concisos ou uniformes. O pandas fornece o método "rename()" para realizar essa operação.

```
# Renomeando uma coluna
df.rename(columns={'nome_antigo': 'nome_novo'}, inplace=True)
# Renomeando várias colunas
df.rename(columns={'coluna1': 'nova_coluna1', 'coluna2': 'nova_coluna2'}, inplace=True)
```

Módulo 2: Unidade 2 – Análise de dados com a biblioteca pandas

Pandas - Exploração de dados

```
import pandas as pd

# Exemplo de DataFrame
data = {
    'idade': [25, 30, 28, 22, 35, 40, 29, 27, 31, 38],
    'salario': [50000, 60000, 55000, 48000, 75000, 90000, 58000, 52000, 62000, 85000]
}

df = pd.DataFrame(data)
# Média
media_idade = df['idade'].mean()
media_salario = df['salario'].mean()
# Mediana
mediana_idade = df['idade'].median()
mediana_salario = df['salario'].median()
…
```

```
…
# Desvio Padrão
desvio_padrao_idade = df['idade'].std()
desvio_padrao_salario = df['salario'].std()
# Mínimo
minimo_idade = df['idade'].min()
minimo_salario = df['salario'].min()
# Máximo
maximo_idade = df['idade'].max()
maximo_salario = df['salario'].max()
```

Módulo 2: Unidade 2 – Análise de dados com a biblioteca pandas

Pandas - Exploração de dados

Contagem de valores únicos em uma coluna (value_counts())

```
import pandas as pd
# Exemplo de DataFrame
data = {
   'classe': ['A', 'B', 'A', 'C', 'B', 'A', 'C', 'C', 'B', 'A']
}
df = pd.DataFrame(data)
# Contagem de valores únicos em uma coluna
contagem_valores = df['classe'].value_counts()
print(contagem_valores)
saída:
A    4
B    3
C    3
Name: classe, dtype: int64
```

Módulo 2: Unidade 2 – Análise de dados com a biblioteca pandas

Pandas - Exploração de dados

Filtragem e seleção de dados com base em critérios específicos (==, !=, >, <, >= e <=)

```
import pandas as pd
# Exemplo de DataFrame
data = {
   'nome': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],
   'idade': [25, 30, 28, 22, 35],
   'salario': [50000, 60000, 55000, 48000, 75000]
}
df = pd.DataFrame(data)
# Filtragem usando operadores de comparação
idade_maior_30 = df[df['idade'] > 30]
salario_acima_55000 = df[df['salario'] > 55000]
print("Pessoas com idade maior que 30:\n", idade_maior_30)
print("Pessoas com salário acima de 55000:\n", salario_acima_55000)
```

Módulo 2: Unidade 2 – Análise de dados com a biblioteca pandas

Pandas - Exploração de dados

Filtragem e seleção de dados com base em critérios específicos (query, loc, and, or)

query

```
idade_maior_30 = df.query('idade > 30')
salario_acima_55000 = df.query('salario > 55000')
```

loc

O método loc[] permite que você selecione linhas e colunas com base em rótulos ou condições.

```
pessoas_idade_maior_30 = df.loc[df['idade'] > 30, ['nome', 'idade']]
```

and, or

```
filtro = (df['idade'] > 25) & (df['salario'] > 55000)
pessoas_filtradas = df[filtro]
```

Módulo 2: Unidade 2 - Análise de dados com a biblioteca pandas

Pandas - Junção (join) de dados com base em colunas compartilhadas

O método join() permite combinar dataframes usando os índices em vez de colunas-chave.

```
# Fusão usando join
df1 = df1.set_index('chave')
df2 = df2.set_index('chave')
resultado_join = df1.join(df2, lsuffix='_left', rsuffix='_right')
```

Módulo 2: Unidade 2 – Análise de dados com a biblioteca pandas

Pandas - Concatenação de dataframes vertical ou horizontalmente

Concatenação de dataframes vertical ou horizontalmente.

```
import pandas as pd
# Exemplo de DataFrames
df1 = pd.DataFrame({'coluna1': [1, 2, 3]})
df2 = pd.DataFrame({'coluna1': [4, 5, 6]})

# Concatenação vertical
resultado_vertical = pd.concat([df1, df2])

# Concatenação horizontal
resultado_horizontal = pd.concat([df1, df2], axis=1)
```

Módulo 2: Unidade 2 – Análise de dados com a biblioteca pandas

Pandas - Divisão de dados em conjuntos de treinamento e teste

Concatenação de dataframes vertical ou horizontalmente.

```
import pandas as pd
# Exemplo de DataFrames
df1 = pd.DataFrame({'coluna1': [1, 2, 3]})
df2 = pd.DataFrame({'coluna1': [4, 5, 6]})

# Concatenação vertical
resultado_vertical = pd.concat([df1, df2])

# Concatenação horizontal
resultado_horizontal = pd.concat([df1, df2], axis=1)
```

Módulo 2: Unidade 2 – Análise de dados com a biblioteca pandas

Pandas - Divisão de dados em conjuntos de treinamento e teste

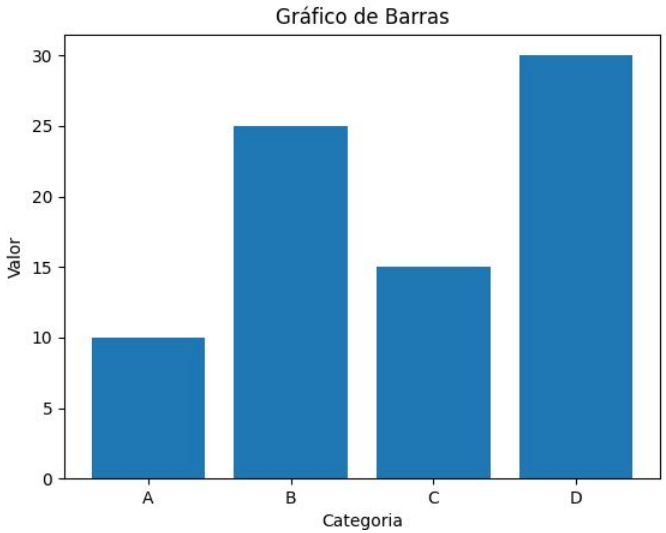
```
import pandas as pd
from sklearn.model_selection import train_test_split
# Exemplo de DataFrame
data = {
    'features': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'target': [0, 0, 0, 0, 1, 1, 1, 1, 1, 1]
}
df = pd.DataFrame(data)
# Divisão dos dados em conjuntos de treinamento e teste
X = df['features']
y = df['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print("Conjunto de treinamento (X):\n", X_train)
print("Conjunto de teste (X):\n", X_test)
print("Conjunto de treinamento (y):\n", y_train)
print("Conjunto de teste (y):\n", y_test)
```

Módulo 2: Unidade 2 – Análise de dados com a biblioteca pandas

Pandas - Visualização de dados (Gráficos de barras, linhas, dispersão, histogramas)

Barras

```
import pandas as pd
import matplotlib.pyplot as plt
data = {'categoria': ['A', 'B', 'C', 'D'],
        'valor': [10, 25, 15, 30]}
df = pd.DataFrame(data)
# Gráfico de barras usando Matplotlib
plt.bar(df['categoria'], df['valor'])
plt.xlabel('Categoria')
plt.ylabel('Valor')
plt.title('Gráfico de Barras')
plt.show()
```

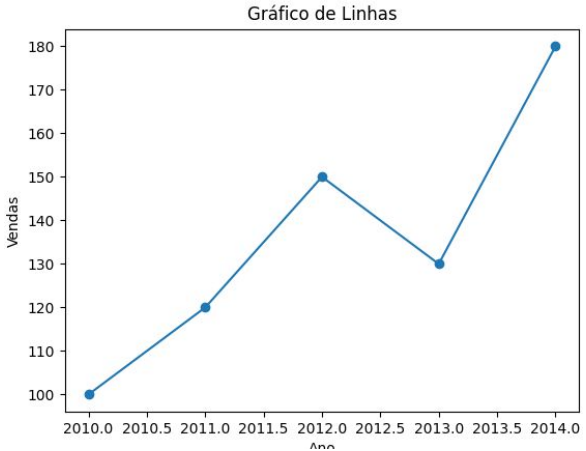


Módulo 2: Unidade 2 – Análise de dados com a biblioteca pandas

Pandas - Visualização de dados (Gráficos de barras, linhas, dispersão, histogramas)

Linhas

```
import pandas as pd
import matplotlib.pyplot as plt
data = {'ano': [2010, 2011, 2012, 2013, 2014],
        'vendas': [100, 120, 150, 130, 180]}
df = pd.DataFrame(data)
# Gráfico de linhas usando Matplotlib
plt.plot(df['ano'], df['vendas'], marker='o')
plt.xlabel('Ano')
plt.ylabel('Vendas')
plt.title('Gráfico de Linhas')
plt.show()
```

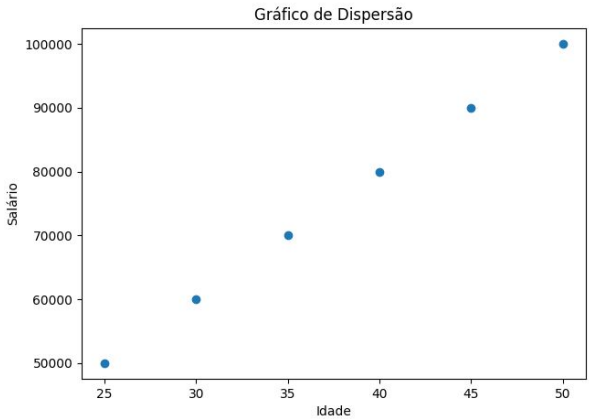


Módulo 2: Unidade 2 – Análise de dados com a biblioteca pandas

Pandas - Visualização de dados (Gráficos de barras, linhas, dispersão, histogramas)

Dispersão

```
import pandas as pd
import matplotlib.pyplot as plt
data = {'idade': [25, 30, 35, 40, 45, 50],
        'salario': [50000, 60000, 70000, 80000, 90000, 100000]}
df = pd.DataFrame(data)
# Gráfico de dispersão usando Matplotlib
plt.scatter(df['idade'], df['salario'])
plt.xlabel('Idade')
plt.ylabel('Salário')
plt.title('Gráfico de Dispersão')
plt.show()
```

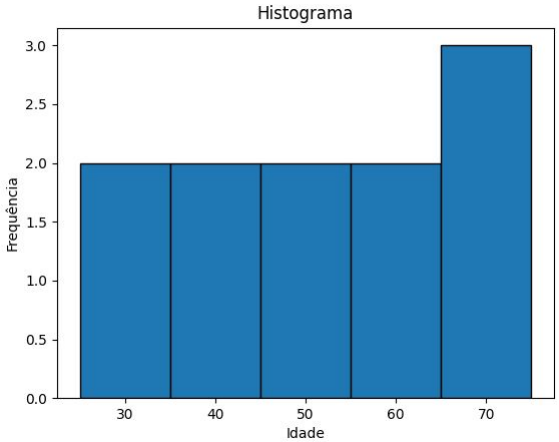


Módulo 2: Unidade 2 – Análise de dados com a biblioteca pandas

Pandas - Visualização de dados (Gráficos de barra, linha, dispersão, histograma)

Histograma

```
import pandas as pd
import matplotlib.pyplot as plt
data = {'idade': [25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75]}
df = pd.DataFrame(data)
# Histograma usando Matplotlib
plt.hist(df['idade'], bins=5, edgecolor='k')
plt.xlabel('Idade')
plt.ylabel('Frequência')
plt.title('Histograma')
plt.show()
```



Atlântico Avanti

(85) 99115-1117

www.instagram.com/avanti.ia/

www.linkedin.com/company/avantiatlantico

www.atlanticoavanti.com.br