FT 16
Curso: UFCD 10793
UFCD/Módulo/Temática: UFCD 10793 **- Fundamentos de Python**
Ação: 10793_05/N
Formador/a: Sandra Liliana Meira de Oliveira
Data:
Nome do Formando/a:

O Python oferece várias bibliotecas para visualização de dados:

1. **Matplotlib**: oferece uma grande variedade de gráficos 2D e 3D. É amplamente utilizada para criar gráficos de linhas, gráficos de dispersão, gráficos de barras e histogramas.
2. **Seaborn**: Seaborn é uma biblioteca de visualização de dados baseada em Matplotlib. Oferece um interface mais apelativo para criar gráficos estatísticos mais atraentes e informativos.
3. **Plotly**: é uma biblioteca interativa de visualização de dados que permite aos utilizadores criar gráficos e painéis interativos. Pode ser utilizada para criar gráficos 2D e 3D, gráficos de dispersão, gráficos de linhas, mapas de calor e muito mais.
4. **Bokeh**: é uma biblioteca Python para criar visualizações interativas para browsers. Oferece uma forma flexível e poderosa de criar visualizações dinâmicas com fluxos de dados, grandes conjuntos de dados e atualizações em tempo real.
5. **ggplot**: é uma implementação Python do popular pacote *R ggplot2*, que fornece uma gramática de gráficos para criar visualizações de dados. É projetada para tornar fácil a criação de visualizações esteticamente atraentes e informativas.
6. **Altair**: é uma biblioteca de visualização estatística declarativa. Permite aos utilizadores criar, facilmente. visualizações interativas usando uma sintaxe concisa e intuitiva.

Matplotlib e Seaborn são duas das bibliotecas de visualização de dados mais comumente usadas em Python. Fornecem uma ampla gama de funcionalidades para criar uma variedade de gráficos 2D e 3D. Matplotlib é uma biblioteca de baixo nível que fornece muita flexibilidade e controlo sobre os detalhes do gráfico, enquanto Seaborn é uma biblioteca de nível superior que torna mais fácil criar gráficos estatísticos atraentes e informativos.

Além de Matplotlib e Seaborn, Plotly e Bokeh também são bibliotecas populares para criar visualizações interativas. Plotly é particularmente adequado para criar gráficos e painéis interativos, enquanto Bokeh é projetado para criar visualizações interativas para navegadores da web modernos.

A escolha da biblioteca depende das necessidades específicas do projeto e dos tipos de visualizações necessárias. Algumas bibliotecas podem ser mais adequadas para determinados tipos de dados ou visualizações, e algumas podem ser mais fáceis de usar do que outras, dependendo do nível de experiência do utilizador com Python e visualização de dados.

MATPLOTLIB

Algumas das funções mais comuns incluem:

- plot(): cria um gráfico de linha, de dispersão ou de barras.
- scatter(): cria um gráfico de dispersão.
- bar(): cria um gráfico de barras verticais ou horizontais.
- hist(): cria um histograma.
- pie(): cria um gráfico de pizza.
- boxplot(): cria um diagrama de caixa.
- imshow(): mostra uma imagem.
- contour(): cria um gráfico de contorno.
- subplots(): cria subplots em uma única figura.
- legend(): adiciona uma legenda ao gráfico.
- xlabel() e ylabel(): adicionam rótulos ao eixo x e y, respectivamente.
- title(): adiciona um título ao gráfico.
- fill_between(): preenche a área entre duas curvas em um gráfico.
- errorbar(): adiciona barras de erro em um gráfico de linha ou de dispersão.
- plot_date(): desenha uma série temporal.

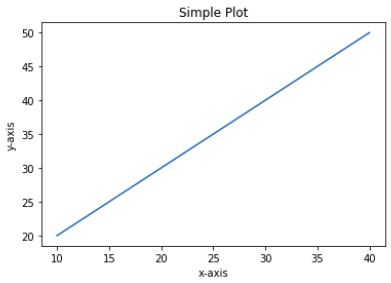
- plot_surface(): desenha um gráfico 3D de superfície.
- plot_wireframe(): desenha um gráfico 3D de arame.
- stem(): desenha um gráfico de hastes verticais.
- barh(): cria um gráfico de barras horizontais.
- violinplot(): cria um diagrama de violino.
- polar(): desenha um gráfico polar.
- streamplot(): desenha um gráfico de fluxo de vetor.
- scatterplot(): cria um gráfico de dispersão com pontos que podem ser coloridos ou marcados de forma diferente com base em uma variável categórica ou numérica.
- hexbin(): cria um gráfico de dispersão hexagonal, útil para visualizar a densidade de dados.
- imshow(): cria um mapa de calor de uma matriz 2D, útil para visualizar dados de imagens ou matrizes.
- contourf(): cria um gráfico de contorno preenchido, útil para visualizar dados em 3D.
- annotate(): adiciona uma anotação a um ponto específico do gráfico.
- arrow(): adiciona uma seta ao gráfico.
- axhline(): adiciona uma linha horizontal a um eixo.
- axhspan(): adiciona uma faixa horizontal a um eixo.
- axvline(): adiciona uma linha vertical a um eixo.
- axvspan(): adiciona uma faixa vertical a um eixo.
- bar(): cria um gráfico de barras verticais ou horizontais.
- barbs(): desenha um gráfico de flechas vetoriais.
- boxplot(): cria um diagrama de caixa.
- broken_barh(): cria um gráfico de barras horizontais com lacunas.
- cla(): limpa o eixo atual.
- clabel(): adiciona rótulos de texto a um gráfico de contorno.
- close(): fecha a janela atual do gráfico.
- cohere(): cria um gráfico de coerência.
- contour(): cria um gráfico de contorno.
- contourf(): cria um gráfico de contorno preenchido.
- csd(): cria um gráfico de densidade espectral cruzada.
- errorbar(): adiciona barras de erro em um gráfico de linha ou de dispersão.
- fill(): preenche a área entre duas curvas em um gráfico.
- fill_between(): preenche a área entre duas curvas em um gráfico.
- fill_betweenx(): preenche a área entre duas curvas em um gráfico horizontal.
- hexbin(): cria um gráfico de dispersão hexagonal.
- hist(): cria um histograma.
- hist2d(): cria um histograma 2D.
- hlines(): desenha várias linhas horizontais.
- imshow(): mostra uma imagem.
- legend(): adiciona uma legenda ao gráfico.
- loglog(): cria um gráfico de escala logarítmica em ambos os eixos.
- matshow(): mostra uma matriz 2D como uma imagem.
- pcolor(): cria um gráfico de cor em um plano 2D.
- pcolormesh(): cria um gráfico de cor em um plano 2D.
- pie(): cria um gráfico de pizza.
- plot(): cria um gráfico de linha, de dispersão ou de barras.
- plot_date(): desenha uma série temporal.
- plotfile(): desenha dados a partir de um arquivo.
- polar(): desenha um gráfico polar.
- psd(): cria um gráfico de densidade espectral.
- quiver(): desenha um gráfico de flechas vetoriais.
- quiverkey(): adiciona uma legenda a um gráfico de flechas vetoriais.
- scatter(): cria um gráfico de dispersão.
- semilogx(): cria um gráfico de escala logarítmica no eixo x.
- semilogy(): cria um gráfico de escala logarítmica no eixo y.
- specgram(): cria um gráfico de espectrograma.
- spy(): cria um gráfico de padrão de matriz.
- stackplot(): cria um gráfico de áreas empilhadas.
- stem(): desenha um gráfico de hastes verticais.
- step(): desenha um gráfico de passos.
- streamplot(): desenha um gráfico de fluxo de vetor.
- subplot(): cria um subplot em uma figura.
- subplot2grid(): cria subplots em uma figura usando um layout de grade.
- subplot_tool(): exibe uma interface interativa para criar subplots.
- suptitle(): adiciona um título à figura.
- table(): adiciona uma tabela ao gráfico.
- text(): adiciona um texto ao gráfico.
- title(): adiciona um título ao gráfico.
- tricontour(): cria um gráfico de contorno a partir de dados triangulados.
- tricontourf(): cria um gráfico de contorno preenchido a partir de dados triangulados.
- triplot(): cria um gráfico de linha a partir de dados triangulados.
- twiny(): cria um segundo eixo x compartilhando o mesmo eixo y.
- violinplot(): cria um diagrama de violino.
- vlines(): plota várias linhas verticais.
- xcorr(): cria um gráfico de correlação cruzada.
- xlabel(): adiciona uma legenda ao eixo x.
- xlim(): define os limites do eixo x.
- xticks(): define os locais e rótulos dos ticks no eixo x.
- ylabel(): adiciona uma legenda ao eixo y.
- ylim(): define os limites do eixo y.
- yticks(): define os locais e rótulos dos ticks no eixo y.

Reproduz os seguintes exemplos usando Matplotlib com Jupyter Notebook e associa até dia 22 de Fevereiro à tarefa indicada

Comecemos por criar um gráfico simples. Vamos traçar duas listas contendo coordenadas X, Y para o gráfico.

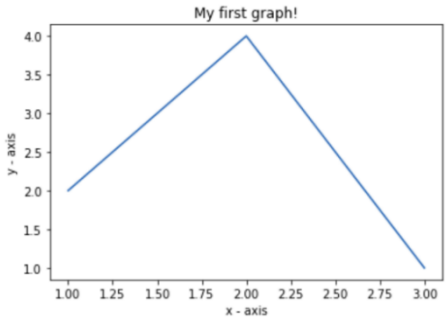
```
import matplotlib.pyplot as plt
# initializing the data
x = [10, 20, 30, 40]
y = [20, 30, 40, 50]
# plotting the data
plt.plot(x, y)
# Adding the title
plt.title("Simple Plot")
# Adding the labels
plt.ylabel("y-axis")
plt.xlabel("x-axis")
plt.show()
```

Saída:




```
import matplotlib.pyplot as plt

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

plt.plot(x, y)

plt.xlabel('x - axis')
plt.ylabel('y - axis')

plt.title('My first graph!')

plt.show()
```

Resultado:



O código parece autoexplicativo. Seguiram-se as seguintes etapas:

- Defina o eixo x e os valores correspondentes do eixo y como listas.
- Desenhe-os usando a função .plot().
- Dê um nome aos eixos x ey usando as funções .xlabel() e .ylabel().
- Dê um título ao seu gráfico usando a função .title().
- Finalmete, para visualizar seu gráfico, usamos a função .show().

Pyplot é um módulo Matplotlib que fornece uma interface semelhante ao MATLAB. Pyplot fornece funções que interagem com a figura, ou seja, cria uma figura, decora o gráfico com rótulos, cria uma área de desenho numa figura.

Sintaxe:

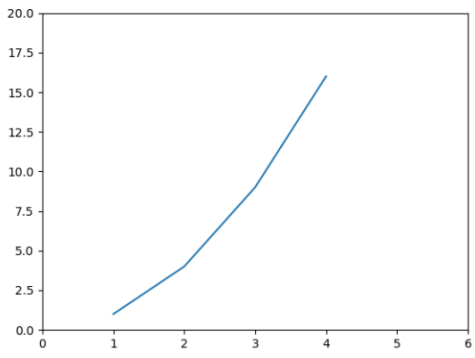
matplotlib.pyplot.plot (* args, scalex = True, scaley = True, data = None, ** kwargs)

No seguinte exemplo, a função de *plot* marca as coordenadas x (1, 2, 3, 4) e as coordenadas y (1, 4, 9, 16) num gráfico linear com escalas especificadas.

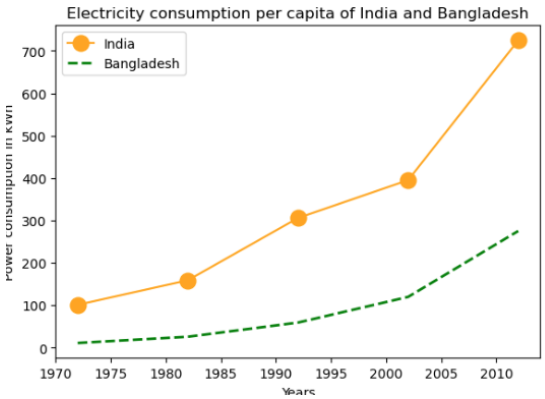
Parâmetros: esta função aceita parâmetros que nos permitem definir escalas de eixos e formatar os gráficos. Esses parâmetros são mencionados abaixo:

- **plot (x, y):** desenha x e y usando o estilo e cor de linha padrão.
- **plot.axis ([xmin, xmax, ymin, ymax])** : dimensiona os eixos x e y do valor mínimo para o máximo
- **plot. (x, y, cor = 'verde', marcador = 'o', estilo de linha = 'tracejado', largura da linha = 2, tamanho do marcador = 12): as** coordenadas x e y são marcadas com marcadores circulares de tamanho 12 e verde linha de cor com - estilo de largura 2
- **plot.xlabel ('eixo X')** : nome eixo x
- **plot.ylabel ('eixo Y')** : nome eixo y
- **plot (x, y, rótulo = 'Linha de amostra')** A linha de amostra plotada será exibida como uma legenda

```
# Python program to show pyplot module
   import matplotlib.pyplot as plt
   plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
   plt.axis([0, 6, 0, 20])
   plt.show()
```



```
import matplotlib.pyplot as plt

year = [1972, 1982, 1992, 2002, 2012]
e_india = [100.6, 158.61, 305.54, 394.96, 724.79]
e_bangladesh = [10.5, 25.21, 58.65, 119.27, 274.87]
#with different colored labels of two countries

plt.plot(year, e_india, color ='orange',
         label ='India')

plt.plot(year, e_bangladesh, color ='g',
         label ='Bangladesh')
plt.xlabel('Years')
plt.ylabel('Power consumption in kWh')
plt.title('Electricity consumption per capita\
 of India and Bangladesh')

plt.legend()
plt.show()
```



Função title

O `title()` método no módulo matplotlib é usado para especificar o título da visualização representada e exibe o título usando vários atributos.

Sintaxe: matplotlib.pyplot.title (label, fontdict = None, loc = 'center', pad = None, ** kwargs)

Parâmetros:

- **`label`**(str): Este argumento se refere à sequência de texto do título real da visualização representada.
- **`fontdict`**(dict): Este argumento controla a aparência do texto, como tamanho do texto, alinhamento do texto etc. usando um dicionário. Abaixo está o fontdict padrão:

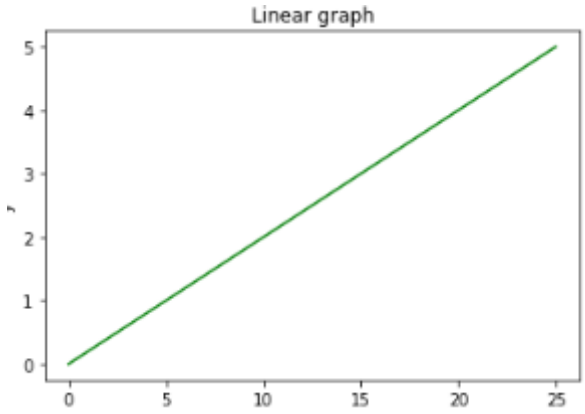
  fontdict = {'fontsize': rcParams ['axes.titlesize'],
  'fontweight': rcParams ['axes.titleweight'],
  'verticalalignment': 'baseline',
  'horizontalalignment': loc}

- **`loc`**(str): Este argumento se refere à localização do título, recebe valores de string como `'center'`, `'left'` e `'right'`.
- **`pad`**(float): Este argumento se refere ao deslocamento do título do topo dos eixos, em pontos. Seus valores padrão são Nenhum.
- **** kwargs:** Este argumento refere-se ao uso de outros argumentos nomeados conforme as propriedades de texto, como `color`, `fonstyle`, `linespacing`, `backgroundcolor`, `rotation`etc.

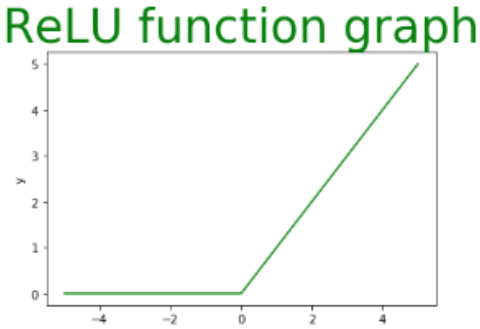
Tipo de retorno: O `title()` método retorna uma string que representa o próprio texto do título.

Abaixo estão alguns exemplos para ilustrar o uso do `title()` método:

```
import matplotlib.pyplot as plt


y = [0,1,2,3,4,5]
x= [0,5,10,15,20,25]
plt.plot(x, y, color='green')
plt.xlabel('x')
plt.ylabel('y')
plt.title("Linear graph")

plt.show()
```

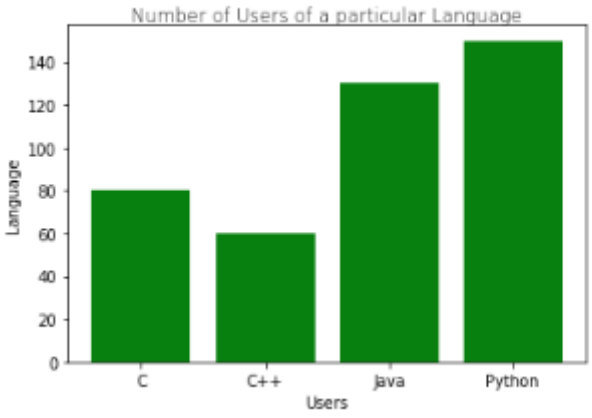



```
import matplotlib.pyplot as plt


x = [-5,-4,-3,-2,-1,0,1,2, 3, 4, 5]
y = []

for i in range(len(x)):
    y.append(max(0,x[i]))
plt.plot(x, y, color='green')
plt.xlabel('x')
plt.ylabel('y')
plt.title(label="ReLU function graph",
          fontsize=40,
          color="green")
```



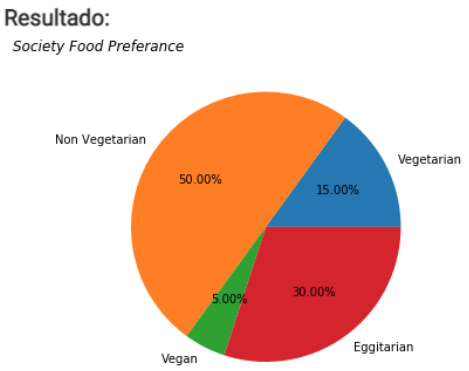

```
import matplotlib.pyplot as plt
import numpy as np
language = ['C','C++','Java','Python']
users = [80,60,130,150]
index = np.arange(len(language))
plt.bar(index, users, color='green')
plt.xlabel('Users')
plt.ylabel('Language')
plt.xticks(index, language)
plt.title(label='Number of Users of a particular Language',
          fontweight=10,
          pad='2.0')
```

Resultado:




```
from matplotlib import pyplot as plt
foodPreferance = ['Vegetarian', 'Non Vegetarian',
                  'Vegan', 'Eggitarian']

consumers = [30,100,10,60]
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.axis('equal')
ax.pie(consumers, labels = foodPreferance,
       autopct='%1.2f%%')
plt.title(label="Society Food Preferance",
          loc="left",
          fontstyle='italic')
```

Resultado:



Função matplotlib.pyplot.imshow():

A **função imshow()** no módulo pyplot da biblioteca matplotlib é usada para exibir dados como uma imagem; ou seja, em um raster regular 2D.

Parâmetros: este método aceita os seguintes parâmetros descritos abaixo:

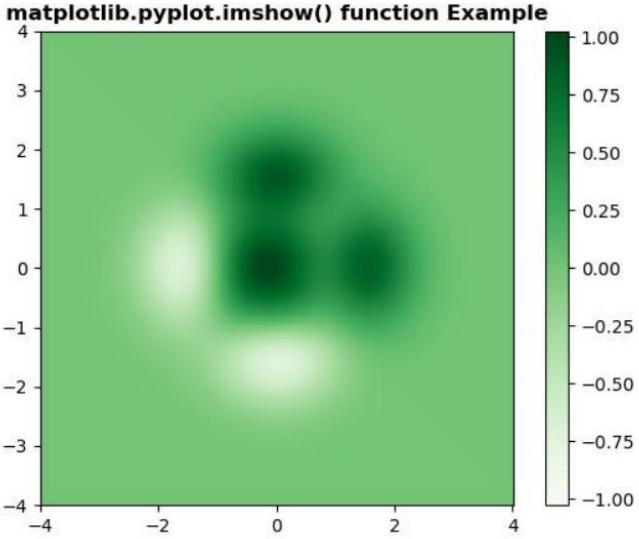
- **X:** este parâmetro são os dados da imagem.
- **cmap:** Este parâmetro é uma instância de mapa de cores ou nome de mapa de cores registrado.
- **norma:** Este parâmetro é a instância Normalizar que escala os valores dos dados para o intervalo do mapa de cores canônico [0, 1] para mapeamento de cores
- **vmin, vmax:** Esses parâmetros são opcionais por natureza e são uma faixa de barras de cores.
- **alfa:** este parâmetro é a intensidade da cor.
- **aspecto:** Este parâmetro é usado para controlar a relação de aspecto dos eixos.
- **interpolação:** Este parâmetro é o método de interpolação usado para exibir uma imagem.
- **origin:** Este parâmetro é usado para colocar o índice [0, 0] da matriz no canto superior esquerdo ou inferior esquerdo dos eixos.
- **resample:** Este parâmetro é o método que é usado para a semelhança.
- **extensão:** este parâmetro é a caixa delimitadora nas coordenadas de dados.
- **filternorm:** Este parâmetro é usado para o filtro de redimensionamento de imagem antigrain.
- **filterrad:** este parâmetro é o raio do filtro para filtros que possuem um parâmetro de raio.
- **url:** Este parâmetro define a url do **AxesImage** criado **.**

- **imagem:** retorna o **AxesImage**

Os exemplos abaixo ilustram a função matplotlib.pyplot.imshow() em matplotlib.pyplot:

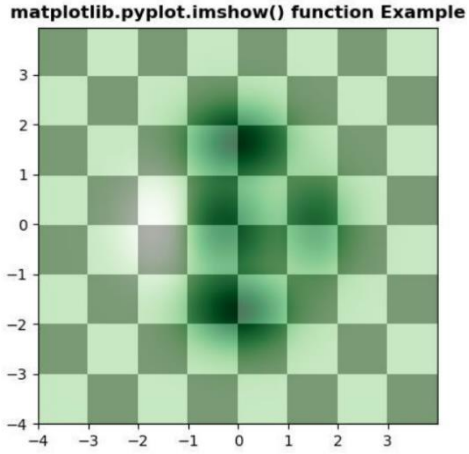
```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.colors import LogNorm

dx, dy = 0.015, 0.05
y, x = np.mgrid[slice(-4, 4 + dy, dy),
                slice(-4, 4 + dx, dx)]
```

```
z = (1 - x / 3. + x ** 5 + y ** 5) * np.exp(-x ** 2 - y ** 2)
z = z[:-1, :-1]
z_min, z_max = -np.abs(z).max(), np.abs(z).max()

c = plt.imshow(z, cmap ='Greens', vmin = z_min, vmax = z_max,
                          extent =[x.min(), x.max(), y.min(), y.max()],
                              interpolation ='nearest', origin ='lower')
plt.colorbar(c)

plt.title('matplotlib.pyplot.imshow() function Example',
                                                    fontweight ="bold")
plt.show()
```




```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.colors import LogNorm

dx, dy = 0.015, 0.05
x = np.arange(-4.0, 4.0, dx)
y = np.arange(-4.0, 4.0, dy)
X, Y = np.meshgrid(x, y)

extent = np.min(x), np.max(x), np.min(y), np.max(y)

Z1 = np.add.outer(range(8), range(8)) % 2
plt.imshow(Z1, cmap ="binary_r", interpolation ='nearest',
                                          extent = extent, alpha = 1)

def geeks(x, y):
    return (1 - x / 2 + x**5 + y**6) * np.exp(-(x**2 + y**2))

Z2 = geeks(X, Y)

plt.imshow(Z2, cmap ="Greens", alpha = 0.7,
```

```
                interpolation ='bilinear', extent =  extent)

plt.title('matplotlib.pyplot.imshow() function Example',
                                                    fontweight ="bold")
plt.show()
```



Matplotlib.pyplot.legend()

Uma legenda é uma área que descreve os elementos do gráfico. Na biblioteca matplotlib, há uma função chamada **legend()** que é usada para colocar uma legenda nos eixos.

O atributo **Loc** in legend()é usado para especificar a localização da legenda. O valor padrão de loc é loc = ”best” (canto superior esquerdo). As strings 'superior esquerdo', 'superior direito', 'inferior esquerdo', 'inferior direito' colocam a legenda no canto correspondente dos eixos / figura.

O atributo **bbox_to_anchor = (x, y)** da função legend() é usado para especificar as coordenadas da legenda, e o atributo **ncol** representa o número de colunas que a legenda possui. Seu valor padrão é 1.

Sintaxe: matplotlib.pyplot.legend ([“azul”, “verde”], bbox_to_anchor = (0,75, 1,15), ncol = 2)

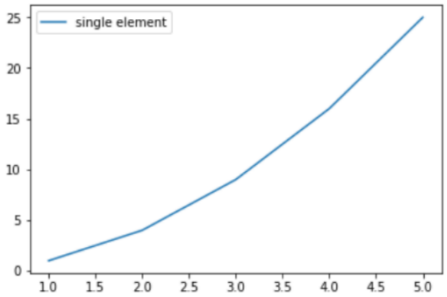
A seguir estão mais alguns atributos de função `legend()`:

- **sombra** : [Nenhum ou bool] Se deve desenhar uma sombra atrás da legenda. O valor padrão é Nenhum.
- **markerscale** : [None ou int ou float] O tamanho relativo dos marcadores de legenda em comparação com os desenhados originalmente. O padrão é Nenhum.
- **numpoints** : [Nenhum ou int] O número de pontos do marcador na legenda ao criar uma entrada de legenda para um Line2D (linha). O padrão é Nenhum.
- **fontsize** : O tamanho da fonte da legenda. Se o valor for numérico, o tamanho será o tamanho absoluto da fonte em pontos.
- **facecolor** : [Nenhum ou “herdar” ou cor] A cor de fundo da legenda.
- **edgecolor** : [Nenhum ou “herdar” ou cor] A cor da borda do patch de fundo da legenda.

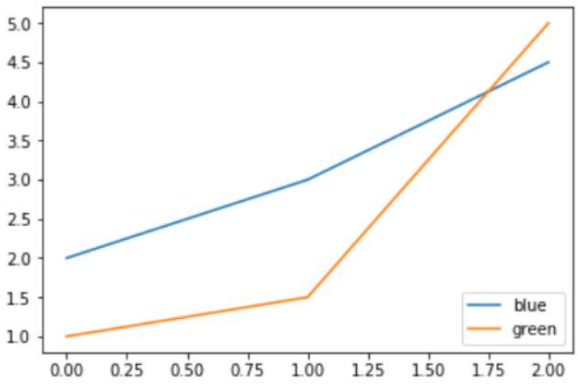
Maneiras de usar a função legend() em Python:

```
import  numpy as np
```

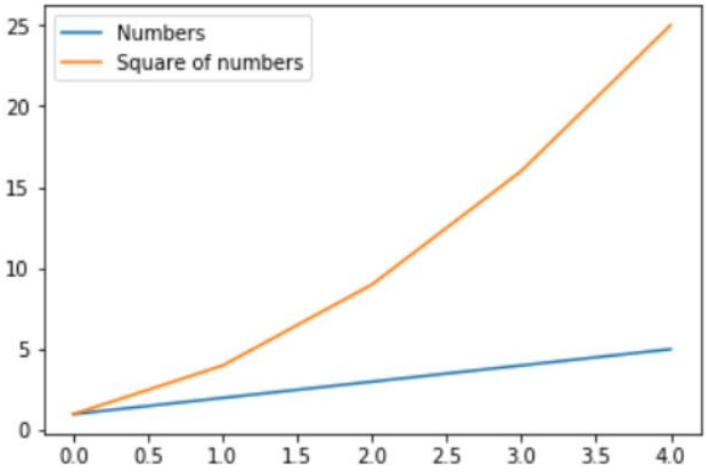
```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]
plt.plot(x, y)
plt.legend(['single element'])
plt.show()
```



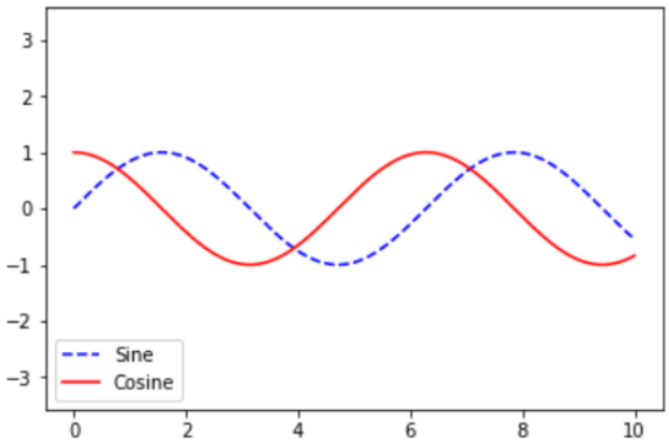

```
import numpy as np
import matplotlib.pyplot as plt
y1 = [2, 3, 4.5]
y2 = [1, 1.5, 5]
plt.plot(y1)
plt.plot(y2)
plt.legend(["blue", "green"], loc ="lower right")
plt.show()
```




```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(5)
y1 = [1, 2, 3, 4, 5]
y2 = [1, 4, 9, 16, 25]
plt.plot(x, y1, label ='Numbers')
plt.plot(x, y2, label ='Square of numbers')
plt.legend()
plt.show()
```

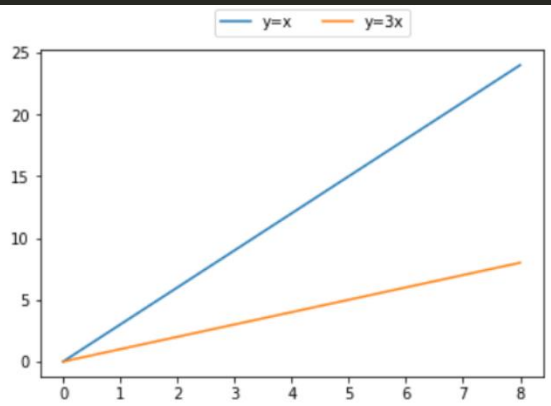



```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 10, 1000)
fig, ax = plt.subplots()

ax.plot(x, np.sin(x), '--b', label ='Sine')
ax.plot(x, np.cos(x), c ='r', label ='Cosine')
ax.axis('equal')

leg = ax.legend(loc ="lower left");
```




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

x = [0, 1, 2, 3, 4, 5, 6, 7, 8]

y1 = [0, 3, 6, 9, 12, 15, 18, 21, 24]
y2 = [0, 1, 2, 3, 4, 5, 6, 7, 8]

plt.plot(y1, label ="y = x")
plt.plot(y2, label ="y = 3x")
```

```
plt.legend(bbox_to_anchor =(0.75, 1.15), ncol =  2)

plt.show()
```



Função matplotlib.pyplot.subplots()

A **função** subplots **()** no módulo pyplot da biblioteca matplotlib é usada para criar uma figura e um conjunto de subplots.

Sintaxe: matplotlib.pyplot.subplots (nrows = 1, ncols = 1, sharex = False, sharey = False, squeeze = True, subplot_kw = None, gridspec_kw = None, ** fig_kw)

Parâmetros: este método aceita os seguintes parâmetros descritos abaixo:

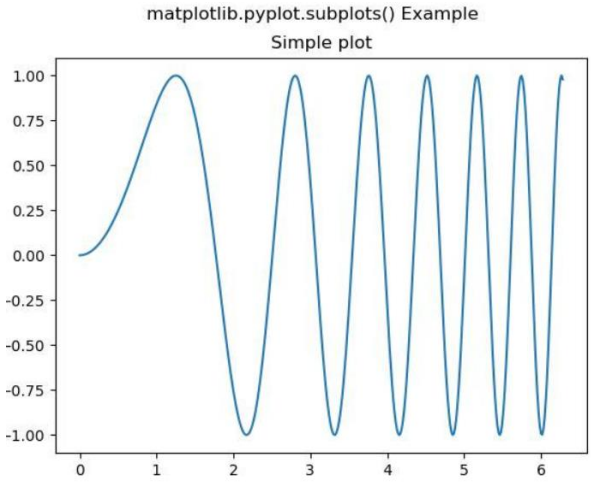
- **nrows, ncols:** Estes parâmetros são o número de linhas / colunas da grade do subplot.
- **sharex, sharey:** Este parâmetro controla o compartilhamento de propriedades entre os eixos x (sharex) ou y (sharey).
- **squeeze:** este parâmetro é um parâmetro opcional e contém o valor booleano com o padrão True.
- **num:** este parâmetro é a palavra-chave pyplot.figure que define o número da figura ou rótulo.
- **subplot_kwd:** Este parâmetro é o dict com palavras-chave passadas para a chamada add_subplot usada para criar cada subplot.
- **gridspec_kw:** Este parâmetro é o dict com palavras-chave passadas para o construtor GridSpec usado para criar a grade em que os subplots são colocados.

Retorna: Este método retorna os seguintes valores.

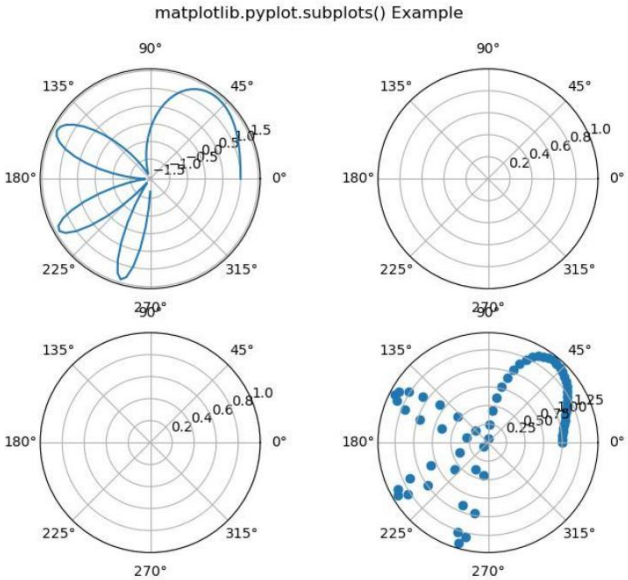
- **fig:** este método retorna o layout da figura.
- axes **:** Este método retorna o objeto axes.Axes ou matriz de objetos Axes.

```
import  numpy as np
import  matplotlib.pyplot as plt
x =  np.linspace(0,  2  *  np.pi,  400)
y =  np.sin(x**2)

fig,  ax =  plt.subplots()
ax.plot(x,  y)
ax.set_title('Simple plot')
```

```
fig.suptitle('matplotlib.pyplot.subplots() Example')
plt.show()
```




```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 1.5 * np.pi, 100)
y = np.sin(x**2)+np.cos(x**2)

fig, axs = plt.subplots(2, 2,
                                    subplot_kw = dict(polar = True))
axs[0, 0].plot(x, y)
axs[1, 1].scatter(x, y)

fig.suptitle('matplotlib.pyplot.subplots() Example')
plt.show()
```



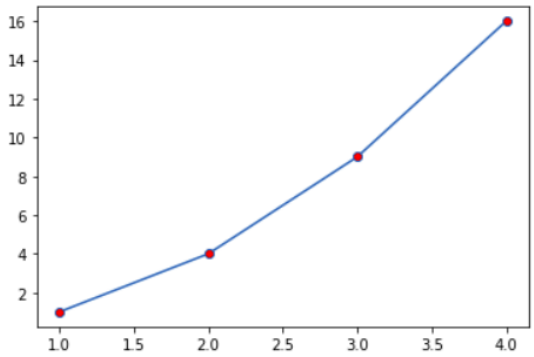
Matplotlib.pyplot.colors()

Esta função é usada para especificar a cor.

Podemos usar essa função para várias visualizações de dados e obter insights deles.

```
import matplotlib.pyplot as plt


color = 'green'
plt.plot([1, 2, 3, 4], color = color)

plt.show()
```

Resultado:



Função matplotlib.pyplot.grid()

A **função grid()** no módulo pyplot da biblioteca matplotlib é usada para configurar as linhas de grade.

Sintaxe: matplotlib.pyplot.grid (b = None, which = 'major', axis = 'both', \ * \ * kwargs)

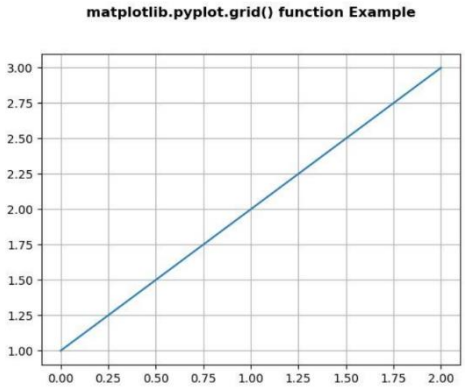
Parâmetros: este método aceita os seguintes parâmetros.

- **b:** Este parâmetro é um parâmetro opcional, se deve mostrar as linhas de grade ou não.
- **qual:** Este parâmetro também é um parâmetro opcional e são as linhas de grade nas quais aplicar as alterações.
- **eixo:** este parâmetro também é um parâmetro opcional e é o eixo para aplicar as alterações.

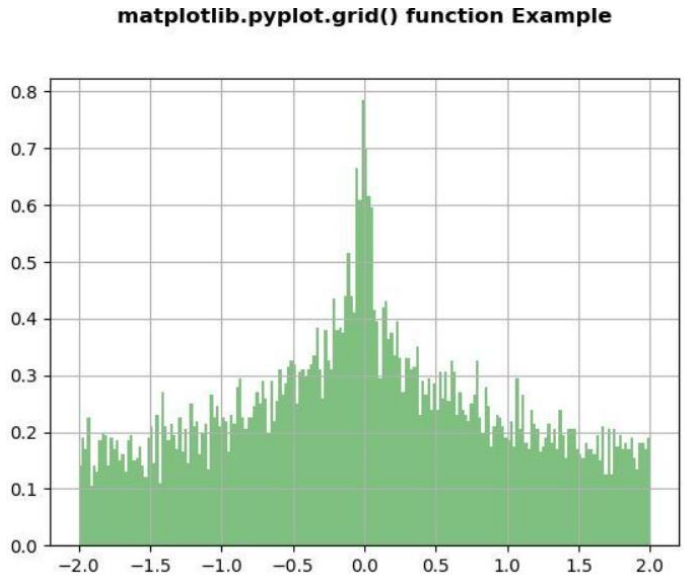
Retorna: Este método não retorna nenhum valor.

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

```
plt.plot([1, 2, 3])
plt.grid()

plt.title('matplotlib.pyplot.grid() function \
Example\n\n', fontweight ="bold")
plt.show()
```




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

np.random.seed(19680801)

val, res = 100, 15
x = np.sin(val + res * np.random.randn(10000)) - np.cos(val + res *
np.random.randn(10000))

n, bins, patches = plt.hist(x, 200,density = True, facecolor ='g',
                                        alpha = 0.5)

plt.grid(True)

plt.title('matplotlib.pyplot.grid() function \
Example\n\n', fontweight ="bold")

plt.show()
```



A classe Figure é o ***container*** ("recipiente") de nível superior que contém um ou mais eixos. É a janela ou página geral, na qual tudo é desenhado.

Sintaxe:

class matplotlib.figure.Figure (figsize = None, dpi = None, facecolor = None, edgecolor = None, linewidth = 0.0, frameon = None, subplotpars = None, tight_layout = None, constrained_layout = None)

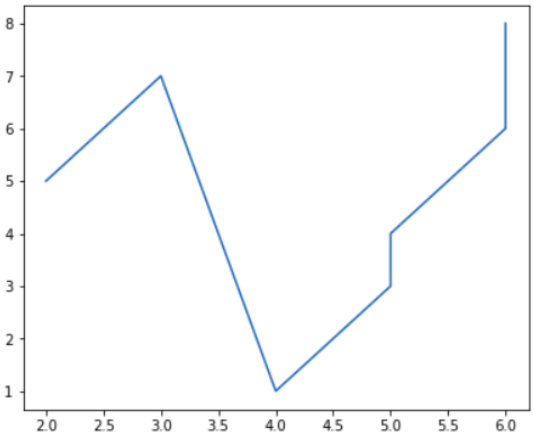
Parâmetros: aceita os seguintes parâmetros descritos abaixo:

- **figsize:** Este parâmetro é a dimensão da Figura (largura, altura) em polegadas.
- **dpi:** Este parâmetro é o número de pontos por polegada.
- **facecolor:** Este parâmetro é a cor da face do patch da figura.
- **edgecolor:** Este parâmetro é a cor da borda do patch da figura.
- **largura de linha:** este parâmetro é a largura de linha do quadro.
- **frameon:** Este parâmetro é a supressão do desenho do patch de fundo da figura.
- **subplotpars:** Este parâmetro são os parâmetros do Subplot.
- **tight_layout:** este parâmetro é usado para ajustar os parâmetros do subplot.
- **constrained_layout:** este parâmetro é usado para ajustar o posicionamento dos elementos do gráfico.

Retorna: Este método retorna as instâncias da figura.

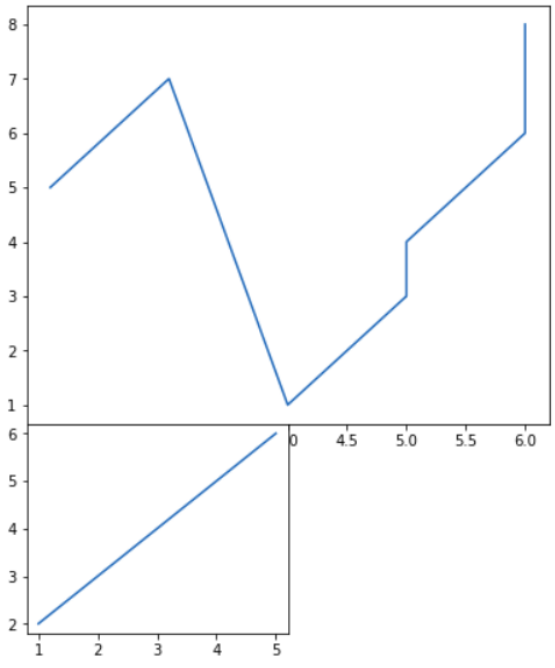
```
# Python program to show pyplot module
   import matplotlib.pyplot as plt
   from matplotlib.figure import Figure
   # Creating a new figure with width = 5 inches
   # and height = 4 inches
   fig = plt.figure(figsize =(5, 4))
   # Creating a new axes for the figure
   ax = fig.add_axes([1, 1, 1, 1])
   # Adding the data to be plotted
   ax.plot([2, 3, 4, 5, 5, 6, 6], [5, 7, 1, 3, 4, 6 ,8])
```

```
    plt.show()
```

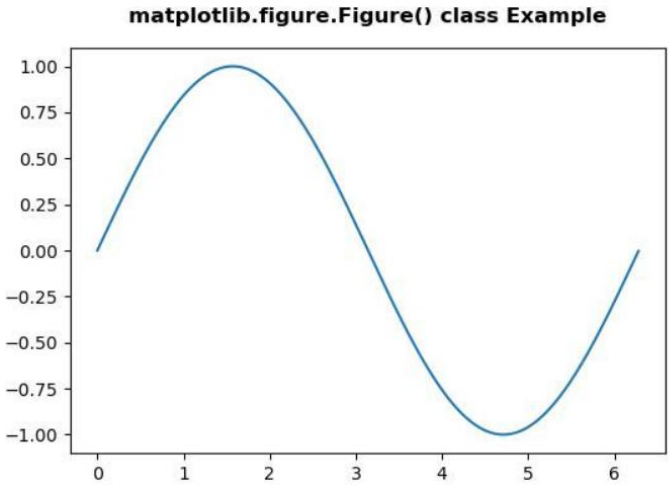


Criação de vários gráficos

```
# Python program to show pyplot module
    import matplotlib.pyplot as plt
    from matplotlib.figure import Figure
    # Creating a new figure with width = 5 inches
    # and height = 4 inches
    fig = plt.figure(figsize =(5, 4))
    # Creating first axes for the figure
    ax1 = fig.add_axes([1, 1, 1, 1])
    # Creating second axes for the figure
    ax2 = fig.add_axes([1, 0.5, 0.5, 0.5])
    # Adding the data to be plotted
    ax1.plot([2, 3, 4, 5, 5, 6, 6], [5, 7, 1, 3, 4, 6 ,8])
    ax2.plot([1, 2, 3, 4, 5], [2, 3, 4, 5, 6])
    plt.show()
```

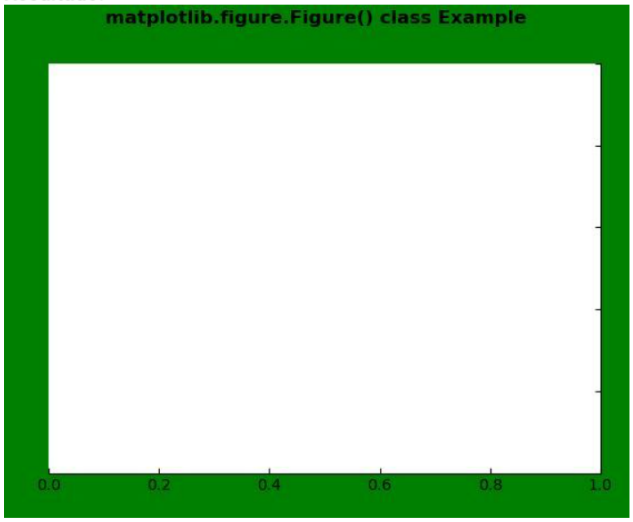


```
import matplotlib.pyplot as plt
from matplotlib.figure import Figure
import numpy as np


fig = plt.figure(figsize =(5, 4))

ax = fig.add_axes([0.1, 0.1, 0.8, 0.8])

xx = np.arange(0, 2 * np.pi, 0.01)
ax.plot(xx, np.sin(xx))

fig.suptitle('matplotlib.figure.Figure() class Example\n\n',
                    fontweight ="bold")


plt.show()
```




```
import matplotlib.pyplot as plt
from matplotlib.figure import Figure
from mpl_toolkits.axisartist.axislines import Subplot
import numpy as np


fig = plt.figure(facecolor ="green")

ax = Subplot(fig, 111)
fig.add_subplot(ax)

ax.axis["left"].set_visible(False)
ax.axis["top"].set_visible(False)

fig.suptitle('matplotlib.figure.Figure() class Example\n\n',
                    fontweight ="bold")


plt.show()
```

Resultado:



função matplotlib.figure.Figure.add_axes()

O módulo de figura do **método add_axes()** da biblioteca matplotlib é usado para adicionar eixos à figura.

Sintaxe: add_axes (self, * args, ** kwargs)

Parâmetros: aceita os seguintes parâmetros descritos abaixo:

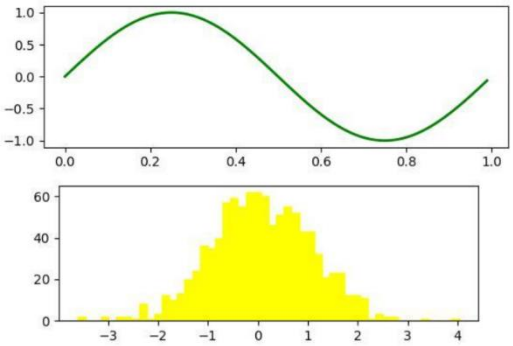
- **rect:** Este parâmetro são as dimensões [esquerda, inferior, largura, altura] dos novos eixos.
- **projeção:** Este parâmetro é o tipo de projeção dos eixos.
- **sharex, sharey:** Esses parâmetros compartilham o eixo x ou y com sharex e / ou sharey.
- **rótulo:** este parâmetro é o rótulo dos eixos retornados.

Retorna: Este método retorna a classe dos eixos que depende da projeção utilizada.

Os exemplos abaixo ilustram a função matplotlib.figure.Figure.add_axes() em matplotlib.figure:

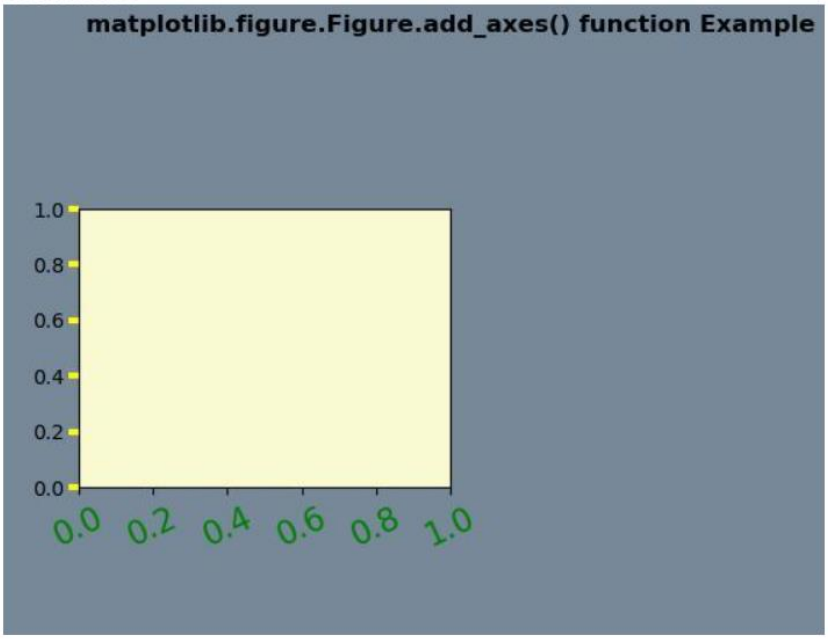
```
import numpy as np
import matplotlib.pyplot as plt

fig = plt.figure()
fig.subplots_adjust(top = 0.8)
ax1 = fig.add_subplot(211)

t = np.arange(0.0, 1.0, 0.01)
s = np.sin(2 * np.pi * t)
line, = ax1.plot(t, s, color ='green', lw = 2)

np.random.seed(19680801)

ax2 = fig.add_axes([0.15, 0.1, 0.7, 0.3])
n, bins, patches = ax2.hist(np.random.randn(1000), 50,
```

```
                                        facecolor ='yellow',
                                        edgecolor ='yellow')

fig.suptitle('matplotlib.figure.Figure.add_axes() \
function Example\n\n', fontweight ="bold")

plt.show()
```




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

fig =  plt.figure()
rect =  fig.patch
rect.set_facecolor('lightslategray')

ax1 =  fig.add_axes([0.1, 0.3, 0.4, 0.4])
rect =  ax1.patch
rect.set_facecolor('lightgoldenrodyellow')



for  label in  ax1.xaxis.get_ticklabels():
      label.set_color('green')
      label.set_rotation(25)
      label.set_fontsize(16)

for  line in  ax1.yaxis.get_ticklines():
      line.set_color('yellow')
      line.set_markersize(5)
      line.set_markeredgewidth(3)

fig.suptitle('matplotlib.figure.Figure.add_axes() \
function Example\n\n', fontweight ="bold")

plt.show()
```

Resultado:



função matplotlib.figure.Figure.clear()

O módulo de figura do **método clear()** da biblioteca matplotlib é usado para limpar a figura.

Sintaxe: clear (self, keep_observers = False)

Parâmetros: aceita os seguintes parâmetros descritos abaixo:

- **keep_observers:** este parâmetro é o valor booleano.

Retorna: Este método não retorna nenhum valor.

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

fig, ax =  plt.subplots()

ax.set_xlabel('x-axis')
ax.set_ylabel('y-axis')

ax.plot([1, 2, 3])
ax.grid(True)

fig.clear(True)

fig.suptitle('matplotlib.figure.Figure.clear() \
function Example\n\n', fontweight ="bold")

plt.show()
```

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

t = np.linspace(0.0, 2.0, 201)
s = np.sin(2 * np.pi * t)

fig, [ax, ax1] = plt.subplots(2, 1,
                                          sharex = True)

ax.set_ylabel('y-axis')
ax.plot(t, s)
ax.grid(True)
ax.set_title('Sample Example',
                  fontsize = 12,
                  fontweight ='bold')

ax1.set_ylabel('y-axis')
ax1.plot(t, s)
ax1.grid(True)

fig.clear(False)

fig.suptitle('matplotlib.figure.Figure.clear() \
function Example\n\n', fontweight ="bold")

plt.show()
```

função matplotlib.figure.Figure.colorbar()

O método colorbar() do módulo de figura da biblioteca matplotlib é usado para adicionar uma barra de cores a um gráfico.

Sintaxe: colorbar (self, mapeável, cax = None, ax = None, use_gridspec = True, ** kw)

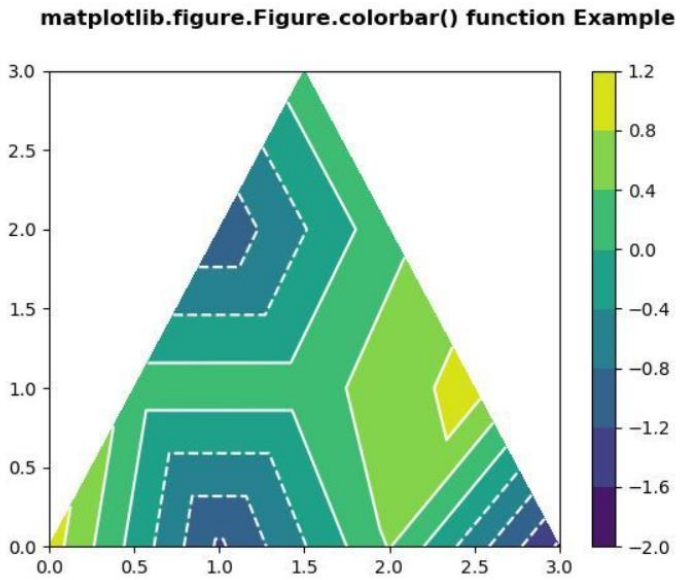
Parâmetros: aceita os seguintes parâmetros descritos abaixo:

- **mapeável:** este parâmetro é obrigatório para o método Figure.colorbar.
- **cax:** Este parâmetro são os eixos nos quais a barra de cores será desenhada.
- **ax:** Este parâmetro são os eixos pais dos quais o espaço para os novos eixos da barra de cores será roubado.
- **use_gridspec:** Este parâmetro é usado para criar uma instância do Subplot usando o módulo gridspec.

Retorna: Este método não retorna nenhum valor.

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

x = np.asarray([0, 1, 2, 3, 0.5,
```

```
                        1.5, 2.5, 1, 2,
                        1.5])

y = np.asarray([0, 0, 0, 0, 1.0,
                        1.0, 1.0, 2, 2,
                        3.0])

triangles = [[0, 1, 4], [1, 5, 4],
                    [2, 6, 5], [4, 5, 7],
                    [5, 6, 8], [5, 8, 7],
                    [7, 8, 9], [1, 2, 5],
                    [2, 3, 6]]

triang = mtri.Triangulation(x, y, triangles)
z = np.cos(3 * x) * np.cos(6 * y)+np.sin(6 * x)

fig, axs = plt.subplots()
t = axs.tricontourf(triang, z)
axs.tricontour(triang, z, colors ='white')
fig.colorbar(t)

fig.suptitle('matplotlib.figure.Figure.colorbar() \
function Example\n\n', fontweight ="bold")

plt.show()
```



Método matplotlib.figure.Figure.get_figwidth()

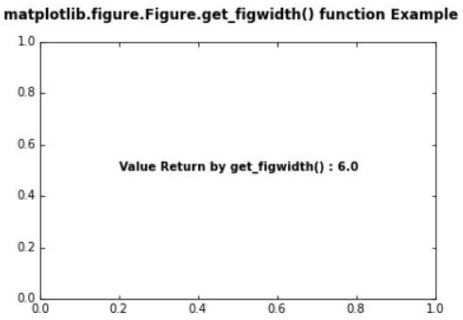
O módulo de figura do **método get_figwidth()** da biblioteca matplotlib é usado para obter a largura da figura como um flutuante.

Sintaxe: get_figwidth (self)

Parâmetros: este método não aceita nenhum parâmetro.

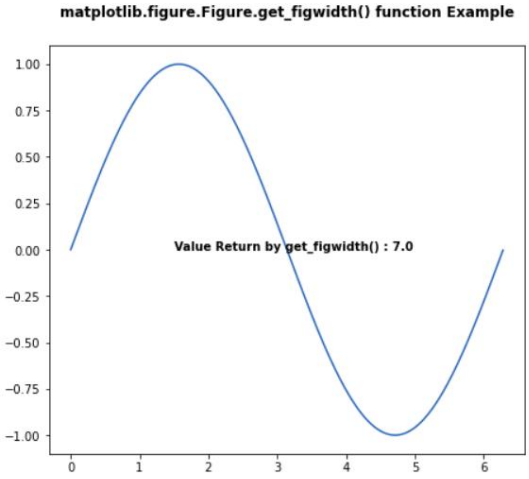
Retorna: Este método retorna a largura da figura.

```
import matplotlib.pyplot as plt
from matplotlib.figure import Figure
from mpl_toolkits.axisartist.axislines import Subplot
import numpy as np


fig = plt.figure()

ax = Subplot(fig, 111)
fig.add_subplot(ax)

w = fig.get_figwidth()
ax.text(0.2, 0.5,
            "Value Return by get_figwidth() : "
            + str(w),
            fontweight ="bold")

fig.canvas.draw()
fig.suptitle('matplotlib.figure.Figure.get_figwidth() \
function Example', fontweight ="bold")

plt.show()
```




```
import matplotlib.pyplot as plt
from matplotlib.figure import Figure
import numpy as np


fig = plt.figure(edgecolor = "red", figsize =(7, 6))

ax = fig.add_axes([0.1, 0.1, 0.8, 0.8])

xx = np.arange(0, 2 * np.pi, 0.01)
```

```
ax.plot(xx, np.sin(xx))

w = fig.get_figwidth()
ax.text(1.5, 0,
            "Value Return by get_figwidth() : "
            + str(w),
            fontweight ="bold")

fig.canvas.draw()
fig.suptitle('matplotlib.figure.Figure.get_figwidth() \
function Example', fontweight ="bold")

plt.show()
```



Método matplotlib.figure.Figure.get_figheight()

O módulo de figura do **método get_figheight()** da biblioteca matplotlib é usado para obter a altura da figura como um flutuador.

Sintaxe: get_figheight (self)

Parâmetros: este método não aceita nenhum parâmetro.

Retorna: Este método retorna a altura da figura.

```
import matplotlib.pyplot as plt
from matplotlib.figure import Figure
from mpl_toolkits.axisartist.axislines import Subplot
import numpy as np


fig = plt.figure()

ax = Subplot(fig, 111)
fig.add_subplot(ax)
```

```
w =  fig.get_figheight()
ax.text(0.2,  0.5,
            "Value Return by get_figheight() : "
            +  str(w),
            fontweight ="bold")

fig.canvas.draw()
fig.suptitle('matplotlib.figure.Figure.get_figheight()\
function Example', fontweight ="bold")

plt.show()
```

Método matplotlib.figure.Figure.subplots()

O módulo de figura do **método subplots()** da biblioteca matplotlib é usado para exibir a janela de figura.

Sintaxe: subplots (self, nrows = 1, ncols = 1, sharex = False, sharey = False, squeeze = True, subplot_kw = None, gridspec_kw = None)

Parâmetros: este método aceita os seguintes parâmetros descritos abaixo:

- **nrows, ncols:** Estes parâmetros são o número de linhas / colunas da grade do subplot.
- **sharex, sharey:** Este parâmetro controla o compartilhamento de propriedades entre os eixos x (sharex) ou y (sharey).
- **squeeze:** este parâmetro é um parâmetro opcional e contém o valor booleano com o padrão True.
- **num:** este parâmetro é a palavra-chave pyplot.figure que define o número da figura ou rótulo.
- **subplot_kwd:** Este parâmetro é o dict com palavras-chave passadas para a chamada add_subplot usada para criar cada subplot.
- **gridspec_kw:** Este parâmetro é o dict com palavras-chave passadas para o construtor GridSpec usado para criar a grade em que os subplots são colocados.

Retorna: Este método retorna os seguintes valores.

- axes: Este método retorna o objeto axes.Axes ou matriz de objetos Axes.

Os exemplos abaixo ilustram a função:

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



x =  np.linspace(0,  2  *  np.pi,  400)
y =  np.sin(x**2)

fig =  plt.figure()
ax =  fig.subplots()

ax.plot(x,  y)

fig.suptitle(
```

```
, fontweight ="bold")

fig.show()
```

Classe Axes

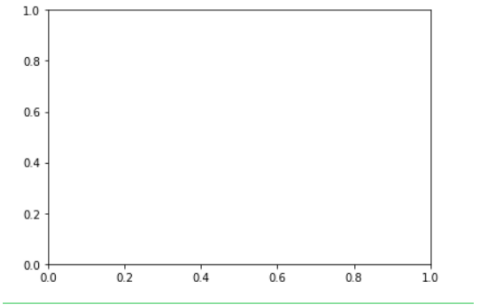
A classe Axes é a unidade mais básica e flexível para a criação de subplots. Uma determinada figura pode conter muitos eixos, mas determinados eixos só podem estar presentes numa figura. A função axes() cria o objeto de eixos. Vamos ver o exemplo abaixo.

Sintaxe:

matplotlib.pyplot.axis (* args, emit = True, ** kwargs)

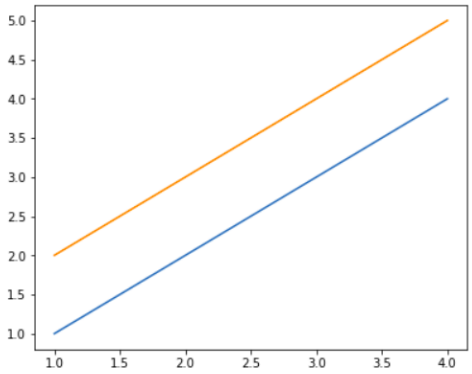
```
# Python program to show pyplot module
   import matplotlib.pyplot as plt
   from matplotlib.figure import Figure
   # Creating the axes object with argument as
   # [left, bottom, width, height]
   ax = plt.axes([1, 1, 1, 1])
```

Saída:




```
# Python program to show pyplot module
   import matplotlib.pyplot as plt
   from matplotlib.figure import Figure
   fig = plt.figure(figsize = (5, 4))
   # Adding the axes to the figure
   ax = fig.add_axes([1, 1, 1, 1])
   # plotting 1st dataset to the figure
   ax1 = ax.plot([1, 2, 3, 4], [1, 2, 3, 4])
   # plotting 2nd dataset to the figure
   ax2 = ax.plot([1, 2, 3, 4], [2, 3, 4, 5])
   plt.show()
```

Saída:



Os eixos permitem a colocação de desenhos em qualquer local da figura. Uma determinada figura pode conter muitos eixos, mas um determinado objeto de eixos só pode estar numa figura. Os eixos contêm dois objetos de eixo 2D, bem como objetos de três eixos no caso de 3D. Vejamos algumas funções básicas desta classe.

função axes()

`axes()` cria objetos de eixos com argumento, onde argumento é uma lista de 4 elementos [esquerda, parte inferior, largura, altura].

Sintaxe:

```
eixos ([esquerda, inferior, largura, altura])
```

```
import matplotlib.pyplot as plt


fig = plt.figure()
ax = plt.axes([0.1, 0.1, 0.8, 0.8])
```

Função matplotlib.axes.Axes.update()

A **função Axes.update()** no módulo de eixos da biblioteca matplotlib é usada para atualizar as propriedades dos eixos

Sintaxe: Axes.update (self, props)

Parâmetros: este método aceita os seguintes parâmetros.

- **props:** Este parâmetro é o dicionário das propriedades.

Retorna: Este método não retorna nenhum valor.

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

```
np.random.seed(10**7)
geeks = np.random.randn(100)

fig, ax = plt.subplots()
ax.acorr(geeks, usevlines = True,
             normed = True,
             maxlags = 80, lw = 3)

ax.grid(True)

prop = {'xticks': np.array([-10., -5.,  0.,   5., 10. ]),
            'yticks': np.array([-0.2,  0.2,  0.6,  1.,  1.4]),
            'ylabel': None, 'xlabel': None}

ax.update(prop)

fig.suptitle('matplotlib.axes.Axes.update()\
 function Example', fontweight ="bold")

plt.show()
```

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



xx = np.random.rand(16, 30)

fig, ax = plt.subplots()

m = ax.pcolor(xx)
m.set_zorder(-20)
prop = {'autoscalex_on': False}
w = ax.update(prop)

fig.suptitle('matplotlib.axes.Axes.update()\
function Example', fontweight ="bold")

plt.show()
```

Função matplotlib.axes.Axes.draw()

A **função Axes.draw()** no módulo de eixos da biblioteca matplotlib é usada para desenhar tudo.

Sintaxe: Axes.draw (self, renderer = None, inframe = False)

Parâmetros: este método aceita os seguintes parâmetros.

- **renderizador:** Este parâmetro é o primeiro parâmetro e seu valor padrão é Nenhum.
- **inframe:** Este parâmetro contém o valor booleano e seu valor padrão é falso.

Retorna: Este método não retorna nenhum valor.

```
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt


fig, ax = plt.subplots()

def tellme(s):
      ax.set_title(s, fontsize = 16)
      fig.canvas.draw()
      renderer = fig.canvas.renderer
      ax.draw(renderer)

tellme('matplotlib.axes.Axes.draw() function Example')
plt.show()
```

```
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt


fig = plt.figure()
ax = fig.add_subplot(111, projection ='3d')

X, Y, Z = axes3d.get_test_data(0.1)
ax.plot_wireframe(X, Y, Z, rstride = 5,
                          cstride = 5)

for angle in range(0, 90):
      ax.view_init(30, angle)
      fig.canvas.draw()
      renderer = fig.canvas.renderer
      ax.draw(renderer)
      plt.pause(.001)
      ax.set_title('matplotlib.axes.Axes.draw()\
      function Example', fontweight ="bold")
```

Função matplotlib.axes.Axes.get_figure()

A **função Axes.get_figure()** no módulo axes da biblioteca matplotlib é usada para obter a instância de Figura à qual pertence.

Sintaxe: Axes.get_figure (self)

Parâmetros: este método não aceita nenhum parâmetro.

Retorna: Este método retorna a instância da Figura à qual o artista pertence.

Função matplotlib.axes.Axes.set_figure()

A **função Axes.set_figure()** no módulo de eixos da biblioteca matplotlib é usada para definir a Figura para estes eixos.

Sintaxe: Axes.set_figure (self, fig)

Parâmetros: este método aceita apenas um parâmetro.

- **fig:** Este parâmetro é a instância da Figura.

Retorna: Este método não retorna nenhum valor.

Função matplotlib.axes.Axes.properties()

A **função Axes.properties()** no módulo de eixos da biblioteca matplotlib é usada para obter o dicionário de todas as propriedades.

Sintaxe: Axes.properties (self)

Parâmetros: este método não aceita nenhum parâmetro.

Retorna: Este método retorna o dicionário de todas as propriedades do artista.

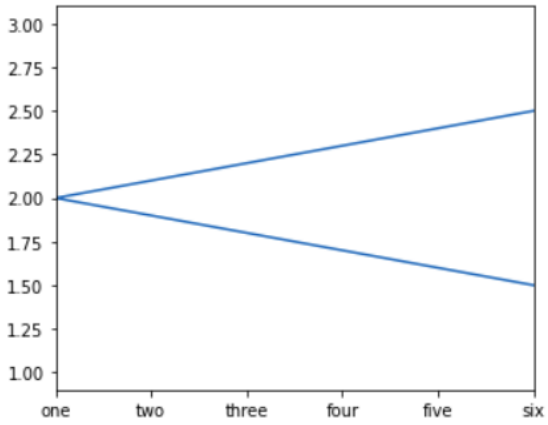
```
import  numpy as np
import  matplotlib.pyplot as plt

xx =  np.random.rand(16, 30)

fig, ax =  plt.subplots()

m =  ax.pcolor(xx)
m.set_zorder(-20)

w =  ax.properties()
print("Display all Properties\n")
for  i in  w:
      print(i, ":", w[i])

fig.suptitle('matplotlib.axes.Axes.properties()\
 function Example', fontweight ="bold")
```

```
plt.show()
import  matplotlib.pyplot as plt
import  numpy as np

np.random.seed(10**7)
geeks =  np.random.randn(100)

fig, ax =  plt.subplots()
ax.acorr(geeks, usevlines =  True,
             normed =  True,
             maxlags =  80, lw =  3)

ax.grid(True)

w =  ax.properties()
print("Display all Properties\n")
for  i in  w:
      print(i, ":", w[i])

fig.suptitle('matplotlib.axes.Axes.properties()\
function Example', fontweight ="bold")

plt.show()
```

Definindo limites e rótulos de escala

O Matplotlib define automaticamente os valores e os marcadores (pontos) dos eixos x e y, no entanto, é possível definir o limite e os marcadores manualmente. **As funções set_xlim()** e **set_ylim()** são usadas para definir os limites do eixo x e y, respetivamente. Similarmente, as **funções set_xticklabels()** e **set_yticklabels()** são usadas para definir rótulos de escala.

```
# Python program to show pyplot module
   import matplotlib.pyplot as plt
   from matplotlib.figure import Figure
   x = [3, 1, 3]
   y = [3, 2, 1]
   # Creating a new figure with width = 5 inches
   # and height = 4 inches
   fig = plt.figure(figsize =(5, 4))
   # Creating first axes for the figure
   ax = fig.add_axes([0.1, 0.1, 0.8, 0.8])
   # Adding the data to be plotted
   ax.plot(x, y)
   ax.set_xlim(1, 2)
   ax.set_xticklabels((
   "one", "two", "three", "four", "five", "six"))
   plt.show()
```



Múltiplos Gráficos

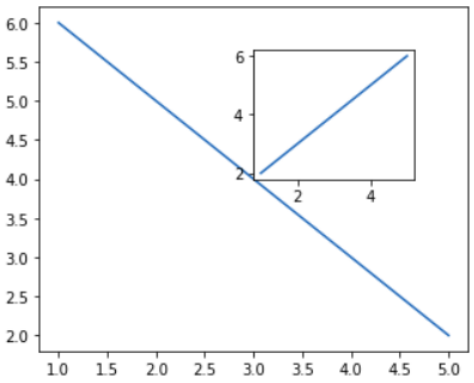
O desenho de múltiplos gráficos pode ser feito de várias formas. Uma maneira foi discutida acima usando o método add_axes() da classe figure. Vamos, no entanto, ver várias formas pelas quais vários gráficos podem ser adicionados com a ajuda de exemplos.

Método 1: usando o método add_axes()

O módulo de figura do método add_axes() da biblioteca matplotlib é usado para adicionar eixos à figura.

Sintaxe: add_axes (self, * args, ** kwargs)

```
# Python program to show pyplot module
   import matplotlib.pyplot as plt
   from matplotlib.figure import Figure
   # Creating a new figure with width = 5 inches
   # and height = 4 inches
   fig = plt.figure(figsize =(5, 4))
   # Creating first axes for the figure
   ax1 = fig.add_axes([0.1, 0.1, 0.8, 0.8])
   # Creating second axes for the figure
   ax2 = fig.add_axes([0.5, 0.5, 0.3, 0.3])
   # Adding the data to be plotted
   ax1.plot([5, 4, 3, 2, 1], [2, 3, 4, 5, 6])
   ax2.plot([1, 2, 3, 4, 5], [2, 3, 4, 5, 6])
   plt.show()
```



O método add_axes() adiciona o gráfico na mesma figura criando outro objeto de eixos.

Método 2: Usando o método subplot() .

Este método adiciona outro gráfico à figura atual na posição de grade especificada.

```
import matplotlib.pyplot as plt
   # data to display on plots
   x = [3, 1, 3]
   y = [3, 2, 1]
   z = [1, 3, 1]
   # Creating figure object
   plt.figure()
   # addind first subplot
   plt.subplot(121)
   plt.plot(x, y)
   # addding second subplot
   plt.subplot(122)
   plt.plot(z, y)
```

A função subplot exclui o gráfico preexistente da figura.

Método 3: usando o método subplots()

```
import matplotlib.pyplot as plt
   # Creating the figure and subplots
   # according the argument passed
   fig, axes = plt.subplots(1, 2)
   # plotting the data in the 1st subplot
   axes[0].plot([1, 2, 3, 4], [1, 2, 3, 4])
   # plotting the data in the 1st subplot only
   axes[0].plot([1, 2, 3, 4], [4, 3, 2, 1])
   # plotting the data in the 2nd subplot only
   axes[1].plot([1, 2, 3, 4], [1, 1, 1, 1])
```

Método 4: usando o método subplot2grid()

Esta função fornece flexibilidade adicional na criação de objetos de eixos num local especificado dentro de uma grid. Também ajuda a estender o objeto de eixos em várias linhas ou colunas. Em palavras mais simples, esta função é usada para criar vários gráficos dentro da mesma figura.

Sintaxe: Plt.subplot2grid (forma, localização, rowspan, colspan)

```
import matplotlib.pyplot as plt
   # data to display on plots
   x = [3, 1, 3]
   y = [3, 2, 1]
   z = [1, 3, 1]
   # adding the subplots
   axes1 = plt.subplot2grid (
   (7, 1), (0, 0), rowspan = 2,  colspan = 1)
   axes2 = plt.subplot2grid (
   (7, 1), (2, 0), rowspan = 2, colspan = 1)
   axes3 = plt.subplot2grid (
   (7, 1), (4, 0), rowspan = 2, colspan = 1)
   # plotting the data
   axes1.plot(x, y)
   axes2.plot(x, z)
   axes3.plot(z, y)
```

Diferentes Tipos de Gráficos

Gráfico de Linhas

Até ao momento estivemos a trabalhar com gráficos de linhas, pois são os mais fáceis de desenhar.

Estilos de linha

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

students =  ["Jane","Joe","Beck","Tom",
                "Sam","Eva","Samuel","Jack",
                "Dana","Ester","Carla","Steve",
                "Fallon","Liam","Culhane","Candance",
                "Ana","Mari","Steffi","Adam"]


marks=[]
for  i in  range(0,len(students)):
       marks.append(random.randint(0, 101))

plt.xlabel("Students")
plt.ylabel("Marks")
plt.title("CLASS RECORDS")
plt.plot(students,marks,'m--')
```

Traçando uma única linha horizontal

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

```
plt.axhline(y = 0.5, color = 'r', linestyle = '-')
plt.show()
```

Traçando várias linhas horizontais

Para desenhar várias linhas horizontais, use o método axhline() várias vezes.

```
import matplotlib.pyplot as plt
plt.axhline(y = .5, xmin = 0.25, xmax = 0.9)
plt.axhline(y = 3, color = 'b', linestyle = ':')
plt.axhline(y = 1, color = 'w', linestyle = '--')
plt.axhline(y = 2, color = 'r', linestyle = 'dashed')

plt.xlabel('x - axis')
plt.ylabel('y - axis')
plt.show()
```

Adicionando a legenda

A legenda pode ser adicionada usando a função **legend()** .

```
import matplotlib.pyplot as plt
plt.axhline(y = .5, xmin = 0.25, xmax = 0.9)
plt.axhline(y = 3, color = 'b', linestyle = ':', label = "blue line")
plt.axhline(y = 1, color = 'w', linestyle = '--', label = "white line")
plt.axhline(y = 2, color = 'r', linestyle = 'dashed', label = "red line")

plt.xlabel('x - axis')
plt.ylabel('y - axis')
plt.legend(bbox_to_anchor = (1.0, 1), loc = 'upper center')
plt.show()
```

Fazendo uma única linha vertical

Método # 1: usando **axvline().**Esta função adiciona as linhas verticais ao longo dos eixos do gráfico

Sintaxe : matplotlib.pyplot.axvline (x, color, xmin, xmax, linestyle)

Parâmetros:

- **x:** Posição no eixo X para traçar a linha, aceita inteiros.
- **xmin e xmax:** escalar, opcional, padrão: 0/1. Ele traça a linha no intervalo determinado
- **color:** cor para a linha, aceita um string. por exemplo, 'r' ou 'b'.
- **linestyle:** especifica o tipo de linha, aceita uma string. por exemplo, '-', '-', '-.', ':', 'Nenhum', '', ”, 'sólido', 'tracejado', 'dashdot', 'pontilhado'

```
import matplotlib.pyplot as plt
import numpy as np
plt.figure(figsize = (10, 5))
plt.axvline(x = 7, color = 'b', label = 'axvline - full height')
plt.show()
```

Método 2: Usando **vlines()**

matplotlib.pyplot.vlines() é uma função usada no desenho de um conjunto de dados. Em matplotlib.pyplot.vlines(), vlines é a abreviatura de linhas verticais. O que essa função faz fica muito claro na forma expandida, que diz que a função lida com o desenho das linhas verticais nos eixos.

Sintaxe : vlines (x, ymin, ymax, cores, estilos de linha)

Parâmetros:

- **x:** Posição no eixo X para traçar a linha, aceita inteiros.
- **xmin e xmax:** escalar, opcional, padrão: 0/1. Ele traça a linha no intervalo determinado
- **color:** cor para a linha, aceita um string. por exemplo, 'r' ou 'b'.
- **linestyle:** especifica o tipo de linha, aceita uma string. por exemplo, '-', '-', '-.', ':', 'Nenhum', '', ”, 'sólido', 'tracejado', 'dashdot', 'pontilhado'

```
import matplotlib.pyplot as plt
import numpy as np
xs = [1, 100]
plt.figure(figsize = (10, 7))
plt.vlines(x = 37, ymin = 0, ymax = max(xs),
               colors = 'purple',
               label = 'vline_multiple - full height')

plt.show()
```

Método # 3: Usando **plot()**

A função plot() no módulo pyplot da biblioteca matplotlib é usada para fazer um gráfico binning hexagonal 2D dos pontos x, y.

Sintaxe: plot (x_points, y_points, scaley = False)

Parâmetros:

- **x_points / y_points:** pontos para traçar
- **scalex / scaley:** Bool, estes parâmetros determinam se os limites de visão são adaptados aos limites de dados

```
import matplotlib.pyplot as plt
plt.figure(figsize = (10, 5))
plt.plot((0, 0), (0, 1), scaley = False)
plt.show()
```

Traçar várias linhas com a legenda

Os métodos abaixo podem ser usados para plotar várias linhas em Python.

Método # 1: usando axvline()

```
import matplotlib.pyplot as plt
import numpy as np
plt.figure(figsize = (10, 5))
plt.axvline(x = 7, color = 'b', label = 'axvline - full height')
plt.axvline(x = 7.25, ymin = 0.1, ymax = 0.90, color = 'r',
            label = 'axvline - % of full height')
plt.legend(bbox_to_anchor = (1.0, 1), loc = 'upper left')
plt.show()
```

Método 2: Usando vlines()

```
import matplotlib.pyplot as plt
import numpy as np
xs = [1, 100]
plt.figure(figsize = (10, 7))
plt.vlines(x = [37, 37.25, 37.5], ymin = 0, ymax = max(xs),
           colors = 'purple',
           label = 'vline_multiple - full height')
plt.vlines(x = [38, 38.25, 38.5], ymin = [0, 25, 75], ymax = max(xs),
           colors = 'teal',
           label = 'vline_multiple - partial height')
plt.vlines(x = 39, ymin = 0, ymax = max(xs), colors = 'green',
           label = 'vline_single - full height')
plt.vlines(x = 39.25, ymin = 25, ymax = max(xs), colors = 'green',
           label = 'vline_single - partial height')
plt.legend(bbox_to_anchor = (1.0, 1), loc = 'up')
plt.show()
```

ALTERAR A OPACIDADE DA LINHA NO MATPLOTLIB

- Um gráfico de linha ou gráfico de linha pode ser um tipo de gráfico que exibe informações como uma série de pontos de conhecimento chamados 'marcadores' conectados por segmentos de linha. Os gráficos de linha costumam encontrar relacionamentos entre dois conjuntos de dados em eixos diferentes; como exemplo X, Y.
- Matplotlib permite regular a transparência de um gráfico usando o atributo alpha.
- Por padrão, alfa = 1.
- Se desejarmos tornar o gráfico mais transparente, colocamos alfa menor que , como 0,5 ou 0,25.
- Se quisermos o desenho menos transparente então colocamos alpha maior que 1. Isso solidifica o desenho do gráfico, tornando-o menos transparente e mais espesso e denso, por assim dizer.

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5]
y = x
plt.plot(x, y, linewidth=10, alpha=0.2)
plt.show()
```

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([-2, -1, 0, 1, 2])
y1 = x*0
y2 = x*x
y3 = -x*x
plt.plot(x, y2, alpha=0.2)
plt.plot(x, y1, alpha=0.5)
plt.plot(x, y3, alpha=1)
plt.legend(["op = 0.2", "op = 0.5", "op = 1"])
plt.show()
```

```
import matplotlib.pyplot as plt
import numpy as np
x = [1, 2, 3, 4, 5]
for i in range(10):
      plt.plot([1, 2.8], [i]*2, linewidth=5, color='red', alpha=0.1*i)
      plt.plot([3.1, 4.8], [i]*2, linewidth=5, color='green', alpha=0.1*i)
      plt.plot([5.1, 6.8], [i]*2, linewidth=5, color='yellow', alpha=0.1*i)
      plt.plot([7.1, 8.8], [i]*2, linewidth=5, color='blue', alpha=0.1*i)

for i in range(10):
      plt.plot([1, 2.8], [-i]*2, linewidth=5, color='red', alpha=0.1*i)
      plt.plot([3.1, 4.8], [-i]*2, linewidth=5, color='green', alpha=0.1*i)
      plt.plot([5.1, 6.8], [-i]*2, linewidth=5, color='yellow', alpha=0.1*i)
      plt.plot([7.1, 8.8], [-i]*2, linewidth=5, color='blue', alpha=0.1*i)

plt.show()
```

Largura da linha : Por padrão, a largura da linha é 1. Para gráficos com várias linhas, torna-se difícil traçar as linhas com cores mais claras. Esta situação pode ser tratada aumentando a largura da linha. A largura de linha pode ser usada para focar em certos dados em comparação com outros. Isso pode ajudar a obter uma visualização detalhada do registo específico no conjunto de dados. Este atributo pertence à função plot().

```
import matplotlib.pyplot as plt

places = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J"]
literacy_rate = [100, 98, 90, 85, 75, 50, 30, 45, 65, 70]
female_literacy = [95, 100, 50, 60, 85, 80, 75, 99, 70, 30]

plt.xlabel("Places")
plt.ylabel("Percentage")

plt.plot(places, literacy_rate, color='blue',
             linewidth=6, label="Literacy rate")
```

```
plt.plot(places, female_literacy, color='fuchsia',
             linewidth=4, label="Female Literacy rate")

plt.legend(loc='lower left', ncol=1)
```

```
import matplotlib.pyplot as plt

age = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
cardiac_cases = [5, 15, 20, 40, 55, 55, 70, 80, 90, 95]
survival_chances = [99, 99, 90, 90, 80, 75, 60, 50, 30, 25]

plt.xlabel("Age")
plt.ylabel("Percentage")

plt.plot(age, cardiac_cases, color='black', linewidth=2,
             label="Cardiac Cases", marker='o', markerfacecolor='red',
markersize=12)

plt.plot(age, survival_chances, color='yellow', linewidth=3,
             label="Survival Chances", marker='o', markerfacecolor='green',
markersize=12)

plt.legend(loc='lower right', ncol=1)
```

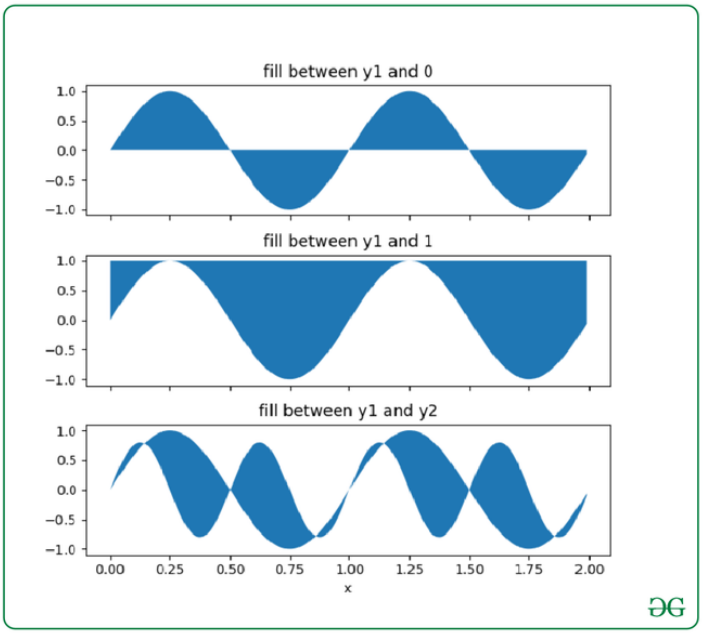
Com o uso da função **fill_between()** na biblioteca Matplotlib em Python, podemos facilmente preencher a cor entre quaisquer linhas múltiplas ou quaisquer duas curvas horizontais em um plano 2D.

Sintaxe: *matplotlib.pyplot.fill_between (x, y1, y2 = 0, onde = None, step = None, interpolar = False, *, data = None, ** kwargs)*

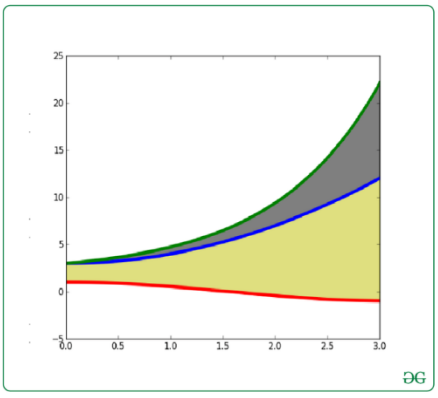
Exemplo 1: Cor entre a curva da função matemática **f (x) = sin (x)**

```
import pylab as plt
import numpy as np

x = np.arange(0.0, 2, 0.01)
y1 = np.sin(2 * np.pi * x)
y2 = 0.8 * np.sin(4 * np.pi * x)

fig, (ax1, ax2, ax3) = plt.subplots(
   3, 1, sharex=True, figsize=(6, 6))

ax1.fill_between(x, y1)
ax1.set_title('fill between y1 and 0')

ax2.fill_between(x, y1, 1)
ax2.set_title('fill between y1 and 1')
```

```
ax3.fill_between(x, y1, y2)
ax3.set_title('fill between y1 and y2')
ax3.set_xlabel('x')
fig.tight_layout()
```




```
import pylab as plt
import numpy as np

X = np.linspace(0, 3, 200)
Y1 = X**2 + 3
Y2 = np.sin(X)
Y3 = np.cos(X)

plt.plot(X, Y1, lw=4)
plt.plot(X, Y2, lw=4)
plt.plot(X, Y3, lw=4)

plt.fill_between(X, Y1, Y2, color='k', alpha=.5)
plt.fill_between(X, Y1, Y3, color='y', alpha=.5)

plt.show()
```



```
import  matplotlib.pyplot as plt



x =  [1, 2, 1, 0]
y =  [2, 1, 0, 1]

plt.fill(x, y)
plt.show()
```

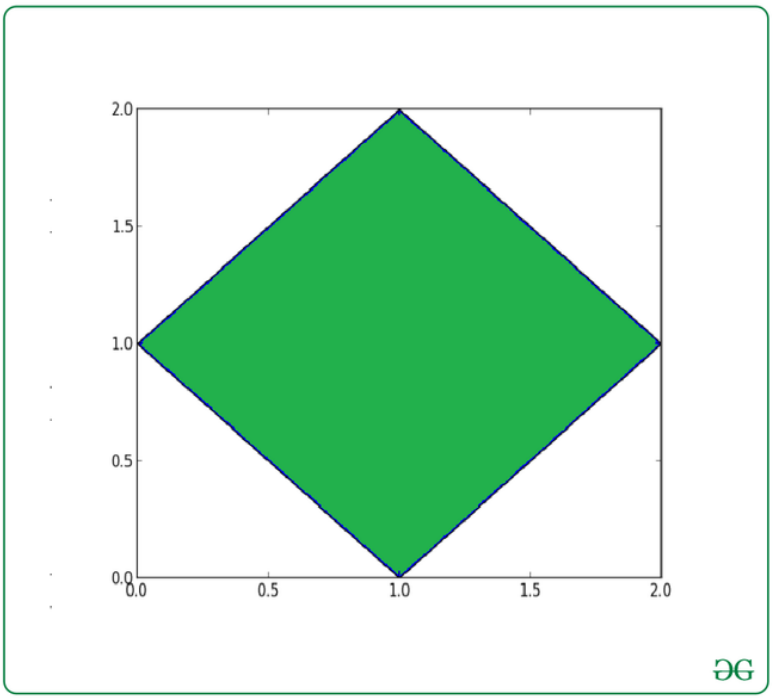


Gráfico de barras

Um **gráfico de barras** é um gráfico que representa a categoria de dados com barras retangulares com comprimentos e alturas proporcionais aos valores que representam. Os gráficos de barra podem ser desenhadps horizontal ou verticalmente. Um gráfico de barras descreve as comparações entre as categorias discretas. Ele pode ser criado usando o método **bar()** .

Sintaxe:

plt.bar (x, altura, largura, fundo, alinhar)

```
import matplotlib.pyplot as plt
   # data to display on plots
   x = [3, 1, 3, 12, 2, 4, 4]
   y = [3, 2, 1, 4, 5, 6, 7]
   # This will plot a simple bar chart
```

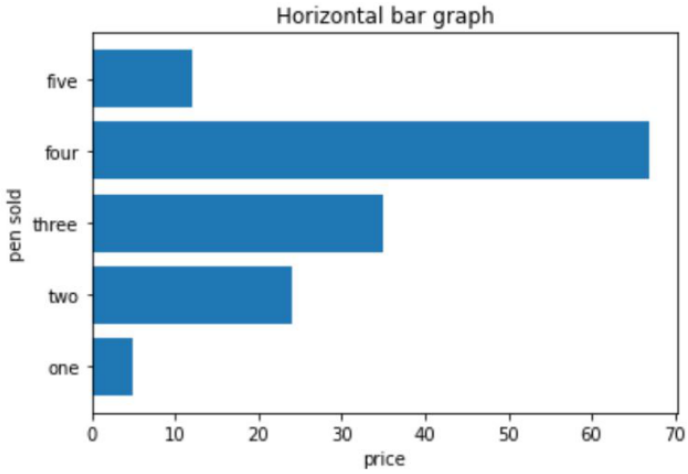
```
    plt.bar(x, y)
    # Title to the plot
    plt.title("Bar Chart")
    # Adding the legends
    plt.legend(["bar"])
    plt.show()
```

A seguir está um exemplo simples do gráfico de barras, que representa o número de alunos matriculados em diferentes cursos de uma escola.

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



data = {'C':20, 'C++':15, 'Java':30,
            'Python':35}
courses = list(data.keys())
values = list(data.values())

fig = plt.figure(figsize = (10, 5))
plt.bar(courses, values, color ='maroon',
            width = 0.4)


plt.xlabel("Courses offered")
plt.ylabel("No. of students enrolled")
plt.title("Students enrolled in different courses")
plt.show()
```

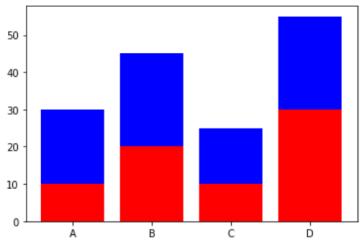
Criação de um gráfico de barras horizontais (função barh)

```
import matplotlib.pyplot as plt
y=['one', 'two', 'three', 'four', 'five']
x=[5,24,35,67,12]
plt.barh(y, x)
plt.ylabel("pen sold")
plt.xlabel("price")
plt.title("Horizontal bar graph")
plt.show()
```

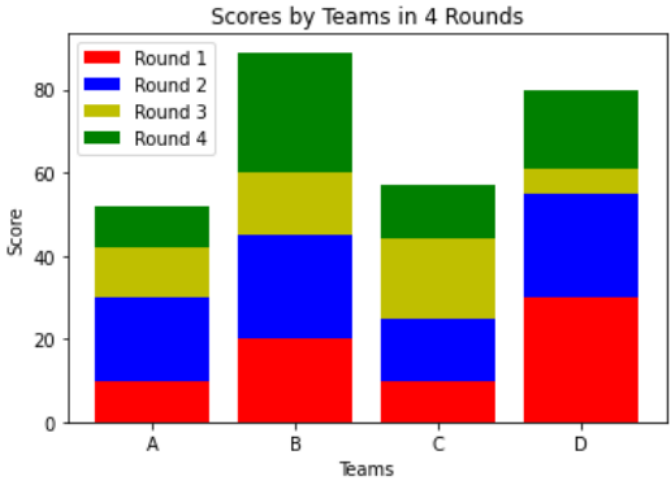


Criação de um gráfico de barras empilhados

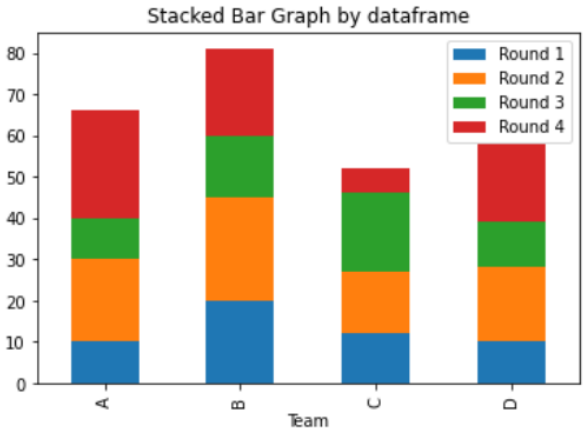
```
import matplotlib.pyplot as plt
x = ['A', 'B', 'C', 'D']
y1 = [10, 20, 10, 30]
y2 = [20, 25, 15, 25]
plt.bar(x, y1, color='r')
plt.bar(x, y2, bottom=y1, color='b')
plt.show()
```




```
import matplotlib.pyplot as plt
import numpy as np
x = ['A', 'B', 'C', 'D']
y1 = np.array([10, 20, 10, 30])
y2 = np.array([20, 25, 15, 25])
y3 = np.array([12, 15, 19, 6])
y4 = np.array([10, 29, 13, 19])
plt.bar(x, y1, color='r')
plt.bar(x, y2, bottom=y1, color='b')
plt.bar(x, y3, bottom=y1+y2, color='y')
plt.bar(x, y4, bottom=y1+y2+y3, color='g')
plt.xlabel("Teams")
plt.ylabel("Score")
plt.legend(["Round 1", "Round 2", "Round 3", "Round 4"])
plt.title("Scores by Teams in 4 Rounds")
plt.show()
```



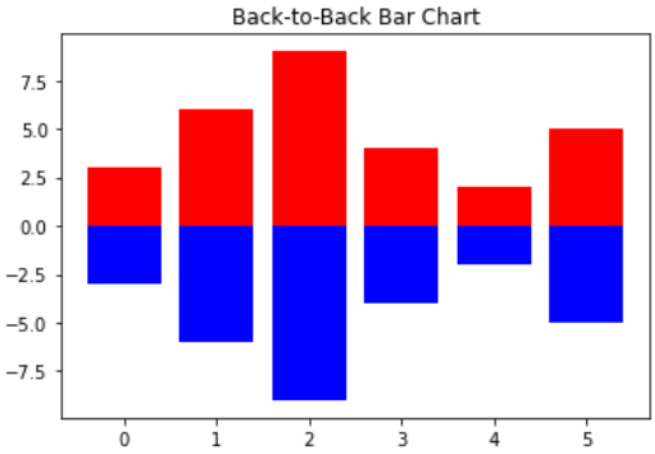

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
df = pd.DataFrame([['A', 10, 20, 10, 26], ['B', 20, 25, 15, 21], ['C', 12, 15, 19,
6], ['D', 10, 18, 11, 19]], columns=['Team', 'Round 1', 'Round 2', 'Round 3',
'Round 4'])
print(df)
df.plot(x='Team', kind='bar', stacked=True,
            title='Stacked Bar Graph by dataframe')
```



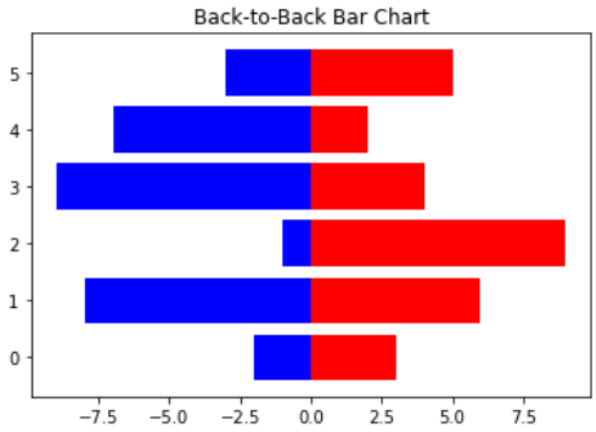
Gráficos de barras consecutivas matplotlib

Gráfico de barras back-to-back: O gráfico de barras back-to-back é apenas uma combinação de dois gráficos de barras desenhados em relação a um eixo.

```
import numpy as np
import matplotlib.pyplot as plt
A = np.array([3,6,9,4,2,5])
X = np.arange(6)
plt.bar(X, A, color = 'r')
plt.bar(X, -A, color = 'b')
plt.title("Back-to-Back Bar Chart")
plt.show()
```

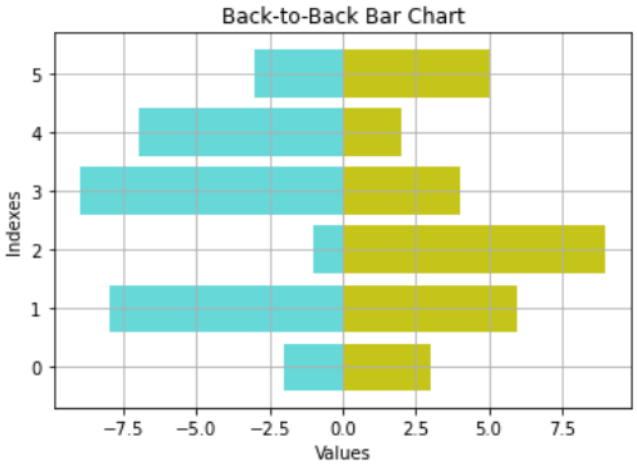



```
import  numpy as np
import  matplotlib.pyplot as plt
A =  np.array([3,6,9,4,2,5])
B =  np.array([2,8,1,9,7,3])
X =  np.arange(6)
plt.barh(X,  A,  color =  'r')
plt.barh(X,  -B,  color =  'b')
plt.title("Back-to-Back Bar Chart")
plt.show()
```




```
import  numpy as np
import  matplotlib.pyplot as plt
A =  np.array([3,6,9,4,2,5])
B =  np.array([2,8,1,9,7,3])
X =  np.arange(6)
plt.barh(X,  A,  align='center',
             alpha=0.9,  color =  'y')

plt.barh(X,  -B,  align='center',
             alpha=0.6,  color =  'c')

plt.grid()
plt.title("Back-to-Back Bar Chart")
plt.ylabel("Indexes")
plt.xlabel("Values")
```

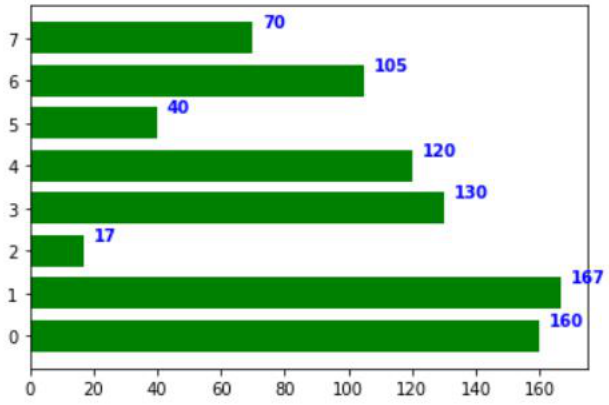
```
plt.show()
```



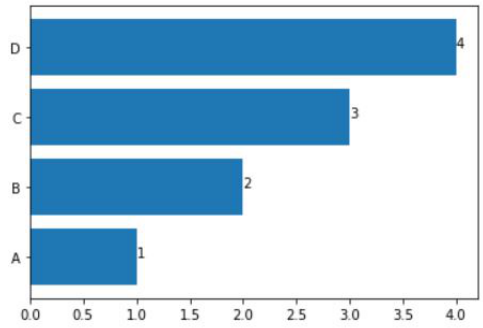
Usando a função matplotlib.axes.Axes.text() :

Esta função é usada basicamente para adicionar algum texto à localização no gráfico.

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

x = [0, 1, 2, 3, 4, 5, 6, 7]
y = [160, 167, 17, 130, 120, 40, 105, 70]
fig, ax = plt.subplots()
width = 0.75
ind = np.arange(len(y))

ax.barh(ind, y, width, color = "green")

for i, v in enumerate(y):
    ax.text(v + 3, i + .25, str(v),
            color = 'blue', fontweight = 'bold')
plt.show()
```



```
import matplotlib.pyplot as plt
x = ["A", "B", "C", "D"]
y = [1, 2, 3, 4]
plt.barh(x, y)

for index, value in enumerate(y):
      plt.text(value, index,
                     str(value))

plt.show()
```



Histogramas

Um **histograma** é basicamente usado para representar dados na forma de alguns grupos. É um tipo de gráfico de barra em que o eixo X representa os intervalos de bin enquanto o eixo Y fornece informações sobre a frequência. Para criar um histograma, o primeiro passo é criar um bin dos intervalos, em seguida, distribuir todo o intervalo dos valores em uma série de intervalos e contar os valores que caem em cada um dos intervalos. Os compartimentos são claramente identificados como intervalos de variáveis consecutivos e não sobrepostos. A função hist() é usada para calcular e criar um histograma de x.

Sintaxe:

matplotlib.pyplot.hist (x, bins = Nenhum, intervalo = Nenhum, densidade = Falso, pesos = Nenhum, cumulativo = Falso, inferior = Nenhum, histtype = 'barra', alinhamento = 'meio', orientação = 'vertical', rwidth = None, log = False, color = None, label = None, stacked = False, \ *, data = None, \ * \ * kwargs)

```
import matplotlib.pyplot as plt
   # data to display on plots
   x = [1, 2, 3, 4, 5, 6, 7, 4]
   # This will plot a simple histogram
   plt.hist(x, bins = [1, 2, 3, 4, 5, 6, 7])
   # Title to the plot
   plt.title("Histogram")
   # Adding the legends
   plt.legend(["bar"])
   plt.show()
```

```
from matplotlib import pyplot as plt
```

```
import numpy as np



data_set = np.random.randint(100, size=(50))
fig = plt.figure(figsize=(10, 6))
plt.hist(data_set, bins=[0, 10, 20, 30, 40, 50,
                                    60, 70, 80, 90, 100])
plt.title("Random Histogram")
plt.show()
```

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



data_set = [45, 85, 95, 10, 58, 77, 92, 72, 52,
                22, 32, 5, 95, 2, 23, 24, 50, 40, 60,
                69, 44, 80, 21, 15, 17, 55, 21, 88]
fig = plt.figure(figsize=(10, 5))
plt.hist(data_set, bins=[0, 15, 30, 45, 60, 75, 90, 105])
plt.title("Predefined Histogram")
plt.show()
```

Matplotlib fornece uma variedade de métodos diferentes para personalizar o histograma. `matplotlib.pyplot.hist()` A própria função fornece muitos atributos com a ajuda dos quais podemos modificar um histograma. A `hist()` função fornece um objeto `patches` que dá acesso às propriedades dos objetos criados, e assim podemos modificar o gráfico de acordo com nossa vontade.

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import colors
from matplotlib.ticker import PercentFormatter
np.random.seed(23685752)
N_points = 10000
n_bins = 20
x = np.random.randn(N_points)
y = .8 ** x + np.random.randn(10000) + 25
fig, axs = plt.subplots(1, 1,
                                    figsize =(10, 7),
                                    tight_layout = True)


axs.hist(x, bins = n_bins)
plt.show()
```

Dois histogramas na mesma imagem:

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

series1 = np.random.randn(500, 1)
```

```
series2 =  np.random.randn(400, 1)
plt.hist(series1)
plt.hist(series2)
plt.show()
```

Gráfico de dispersão

Os gráficos de dispersão são usados para observar a relação entre as variáveis e usa pontos para representar a relação entre elas. O método **scatter()** na biblioteca matplotlib é usado para desenhar um gráfico de dispersão.

Sintaxe:

matplotlib.pyplot.scatter (x_axis_data, y_axis_data, s = None, c = None, marker = None, cmap = None, vmin = None, vmax = None, alpha = None, linewidths = None, edgecolors = None)

```
import matplotlib.pyplot as plt
   # data to display on plots
   x = [3, 1, 3, 12, 2, 4, 4]
   y = [3, 2, 1, 4, 5, 6, 7]
   # This will plot a simple scatter chart
   plt.scatter(x, y)
   # Adding legend to the plot
   plt.legend("A")
   # Title to the plot
   plt.title("Scatter chart")
   plt.show()
import  matplotlib.pyplot as plt
import  numpy as np
x =  np.arange(1, 6)
y1 =  x**2
y2 =  x**4
plt.scatter(x, y1, label="x**2")
plt.scatter(x, y2, label="x**4")
plt.legend()
plt.show()
```

Com várias cores:

```
import  matplotlib.pyplot as plt
x =  [1, 2, 3, 4]
y =  [4, 1, 3, 6]
plt.scatter(x, y, c='green')
x =  [5, 6, 7, 8]
y =  [1, 3, 5, 2]
plt.scatter(x, y, c='red')
plt.show()
```

Aumentando o tamanho dos pontos

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

```
import  numpy as np

plt.style.use('seaborn')

x =  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
y =  [8, 7, 6, 4, 5, 6, 7, 8, 9, 10]

plt.xticks(np.arange(11))
plt.yticks(np.arange(11))

plt.scatter(x, y, s=500, c='g')

plt.title("Scatter Plot", fontsize=25)

plt.xlabel('x-axis', fontsize=18)
plt.ylabel('y-axis', fontsize=18)

plt.show()
```

Pontos com tamanho variável

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

plt.style.use('seaborn')

x =  [1,2,3,4,5,6,7,8,9,10,11,12]
y =  [1,2,3,4,5,6,7,8,9,10,11,12]
points_size =  [100,200,300,400,500,600,700,800,900,1000,1100,1200]



plt.xticks(np.arange(13))
plt.yticks(np.arange(13))

plt.scatter(x,y,s=points_size,c='g')

plt.title("Scatter Plot with increase in size of scatter points ", fontsize=22)

plt.xlabel('x-axis',fontsize=20)
plt.ylabel('y-axis',fontsize=20)

plt.show()
```

```
import  matplotlib.pyplot as plt

plt.style.use('seaborn')
plt.figure(figsize=(10, 10))

x =  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
y =  [3*i+2  for  i in  x]
```

```
size =  [n*100  for  n in  range(1, len(x)+1)]


plt.scatter(x, y, s=size, c='g')
plt.title("Scatter Plot with increase in size of scatter points ", fontsize=22)


plt.xlabel('X-axis', fontsize=20)
plt.ylabel('Y-axis', fontsize=20)


plt.xticks(x, fontsize=12)
plt.yticks(y, fontsize=12)


plt.show()
```

Gráfico de Circular

A área do gráfico é a percentagem total dos dados fornecidos. A área dos setores circulares representa a percentagem das partes dos dados. Podem ser criados usando o método pie().

Sintaxe:

matplotlib.pyplot.pie (data, explode = None, labels = None, colors = None, autopct = None, shadow = False)

```
import matplotlib.pyplot as plt
   # data to display on plots
   x = [1, 2, 3, 4]
   # this will explode the 1st wedge
   # i.e. will separate the 1st wedge
   # from the chart
   e  =(0.1, 0, 0, 0)
   # This will plot a simple pie chart
   plt.pie(x, explode = e)
   # Title to the plot
   plt.title("Pie chart")
   plt.show()
```

A **função Axes.pie()** no módulo de eixos da biblioteca matplotlib é usada para traçar um gráfico de pizza.

Sintaxe: Axes.pie (self, x, explode = None, labels = None, colors = None, autopct = None, pctdistance = 0.6, shadow = False, labeldistance = 1.1, startangle = None, radius = None, counterclock = True, wedgeprops = None, textprops = None, center = (0, 0), frame = False, rotatelabels = False, *, data = None)

Parâmetros: este método aceita os seguintes parâmetros descritos abaixo:

- **x:** Este parâmetro é o tamanho das cunhas.
- **explodir:** Este parâmetro é um array len (x) que especifica a fração do raio com a qual compensar cada fatia.
- **autopct:** Este parâmetro é uma string ou função usada para rotular as fatias com seu valor numérico.
- **cores:** este parâmetro é a sequência de argumentos de cores matplotlib através dos quais o gráfico de pizza circulará.
- **rótulo:** este parâmetro é a sequência de strings que fornece os rótulos para cada fatia.

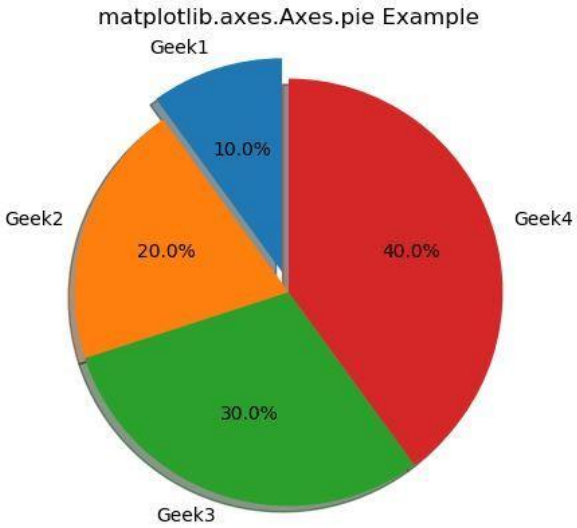
- **pctdistance:** Este parâmetro é a relação entre o centro de cada fatia da pizza e o início do texto gerado pelo autopct.
- **sombra:** este parâmetro é usado para desenhar uma sombra abaixo da pizza.
- **labeldistance:** Este parâmetro é a distância radial na qual os rótulos de pizza são desenhados.
- **startangle:** Este parâmetro é usado para girar o início do gráfico de pizza em graus de ângulo no sentido anti-horário a partir do eixo x.
- **raio:** este parâmetro é o raio da pizza.
- **counterclock:** Este parâmetro especifica a direção das frações, no sentido horário ou anti-horário.
- **wedgeprops:** este parâmetro é o ditado de argumentos passados para os objetos de cunha que fazem a pizza.
- **textprops:** Este parâmetro é um ditado de argumentos a serem passados para os objetos de texto.
- **Centro:** este parâmetro é a posição central do gráfico.
- **quadro, Armação:** Este parâmetro é usado para traçar o frame dos eixos com o gráfico, se verdadeiro.
- **rotatelabels:** Este parâmetro é usado para girar cada rótulo para o ângulo da fatia correspondente, se verdadeiro.

Retorna:

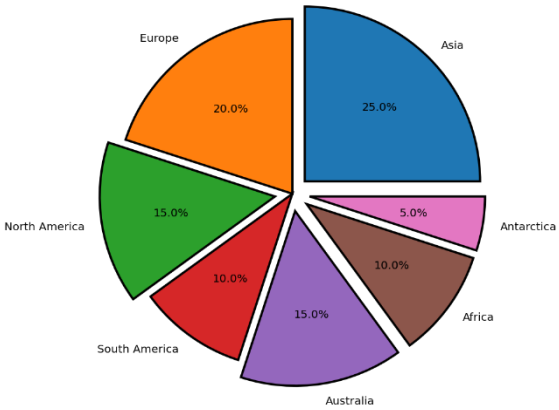
- **patches:** Isso retorna a sequência de instâncias matplotlib.patches.Wedge.
- **textos:** retorna a lista das instâncias do rótulo matplotlib.text.Text.
- **autotexts:** retorna a lista de ocorrências de texto para os rótulos numéricos.

Os exemplos abaixo ilustram a função matplotlib.axes.Axes.pie() em matplotlib.axes:

```
import matplotlib.pyplot as plt

labels = 'Geek1', 'Geek2', 'Geek3', 'Geek4'
sizes = [10, 20, 30, 40]
explode = (0.1, 0, 0, 0)

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode = explode,
          labels = labels, autopct ='% 1.1f %%',
          shadow = True, startangle = 90)
ax1.axis('equal')

ax1.set_title('matplotlib.axes.Axes.pie Example')
plt.show()
```

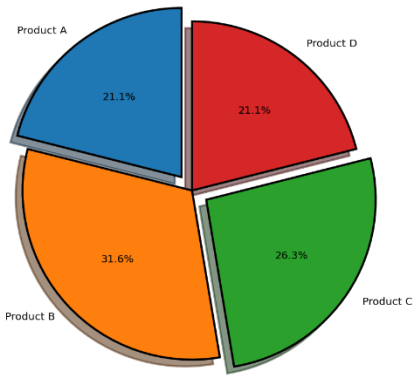



```
import matplotlib.pyplot as plt
continents = ['Asia', 'Europe', 'North America',
                    'South America','Australia',
                    'Africa','Antarctica']

area = [25, 20, 15, 10,15,10,5]
explode = (0.1, 0, 0.1, 0,0.1,0.1,0.1)

plt.pie(area, explode = explode, labels = continents,
            autopct = '%1.1f%%',startangle = 0,
            wedgeprops = {"edgecolor" : "black",
                                'linewidth' : 2,
                                'antialiased': True})
plt.axis('equal')

plt.show()
```




```
import matplotlib.pyplot as plt
sales = ['Product A', 'Product B',
```

```
             'Product C', 'Product D']

profit = [20, 30, 25, 20]
explode = (0.1, 0, 0.1, 0)

plt.pie(profit, explode = explode, labels = sales,
            autopct = '%1.1f%%',shadow = True,
            startangle = 90,
            wedgeprops = {"edgecolor":"black",
                          'linewidth': 2,
                          'antialiased': True})
plt.axis('equal')

plt.show()
```



Plots 3D

```
import matplotlib.pyplot as plt
   # Creating the figure object
   fig = plt.figure()
   # keeping the projection = 3d
   # ctreates the 3d plot
   ax = plt.axes(projection = '3d')
```

O código acima permite a criação de um gráfico 3D no Matplotlib. Podemos criar diferentes tipos de gráficos 3D, como gráficos de dispersão, gráficos de contorno, gráficos de superfície, etc. Vamos criar um gráfico de linha 3D simples.

```
import matplotlib.pyplot as plt
   x = [1, 2, 3, 4, 5]
   y = [1, 4, 9, 16, 25]
   z = [1, 8, 27, 64, 125]
   # Creating the figure object
   fig = plt.figure()
   # keeping the projection = 3d
   # ctreates the 3d plot
   ax = plt.axes(projection = '3d')
   ax.plot3D(z, y, x)
```