Matplotlib - Guia rápido

Matplotlib - Introdução

Matplotlib é um dos pacotes Python mais populares usados para visualização de dados. É uma biblioteca multiplataforma para criar gráficos 2D a partir de dados em arrays. Matplotlib é escrito em Python e faz uso de NumPy, a extensão matemática numérica do Python. Ele fornece uma API orientada a objetos que ajuda a incorporar gráficos em aplicativos que usam kits de ferramentas Python GUI, como PyQt, WxPythonotTkinter. Ele pode ser usado em shells Python e IPython, notebook Jupyter e servidores de aplicativos da web também.

Matplotlib possui uma interface processual chamada Pylab, que foi projetada para se assemelhar ao MATLAB, uma linguagem de programação proprietária desenvolvida pela MathWorks. Matplotlib junto com NumPy podem ser considerados o equivalente de código aberto do MATLAB.

Matplotlib foi originalmente escrito por John D. Hunter em 2003. A versão estável atual é 2.2.0 lançada em janeiro de 2018.

Matplotlib - Configuração do ambiente

Matplotlib e seus pacotes de dependência estão disponíveis na forma de pacotes wheel nos repositórios de pacotes Python padrão e podem ser instalados em sistemas Windows, Linux e MacOS usando o gerenciador de pacotes pip.

```
pip3 install matplotlib
```

Caso as versões Python 2.7 ou 3.4 não estejam instaladas para todos os usuários, os pacotes redistribuíveis Microsoft Visual C++ 2008 (64 bits ou 32 bits para Python 2.7) ou Microsoft Visual C++ 2010 (64 bits ou 32 bits para Python 3.4) precisam ser instalados.

Se você estiver usando Python 2.7 em um Mac, execute o seguinte comando -

```
xcode-select –install
```

Após a execução do comando acima, o subprocess32 - uma dependência, pode ser compilado.

Em versões extremamente antigas do Linux e Python 2.7, pode ser necessário instalar a versão master do subprocess32.

Matplotlib requer um grande número de dependências -

- Python (>= 2,7 ou >= 3,4)

- NumPy
- ferramentas de configuração
- datautil
- análise de py
- libpng
- pytz
- Tipo Livre
- ciclador
- seis

Opcionalmente, você também pode instalar vários pacotes para ativar kits de ferramentas de interface de usuário melhores.

- obrigado
- PyQt4
- PyQt5
- pygtk
- wxpython
- pycairo
- Tornado

Para melhor suporte ao formato de saída de animação e formatos de arquivo de imagem, LaTeX, etc., você pode instalar o seguinte -

- _mpeg/avconv
- ImagemMagick
- Travesseiro (>=2,0)
- LaTeX e GhostScript (para renderizar texto com LaTeX).
- LaTeX e GhostScript (para renderizar texto com LaTeX).

Matplotlib - distribuição Anaconda

Anaconda é uma distribuição gratuita e de código aberto das linguagens de programação Python e R para processamento de dados em larga escala, análise preditiva e computação científica. A distribuição torna o gerenciamento e a implantação de pacotes simples e fáceis. Matplotlib e muitas outras ferramentas úteis de ciência (dados) fazem parte da distribuição. As versões dos pacotes são gerenciadas pelo sistema de gerenciamento de pacotes Conda. A vantagem do Anaconda é que você tem acesso a mais de 720 pacotes que podem ser

facilmente instalados com o Conda do Anaconda, um gerenciador de pacotes, dependências e ambiente.

A distribuição Anaconda está disponível para instalação em https://www.anaconda.com/download/. Para instalação no Windows, estão disponíveis binários de 32 e 64 bits -

https://repo.continuum.io/archive/Anaconda3-5.1.0-Windows-x86.exe

https://repo.continuum.io/archive/Anaconda3-5.1.0-Windows-x86_64.exe

A instalação é um processo bastante simples baseado em assistente. Você pode escolher entre adicionar o Anaconda na variável PATH e registrar o Anaconda como seu Python padrão.

Para instalação no Linux, baixe instaladores para instaladores de 32 e 64 bits na página de downloads -

https://repo.continuum.io/archive/Anaconda3-5.1.0-Linux-x86.sh

https://repo.continuum.io/archive/Anaconda3-5.1.0-Linux-x86_64.sh

Agora, execute o seguinte comando no terminal Linux -

```
$ bash Anaconda3-5.0.1-Linux-x86_64.sh
```

Canopy e ActiveState são as opções mais procuradas para Windows, macOS e plataformas Linux comuns. Os usuários do Windows podem encontrar uma opção no WinPython.

Matplotlib - Caderno Jupyter

Jupyter é um acrônimo vago que significa Julia, Python e R. Essas linguagens de programação foram as primeiras linguagens alvo do aplicativo Jupyter, mas hoje em dia, a tecnologia de notebook também oferece suporte a muitas outras linguagens.

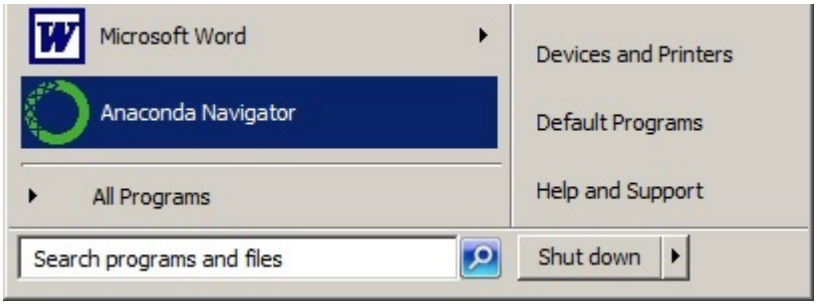
Em 2001, Fernando Pérez começou a desenvolver o Ipython. **IPython** é um shell de comando para computação interativa em múltiplas linguagens de programação, originalmente desenvolvido para Python.

Considere os seguintes recursos fornecidos pelo IPython -

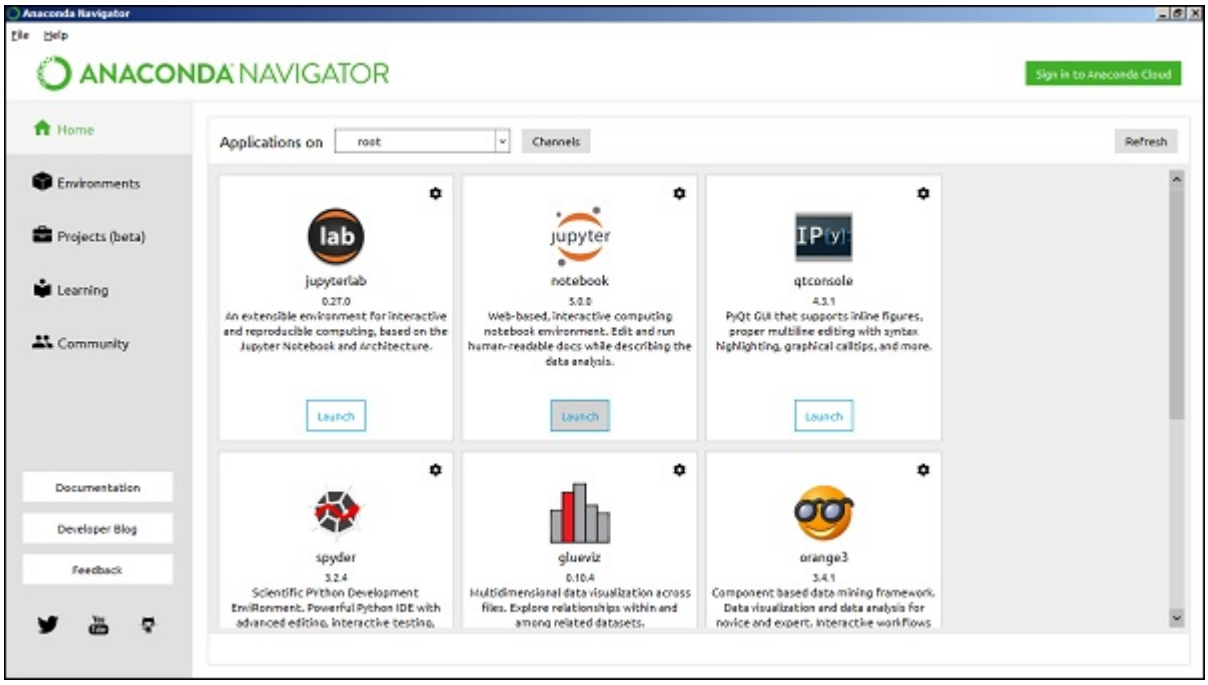
- Shells interativos (terminal e baseados em Qt).
- Um notebook baseado em navegador com suporte para código, texto, expressões matemáticas, gráficos embutidos e outras mídias.
- Suporte para visualização interativa de dados e uso de kits de ferramentas GUI.
- Intérpretes flexíveis e incorporáveis para carregar em seus próprios projetos.

Em 2014, Fernando Pérez anunciou um projeto spin-off do IPython chamado Projeto Jupyter. O IPython continuará a existir como um shell Python e um kernel para Jupyter, enquanto o notebook e outras partes independentes de linguagem do IPython passarão sob o nome Jupyter. Jupyter adicionou suporte para Julia, R, Haskell e Ruby.

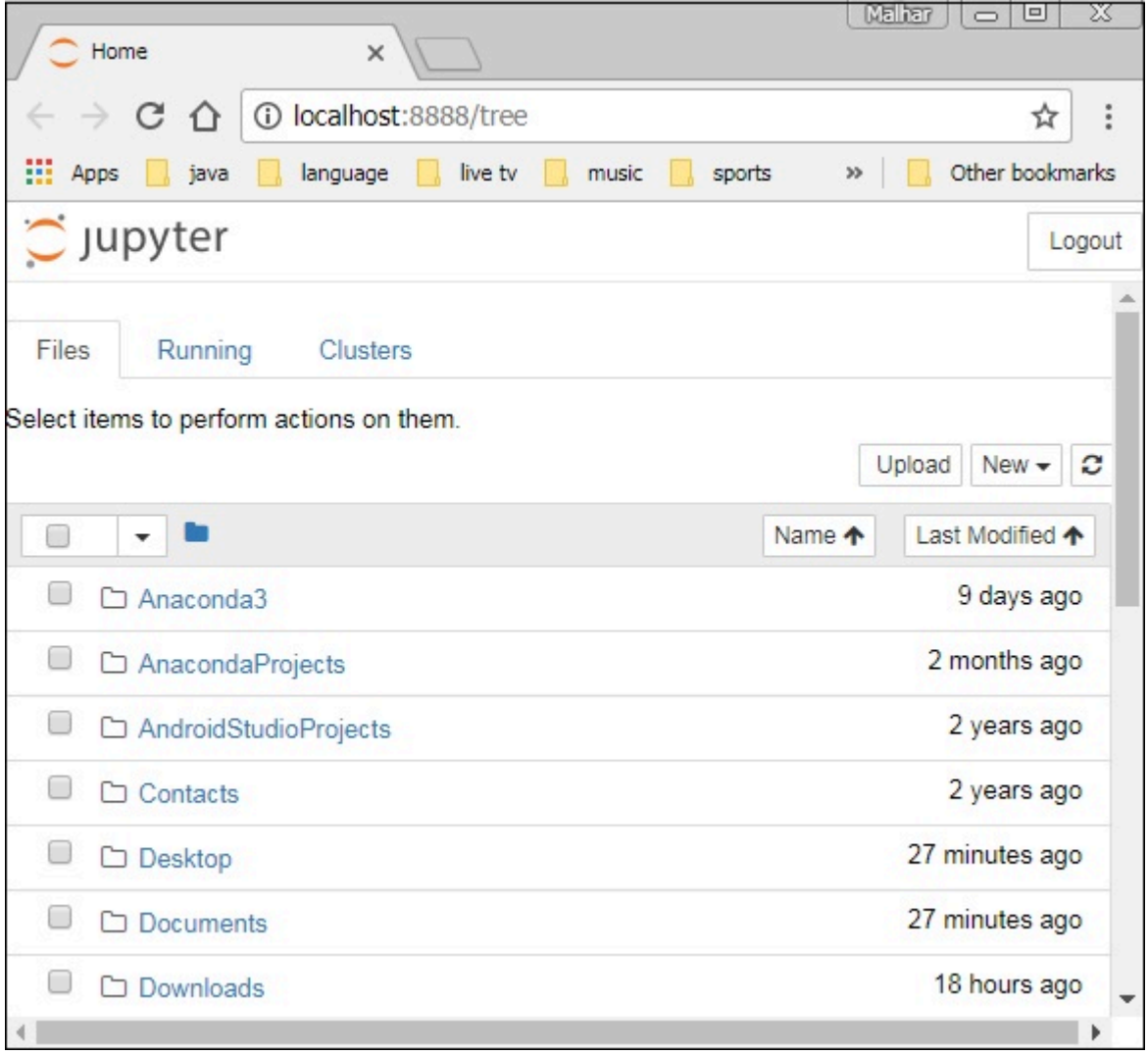
Para iniciar o notebook Jupyter, abra o navegador Anaconda (uma interface gráfica de usuário de desktop incluída no Anaconda que permite iniciar aplicativos e gerenciar facilmente pacotes, ambientes e canais Conda sem a necessidade de usar comandos de linha de comando).



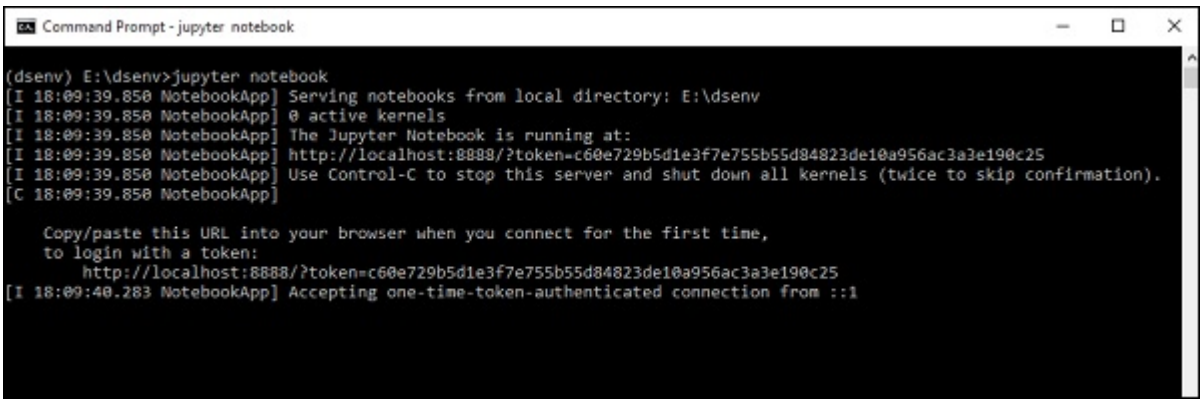
O Navigator exibe os componentes instalados na distribuição.



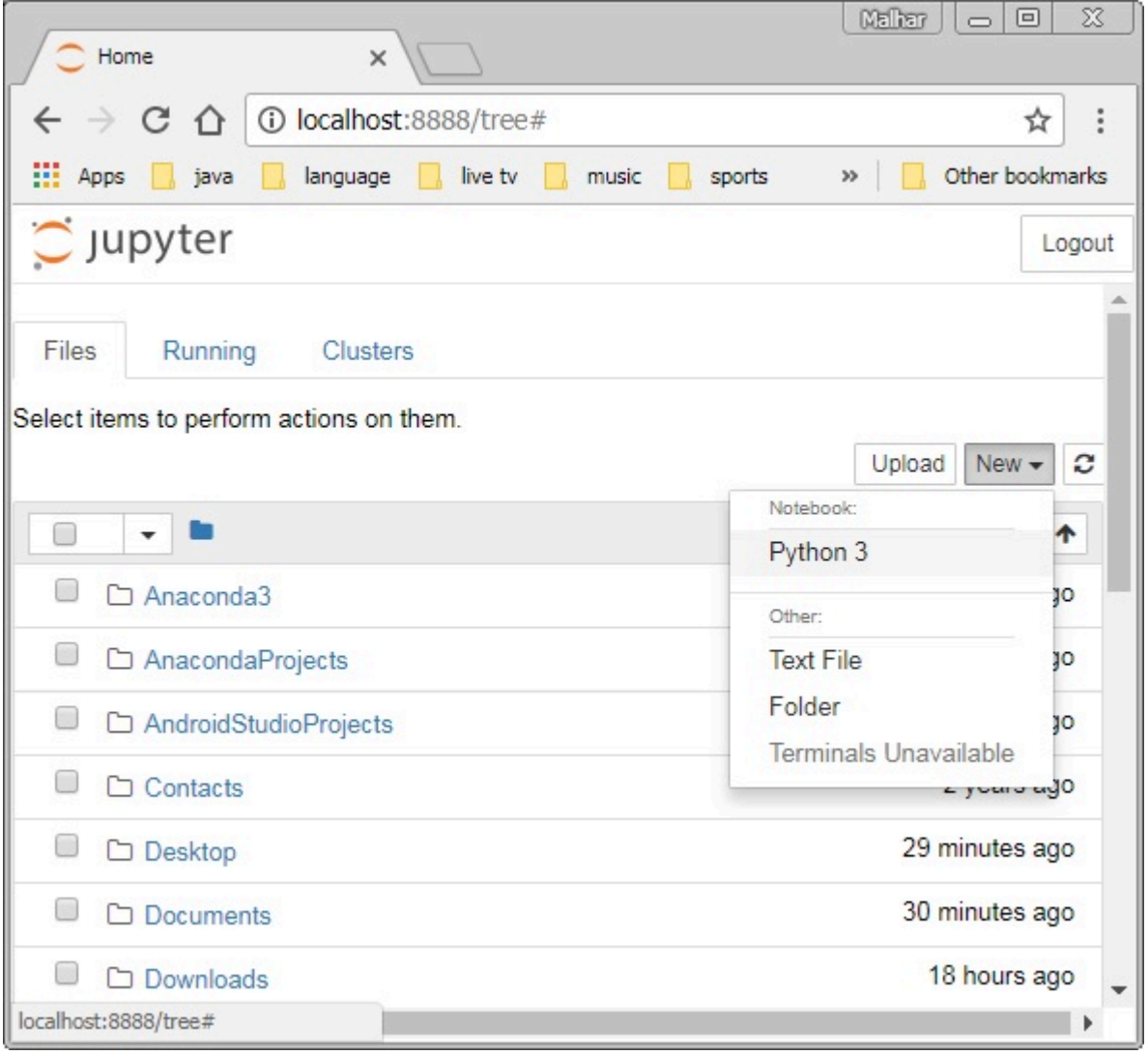
Inicie o Jupyter Notebook a partir do Navigator -



Você verá o aplicativo abrindo no navegador no seguinte endereço - **http://localhost:8888.**

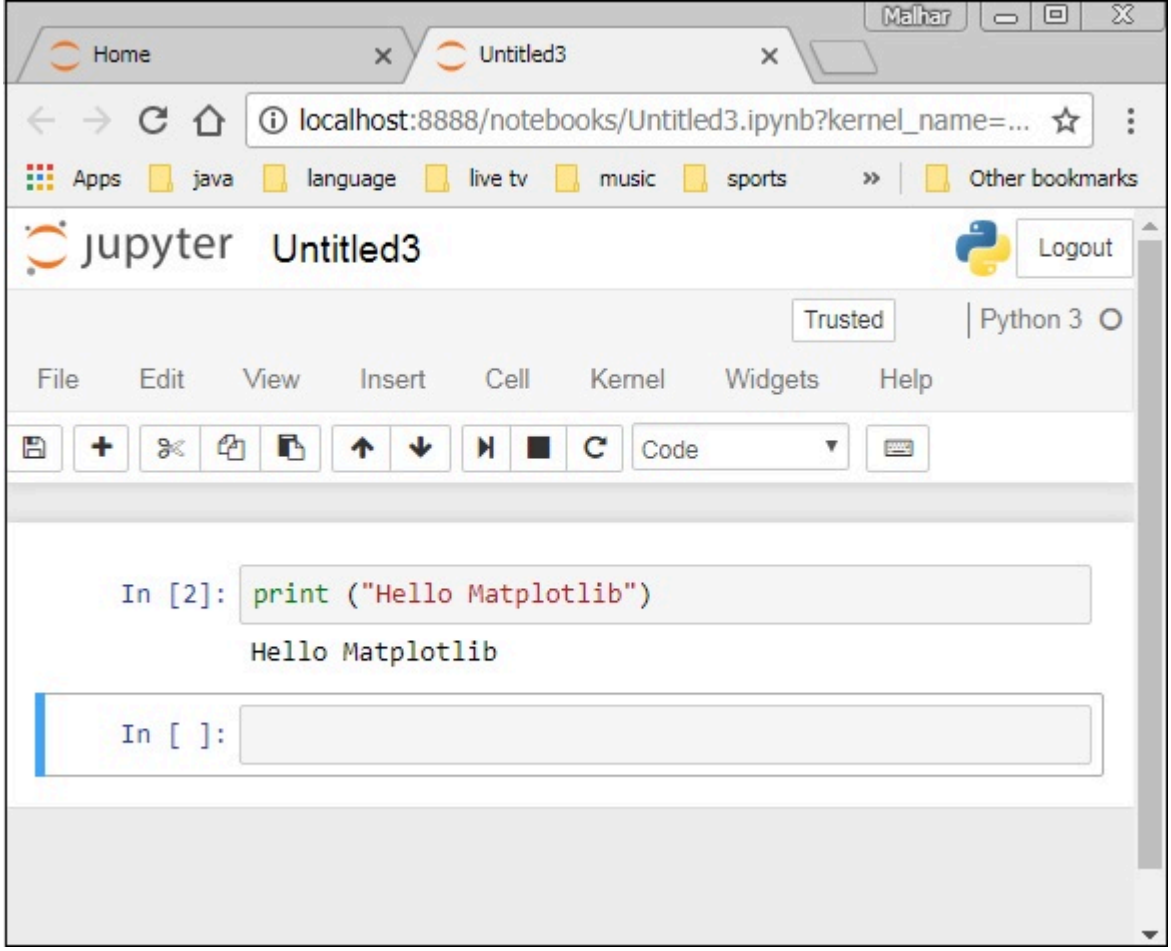


Você provavelmente deseja começar fazendo um novo caderno. Você pode fazer isso facilmente clicando no "botão Novo" na "guia Arquivos". Você vê que tem a opção de criar um arquivo de texto normal, uma pasta e um terminal. Por último, você também verá a opção de fazer um notebook Python 3.



Matplotlib - API Pyplot

Um novo notebook sem título com a extensão **.ipynb** (significa notebook IPython) é exibido na nova aba do navegador.



matplotlib.pyplot é uma coleção de funções de estilo de comando que fazem o Matplotlib funcionar como o MATLAB. Cada função Pyplot faz alguma alteração em uma figura. Por exemplo, uma função cria uma figura, uma área de plotagem em uma figura, plota algumas linhas em uma área de plotagem, decora a plotagem com rótulos, etc.

Tipos de parcelas

Sr. Não	Descrição da função
1	**Bar** Faça um gráfico de barras.
2	**Barh** Faça um gráfico de barras horizontais.
3	**Gráfico de caixa** Faça um gráfico de caixa e bigode.
4	**História** Trace um histograma.
5	**histórico2d** Faça um gráfico de histograma 2D.
6	**Torta**

	Trace um gráfico de pizza.
7	**Trama** Trace linhas e/ou marcadores nos eixos.
8	**Polar** Faça um gráfico polar..
9	**Dispersão** Faça um gráfico de dispersão de x vs y.
10	**Gráfico de pilha** Desenha um gráfico de área empilhada.
11	**Tronco** Crie um gráfico de caule.
12	**Etapa** Faça um gráfico de etapas.
13	**Tremor** Trace um campo 2-D de setas.

Funções de imagem

Sr. Não	Descrição da função
1	**Eu estou lendo** Leia uma imagem de um arquivo em um array.
2	**Salvar** Salve um array como no arquivo de imagem.
3	**Imshow** Exiba uma imagem nos eixos.

Funções do eixo

Sr. Não	Descrição da função
1	**Eixos** Adicione eixos à figura.
2	**Texto** Adicione texto aos eixos.

3	**Título** Defina um título para os eixos atuais.
4	**Xlabel** Defina o rótulo do eixo x do eixo atual.
5	**Xlim** Obtenha ou defina os limites x dos eixos atuais.
6	**Escala X** .
7	**Xtiques** Obtenha ou defina os limites x dos locais e rótulos atuais dos ticks.
8	**Ylabel** Defina o rótulo do eixo y do eixo atual.
9	**Ylim** Obtenha ou defina os limites y dos eixos atuais.
10	**Escala Y** Defina a escala do eixo y.
11	**Carrapatos** Obtenha ou defina os limites y dos locais e rótulos atuais dos ticks.

Funções de Figura

Sr. Não	Descrição da função
1	**Texto da figura** Adicione texto à figura.
2	**Figura** Cria uma nova figura.
3	**Mostrar** Mostre uma figura.
4	**Salvar fig** Salve a figura atual.
5	**Fechar** Feche uma janela de figura.

Matplotlib - Gráfico Simples

Neste capítulo, aprenderemos como criar um gráfico simples com Matplotlib.

Vamos agora exibir um gráfico de linha simples de ângulo em radianos versus seu valor seno no Matplotlib. Para começar, o módulo Pyplot do pacote Matplotlib é importado, com um alias plt por convenção.

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

Em seguida, precisamos de uma matriz de números para representar graficamente. Várias funções de array são definidas na biblioteca NumPy que é importada com o alias np.

```
import numpy as np
```

Agora obtemos o objeto ndarray de ângulos entre 0 e 2π usando a função arange() da biblioteca NumPy.

```
x = np.arange(0, math.pi*2, 0.05)
```

O objeto ndarray serve como valores no eixo x do gráfico. Os valores de seno correspondentes dos ângulos em x a serem exibidos no eixo y são obtidos pela seguinte afirmação -

```
y = np.sin(x)
```

Os valores de duas matrizes são plotados usando a função plot().

```
plt.plot(x,y)
```

Você pode definir o título do gráfico e os rótulos dos eixos x e y.

```
You can set the plot title, and labels for x and y axes.
plt.xlabel("angle")
plt.ylabel("sine")
plt.title('sine wave')
```

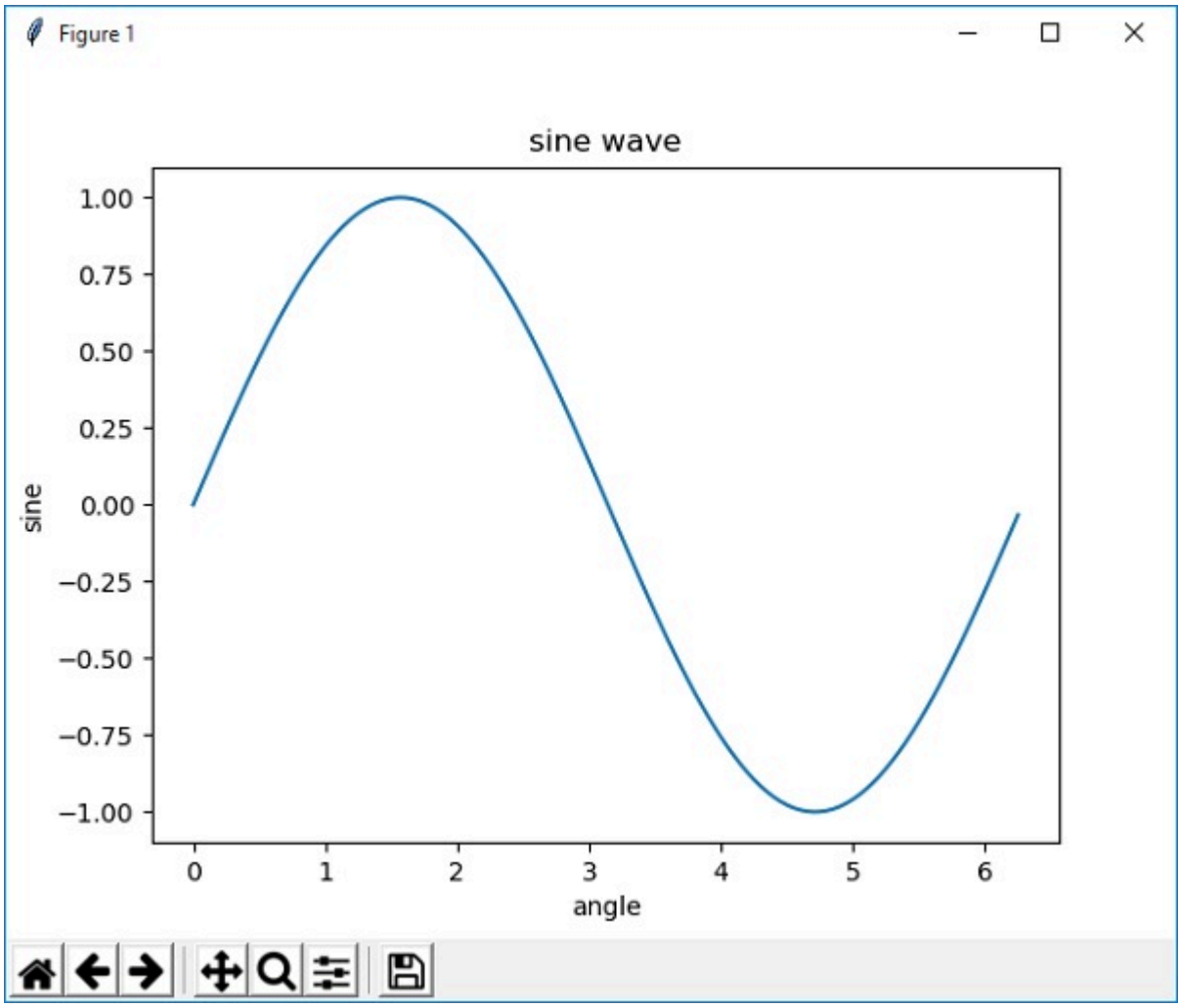
A janela do visualizador Plot é invocada pela função show() -

```
plt.show()
```

O programa completo é o seguinte -

```
from matplotlib import pyplot as plt
import numpy as np
import math #needed for definition of pi
x = np.arange(0, math.pi*2, 0.05)
y = np.sin(x)
plt.plot(x,y)
plt.xlabel("angle")
plt.ylabel("sine")
plt.title('sine wave')
plt.show()
```

Quando a linha de código acima é executada, o seguinte gráfico é exibido -



Agora, use o notebook Jupyter com Matplotlib.

Inicie o notebook Jupyter no navegador Anaconda ou na linha de comando conforme descrito anteriormente. Na célula de entrada, insira instruções de importação para Pyplot e NumPy -

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

Para exibir os resultados do gráfico dentro do próprio notebook (e não no visualizador separado), insira a seguinte instrução mágica -

```
%matplotlib inline
```

Obtenha x como o objeto ndarray contendo ângulos em radianos entre 0 e 2π e y como o valor do seno de cada ângulo -

```
import math
x = np.arange(0, math.pi*2, 0.05)
y = np.sin(x)
```

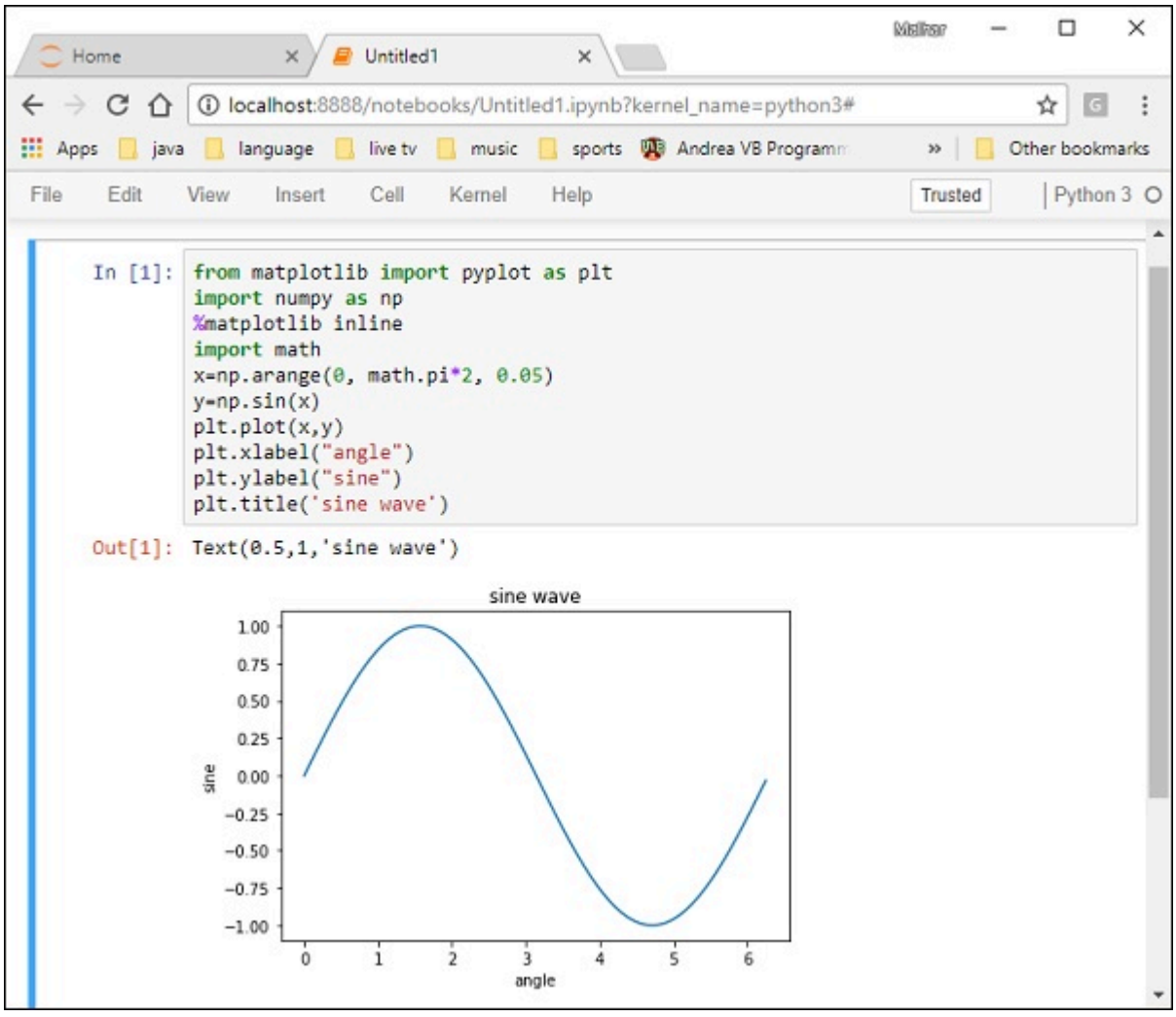
Defina rótulos para os eixos xey, bem como o título do gráfico -

```
plt.xlabel("angle")
plt.ylabel("sine")
plt.title('sine wave')
```

Por fim, execute a função plot() para gerar a exibição da onda senoidal no notebook (não há necessidade de executar a função show()) -

```
plt.plot(x,y)
```

Após a execução da linha final do código, a seguinte saída é exibida -



Matplotlib - módulo PyLab

PyLab é uma interface processual para a biblioteca de plotagem orientada a objetos Matplotlib. Matplotlib é o pacote completo; matplotlib.pyplot é um módulo do Matplotlib; e PyLab é um módulo instalado junto com o Matplotlib.

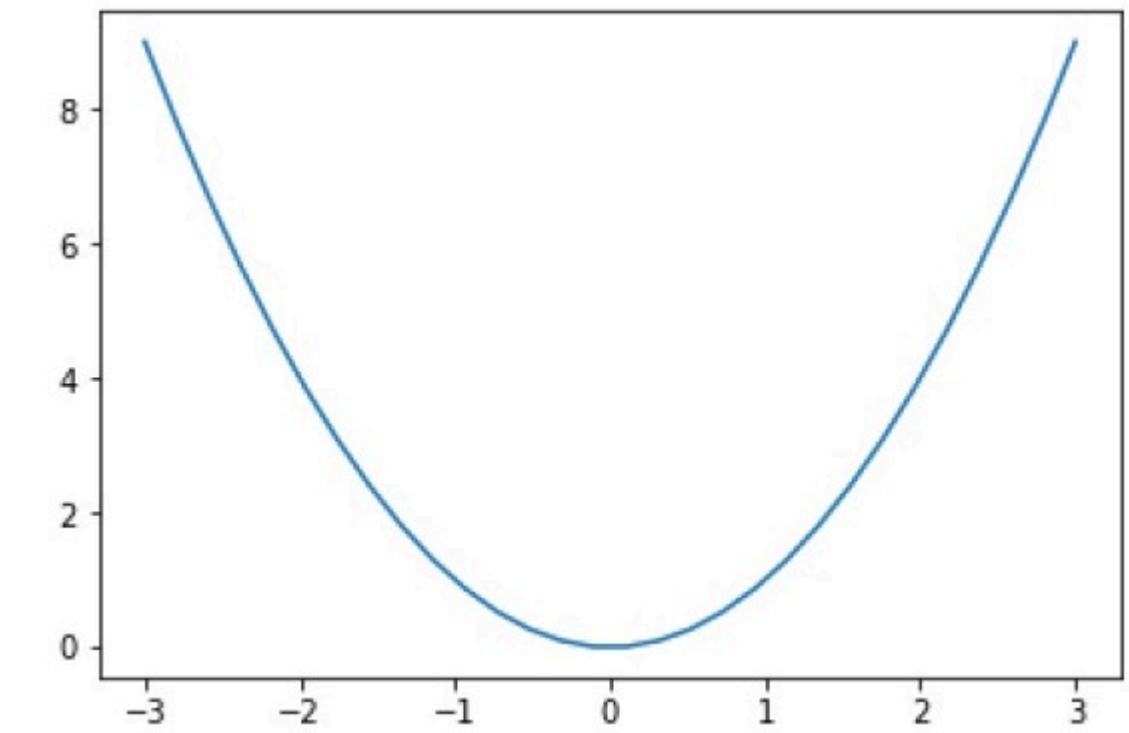
PyLab é um módulo de conveniência que importa em massa matplotlib.pyplot (para plotagem) e NumPy (para matemática e trabalho com matrizes) em um único espaço de nomes. Embora muitos exemplos usem PyLab, ele não é mais recomendado.

Plotagem Básica

A plotagem de curvas é feita com o comando plot. É necessário um par de matrizes (ou sequências) do mesmo comprimento -

```
from numpy import *
from pylab import *
x = linspace(-3, 3, 30)
y = x**2
plot(x, y)
show()
```

A linha de código acima gera a seguinte saída -



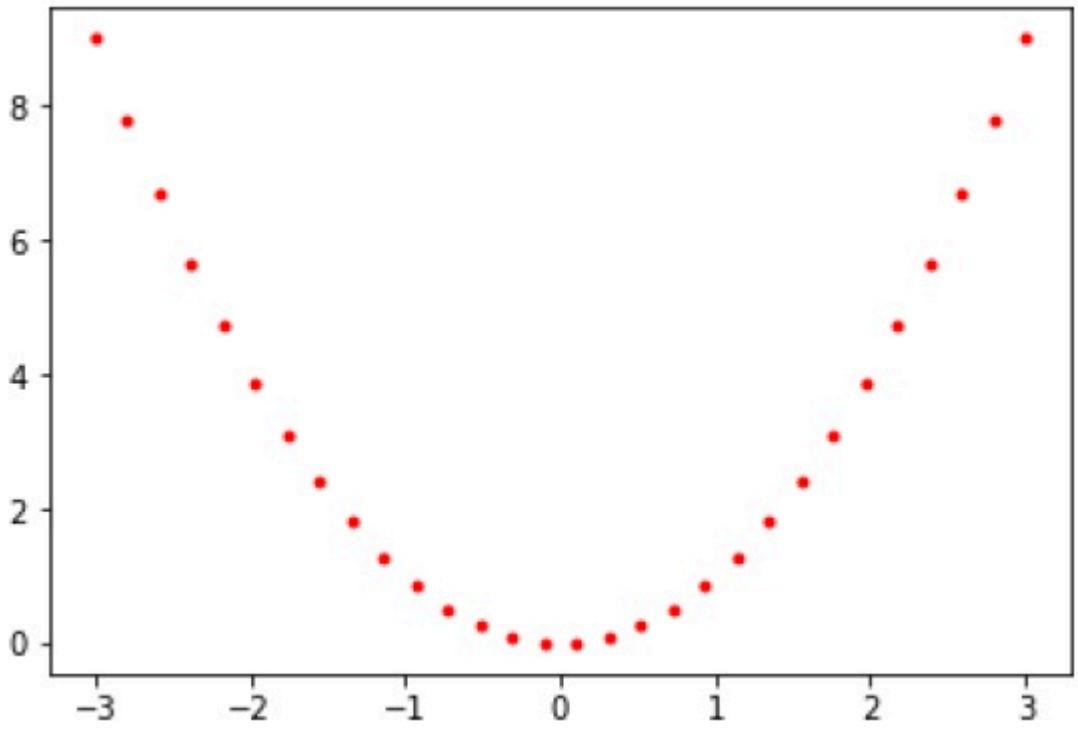
To plot symbols rather than lines, provide an additional string argument.

symbols	- , –, -., , . , , , , o , ^ , v , < , > , s , + , x , D , d , 1 , 2 , 3 , 4 , h , H , p , | , _
colors	b, g, r, c, m, y, k, w

Now, consider executing the following code –

```
from pylab import *
x = linspace(-3, 3, 30)
y = x**2
plot(x, y, 'r.')
show()
```

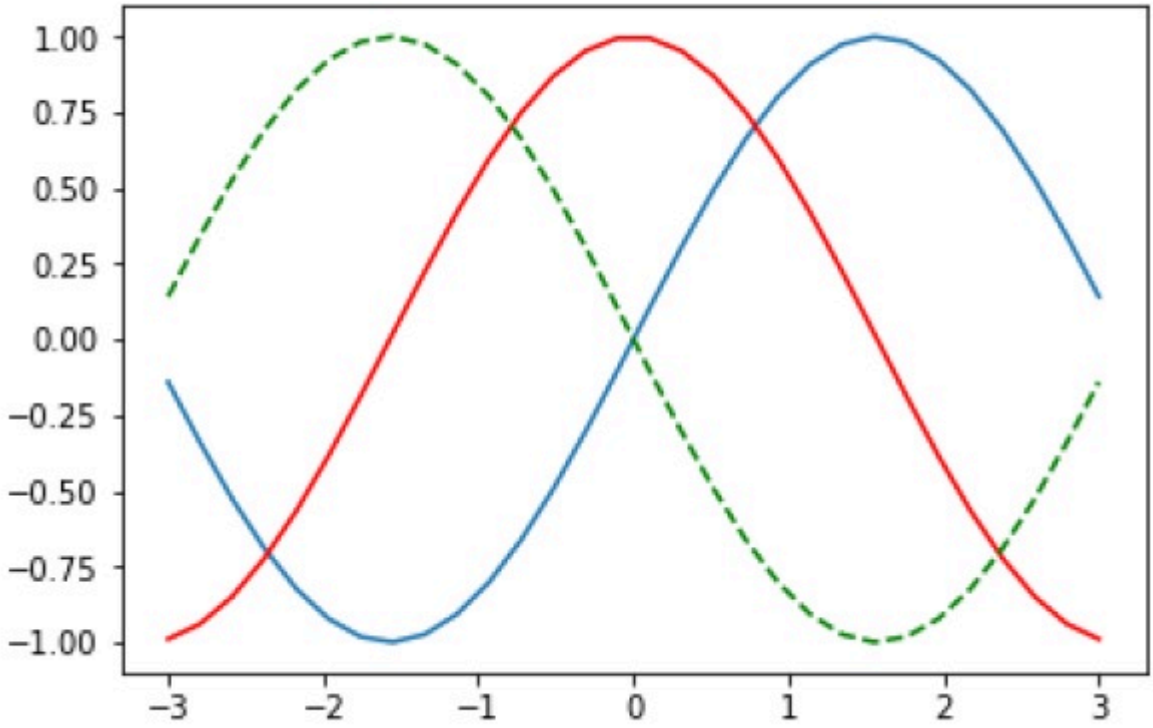
It plots the red dots as shown below –



Plots can be overlaid. Just use the multiple plot commands. Use clf() to clear the plot.

```
from pylab import *
plot(x, sin(x))
plot(x, cos(x), 'r-')
plot(x, -sin(x), 'g--')
show()
```

The above line of code generates the following output –



Matplotlib - Object-oriented Interface

While it is easy to quickly generate plots with the **matplotlib.pyplot** module, the use of object-oriented approach is recommended as it gives more control and customization of your plots. Most of the functions are also available in the **matplotlib.axes.Axes** class.

The main idea behind using the more formal object-oriented method is to create figure objects and then just call methods or attributes off of that object. This approach helps better in dealing with a canvas that has multiple plots on it.

In object-oriented interface, Pyplot is used only for a few functions such as figure creation, and the user explicitly creates and keeps track of the figure and axes objects. At this level, the user uses Pyplot to create figures, and through those figures, one or more axes objects can be created. These axes objects are then used for most plotting actions.

To begin with, we create a figure instance which provides an empty canvas.

```
fig = plt.figure()
```

Now add axes to figure. The **add_axes()** method requires a list object of 4 elements corresponding to left, bottom, width and height of the figure. Each number must be between 0 and 1 −

```
ax=fig.add_axes([0,0,1,1])
```

Set labels for x and y axis as well as title −

```
ax.set_title("sine wave")
ax.set_xlabel('angle')
ax.set_ylabel('sine')
```

Invoke the plot() method of the axes object.

```
ax.plot(x,y)
```

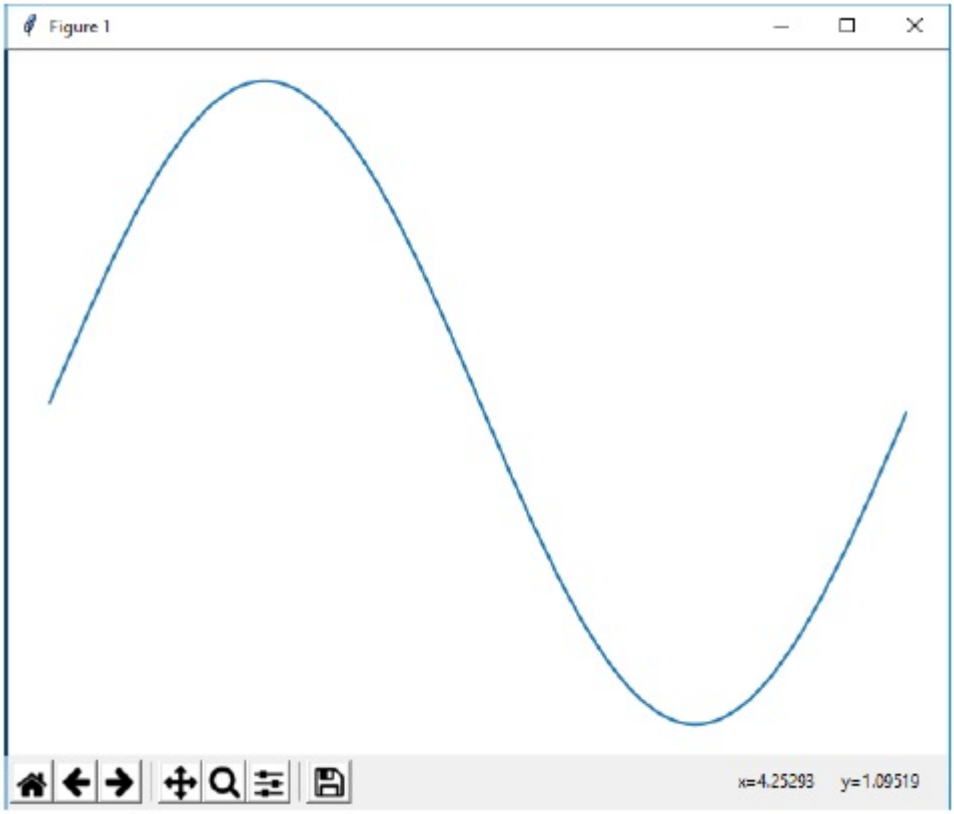
If you are using Jupyter notebook, the %matplotlib inline directive has to be issued; the otherwistshow() function of pyplot module displays the plot.

Consider executing the following code −

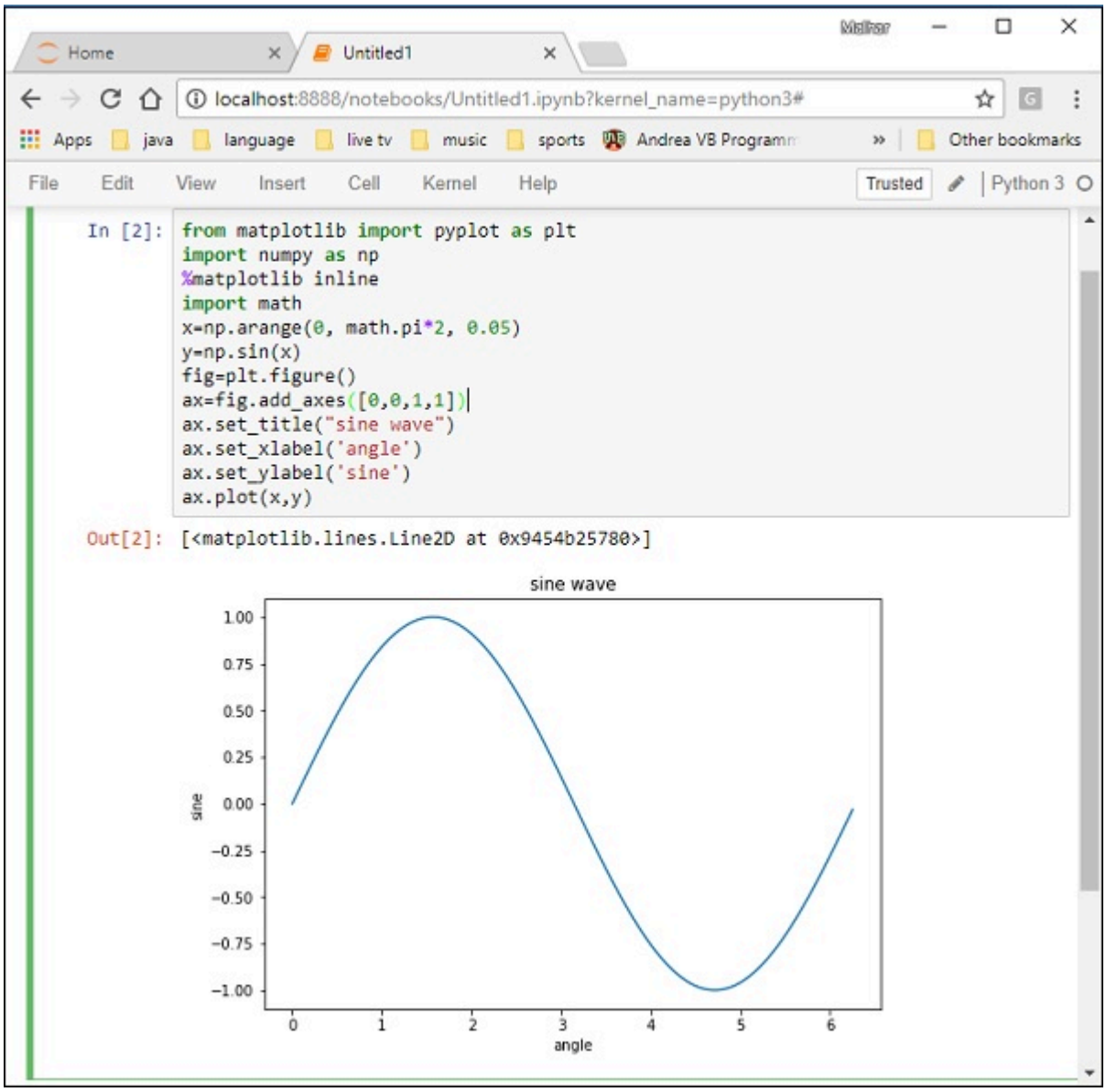
```
from matplotlib import pyplot as plt
import numpy as np
import math
x = np.arange(0, math.pi*2, 0.05)
y = np.sin(x)
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.plot(x,y)
ax.set_title("sine wave")
ax.set_xlabel('angle')
ax.set_ylabel('sine')
plt.show()
```

Output

The above line of code generates the following output −



The same code when run in Jupyter notebook shows the output as shown below –



Matplotlib - Figure Class

The **matplotlib.figure** module contains the Figure class. It is a top-level container for all plot elements. The Figure object is instantiated by calling the **figure()** function from the pyplot module –

```
fig = plt.figure()
```

The following table shows the additional parameters –

Figsize	(width,height) tuple in inches
Dpi	Dots per inches
Facecolor	Figure patch facecolor
Edgecolor	Figure patch edge color
Linewidth	Edge line width

Matplotlib - Axes Class

Axes object is the region of the image with the data space. A given figure can contain many Axes, but a given Axes object can only be in one Figure. The Axes contains two (or three in the case of 3D) Axis objects. The Axes class and its member functions are the primary entry point to working with the OO interface.

O objeto Axes é adicionado à figura chamando o método add_axes(). Ele retorna o objeto de eixos e adiciona eixos na posição ret [esquerda, inferior, largura, altura] onde todas as quantidades estão em frações da largura e altura da figura.

Parâmetro

A seguir está o parâmetro para a classe Axes -

- rect - Uma sequência de 4 comprimentos de quantidades [esquerda, inferior, largura, altura].

```
ax=fig.add_axes([0,0,1,1])
```

As seguintes funções-membro da classe axes adicionam diferentes elementos ao gráfico -

Lenda

O método **legend()** da classe axes adiciona uma legenda à figura do gráfico. São necessários três parâmetros -

```
ax.legend(handles, labels, loc)
```

Onde rótulos é uma sequência de strings e trata uma sequência de instâncias Line2D ou Patch. loc pode ser uma string ou um número inteiro especificando a localização da legenda.

Sequência de localização	**Código de localização**
Melhor	0
canto superior direito	1
canto superior esquerdo	2
canto inferior esquerdo	3
inferior direito	4
Certo	5
Centro-esquerda	6
Centro-direita	7
centro inferior	8
centro superior	9
Centro	10

eixos.plot()

Este é o método básico da classe de eixos que plota os valores de um array versus outro como linhas ou marcadores. O método plot() pode ter um argumento de string de formato opcional para especificar a cor, o estilo e o tamanho da linha e do marcador.

Códigos de cores

Personagem	Cor
'b'	Azul
'g'	Verde
'r'	Vermelho
'b'	Azul
'c'	Ciano
'eu'	Magenta
'você'	Amarelo
'k'	Preto
'b'	Azul
'c'	Branco

Códigos marcadores

Personagem	Descrição
'.'	Marcador de ponto
'o'	Marcador de círculo
'x'	Marcador X
'D'	Marcador de diamante
'H'	Marcador hexagonal
'é'	Marcador quadrado
'+'	Mais marcador

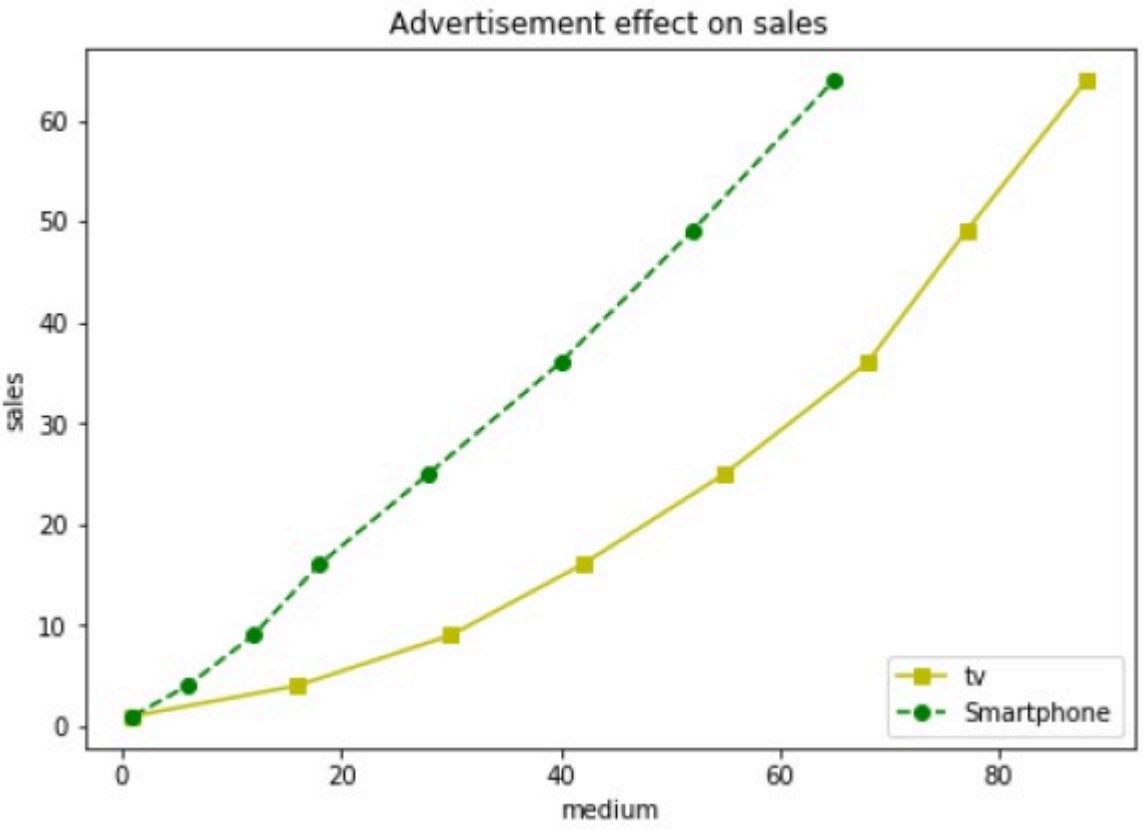
Estilos de linha

Personagem	Descrição
'-'	Linha sólida
'—'	Linha tracejada
'-.'	Linha traço-ponto
':'	Linha pontilhada
'H'	Marcador hexagonal

O exemplo a seguir mostra as despesas com publicidade e os números de vendas de TV e smartphones na forma de gráficos de linhas. A linha que representa a TV é uma linha sólida com cor amarela e marcadores quadrados, enquanto a linha do smartphone é uma linha tracejada com cor verde e marcador circular.

```
import matplotlib.pyplot as plt
y = [1, 4, 9, 16, 25,36,49, 64]
x1 = [1, 16, 30, 42,55, 68, 77,88]
x2 = [1,6,12,18,28, 40, 52, 65]
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
l1 = ax.plot(x1,y,'ys-') # solid line with yellow colour and square marker
l2 = ax.plot(x2,y,'go--') # dash line with green colour and circle marker
ax.legend(labels = ('tv', 'Smartphone'), loc = 'lower right') # legend placed at lowe
ax.set_title("Advertisement effect on sales")
ax.set_xlabel('medium')
ax.set_ylabel('sales')
plt.show()
```

Quando a linha de código acima é executada, ela produz o seguinte gráfico -



Matplotlib - Multiplots

Neste capítulo, aprenderemos como criar vários subtramas na mesma tela.

A função **subplot()** retorna o objeto dos eixos em uma determinada posição da grade. A assinatura de chamada desta função é -

```
plt.subplot(subplot(nrows, ncols, index)
```

Na figura atual, a função cria e retorna um objeto Axes, na posição índice de uma grade de nrows por ncolsaxes. Os índices vão de 1 a nrows * ncols, incrementando na ordem da linha principal. Ifnrows, ncols e índice são todos menores que 10. Os índices também podem ser fornecidos como um número único, concatenado, de três dígitos.

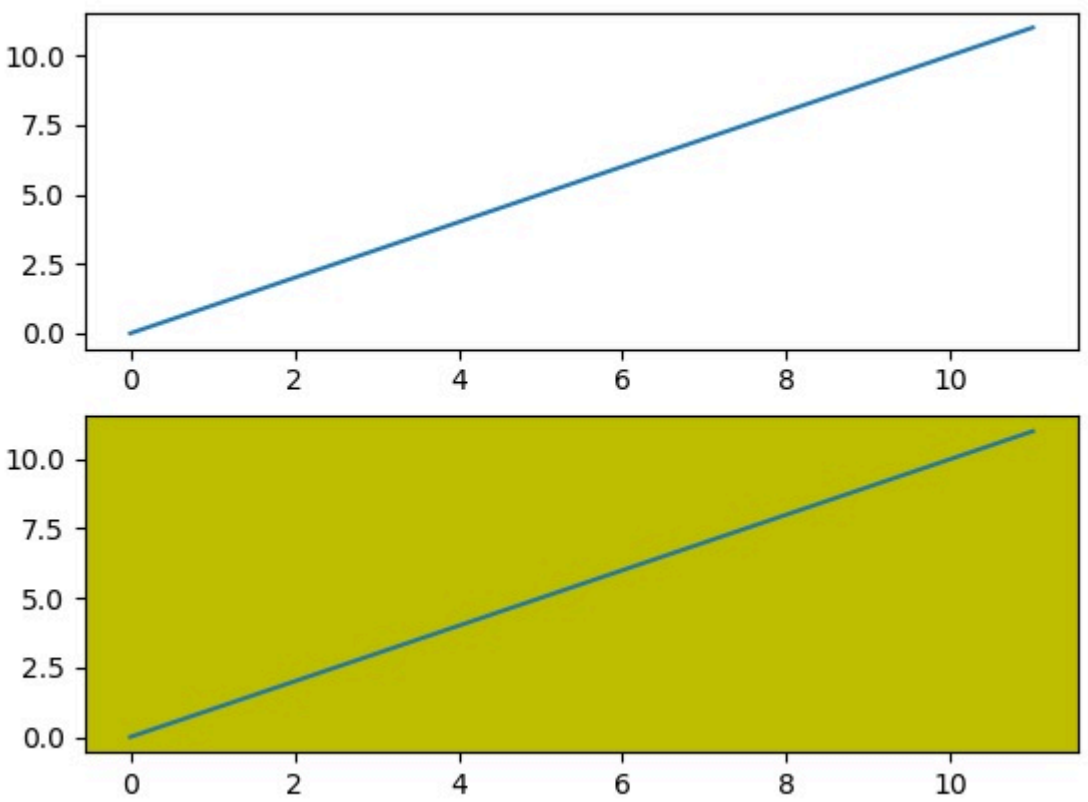
Por exemplo, subplot(2, 3, 3) e subplot(233) criam Eixos no canto superior direito da figura atual, ocupando metade da altura da figura e um terço da largura da figura.

A criação de uma subtrama excluirá qualquer subtrama pré-existente que se sobreponha a ela, além de compartilhar um limite.

```
import matplotlib.pyplot as plt
# plot a line, implicitly creating a subplot(111)
plt.plot([1,2,3])
# now create a subplot which represents the top plot of a grid with 2 rows and 1 colu
#Since this subplot will overlap the first, the plot (and its axes) previously
created, will be removed
plt.subplot(211)
```

```
plt.plot(range(12))
plt.subplot(212, facecolor='y') # creates 2nd subplot with yellow background
plt.plot(range(12))
```

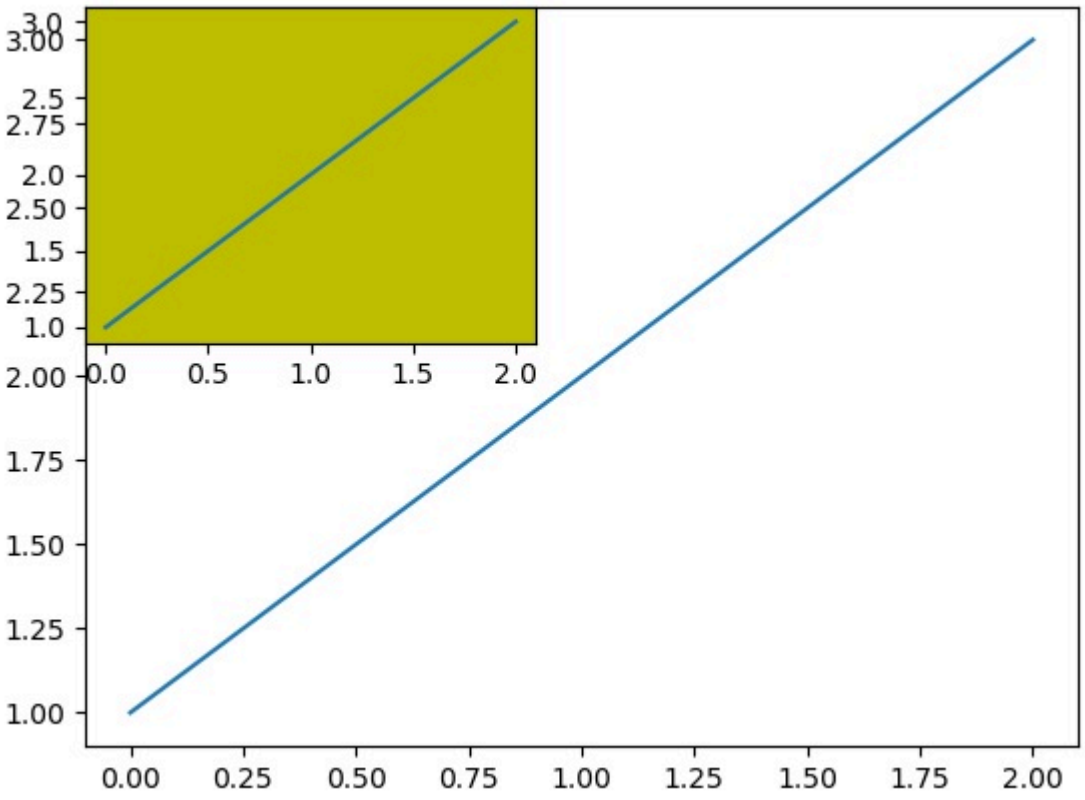
A linha de código acima gera a seguinte saída -



A função add_subplot() da classe figure não substituirá o gráfico existente -

```
import matplotlib.pyplot as plt
fig = plt.figure()
ax1 = fig.add_subplot(111)
ax1.plot([1,2,3])
ax2 = fig.add_subplot(221, facecolor='y')
ax2.plot([1,2,3])
```

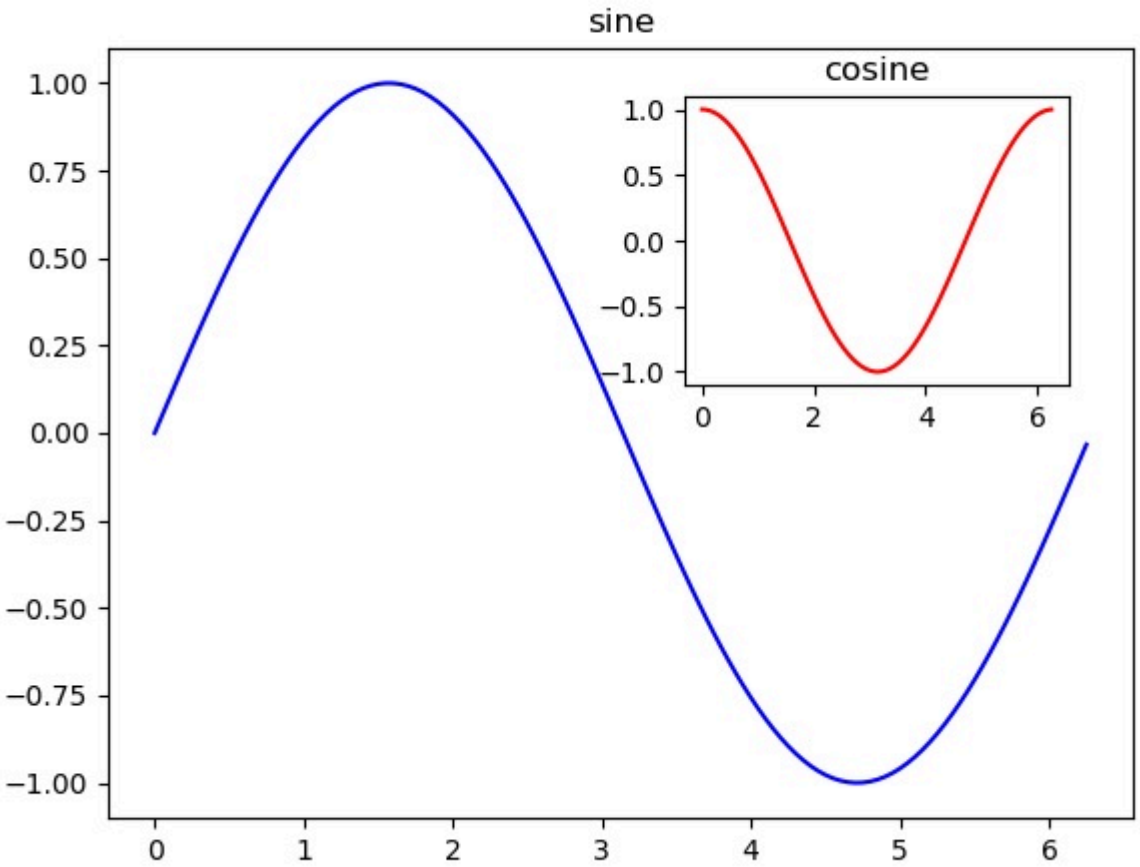
Quando a linha de código acima é executada, ela gera a seguinte saída -



Você pode adicionar um gráfico de inserção na mesma figura adicionando outro objeto de eixo na mesma tela da figura.

```
import matplotlib.pyplot as plt
import numpy as np
import math
x = np.arange(0, math.pi*2, 0.05)
fig=plt.figure()
axes1 = fig.add_axes([0.1, 0.1, 0.8, 0.8]) # main axes
axes2 = fig.add_axes([0.55, 0.55, 0.3, 0.3]) # inset axes
y = np.sin(x)
axes1.plot(x, y, 'b')
axes2.plot(x,np.cos(x),'r')
axes1.set_title('sine')
axes2.set_title("cosine")
plt.show()
```

Após a execução da linha de código acima, a seguinte saída é gerada -



Matplotlib - Função Subtramas()

A API pyplot do Matplotlib tem uma função de conveniência chamada subplots() que atua como um wrapper de utilitário e ajuda na criação de layouts comuns de subtramas, incluindo o objeto de figura envolvente, em uma única chamada.

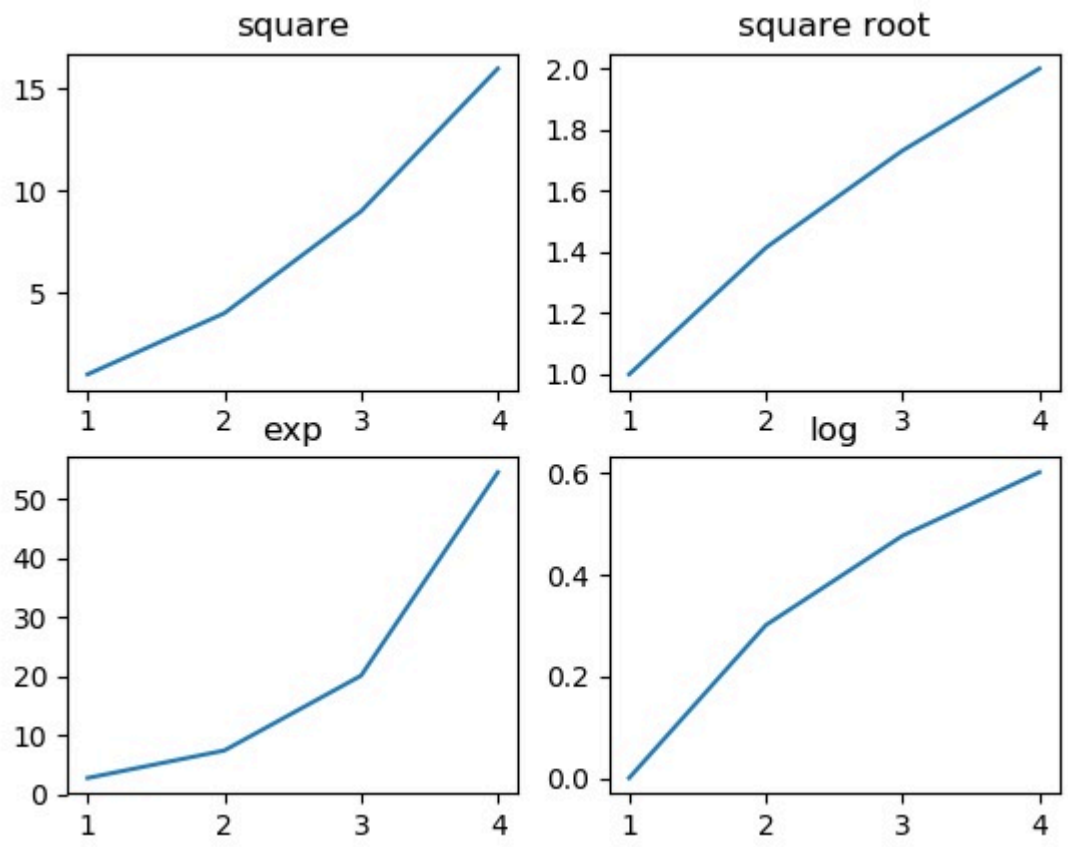
```
Plt.subplots(nrows, ncols)
```

Os dois argumentos inteiros para esta função especificam o número de linhas e colunas da grade da subtrama. A função retorna um objeto figura e uma tupla contendo objetos eixos iguais a nrows*ncols. Cada objeto de eixos é acessível por seu índice. Aqui criamos uma subparcela de 2 linhas por 2 colunas e exibimos 4 parcelas diferentes em cada subparcela.

```
import matplotlib.pyplot as plt
fig,a =  plt.subplots(2,2)
import numpy as np
x = np.arange(1,5)
a[0][0].plot(x,x*x)
a[0][0].set_title('square')
a[0][1].plot(x,np.sqrt(x))
a[0][1].set_title('square root')
a[1][0].plot(x,np.exp(x))
a[1][0].set_title('exp')
a[1][1].plot(x,np.log10(x))
```

```
a[1][1].set_title('log')
plt.show()
```

A linha de código acima gera a seguinte saída -



Matplotlib - Função Subplot2grid()

Esta função oferece mais flexibilidade na criação de um objeto de eixos em um local específico da grade. Também permite que o objeto de eixos seja estendido por várias linhas ou colunas.

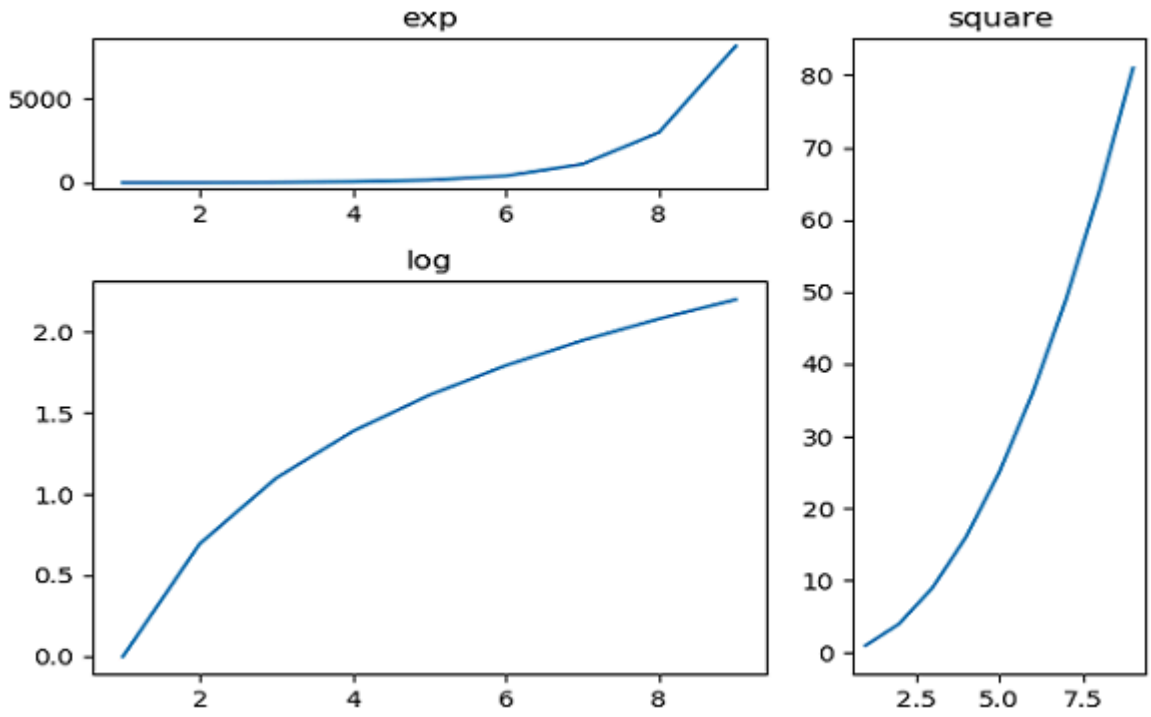
```
Plt.subplot2grid(shape, location, rowspan, colspan)
```

No exemplo a seguir, uma grade 3X3 do objeto de figura é preenchida com objetos de eixos de tamanhos variados em extensões de linha e coluna, cada um mostrando um gráfico diferente.

```
import matplotlib.pyplot as plt
a1 = plt.subplot2grid((3,3),(0,0),colspan = 2)
a2 = plt.subplot2grid((3,3),(0,2), rowspan = 3)
a3 = plt.subplot2grid((3,3),(1,0),rowspan = 2, colspan = 2)
import numpy as np
x = np.arange(1,10)
a2.plot(x, x*x)
a2.set_title('square')
a1.plot(x, np.exp(x))
a1.set_title('exp')
```

```
a3.plot(x, np.log(x))
a3.set_title('log')
plt.tight_layout()
plt.show()
```

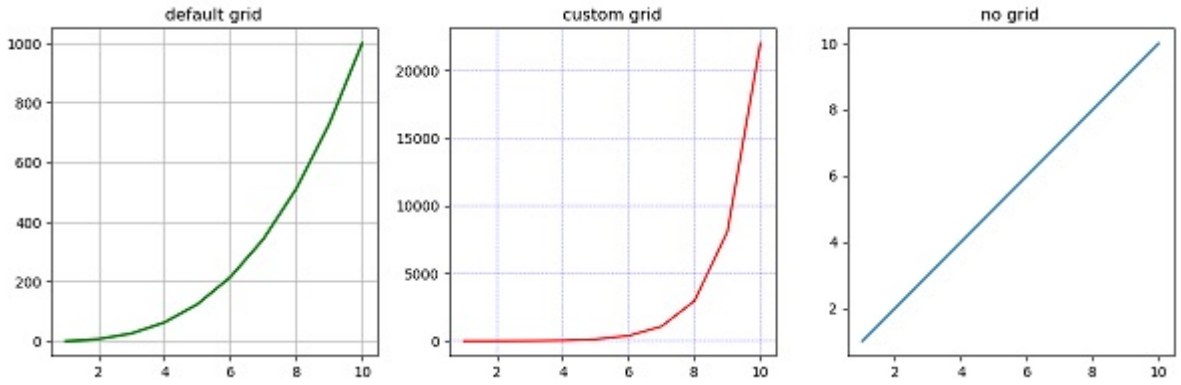
Após a execução do código de linha acima, a seguinte saída é gerada -



Matplotlib - Grades

A função grid() do objeto de eixos ativa ou desativa a visibilidade da grade dentro da figura. Você também pode exibir ticks maiores/menores (ou ambos) da grade. Além disso, as propriedades de cor, estilo de linha e largura de linha podem ser definidas na função grid().

```
import matplotlib.pyplot as plt
import numpy as np
fig, axes = plt.subplots(1,3, figsize = (12,4))
x = np.arange(1,11)
axes[0].plot(x, x**3, 'g',lw=2)
axes[0].grid(True)
axes[0].set_title('default grid')
axes[1].plot(x, np.exp(x), 'r')
axes[1].grid(color='b', ls = '-.', lw = 0.25)
axes[1].set_title('custom grid')
axes[2].plot(x,x)
axes[2].set_title('no grid')
fig.tight_layout()
plt.show()
```



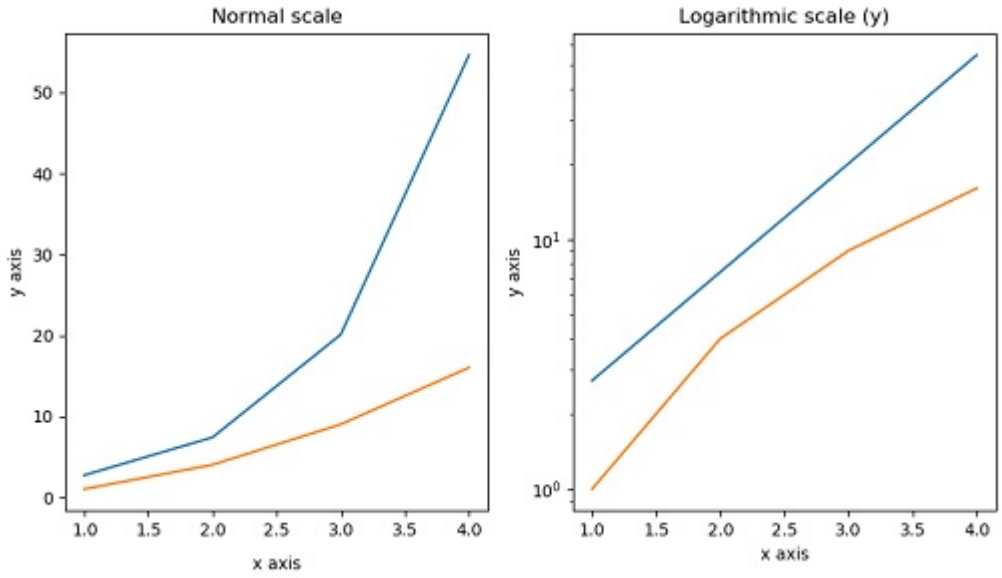
Matplotlib - Formatação de Eixos

Às vezes, um ou alguns pontos são muito maiores que a maior parte dos dados. Nesse caso, a escala de um eixo precisa ser definida como logarítmica em vez da escala normal. Esta é a escala logarítmica. No Matplotlib, é possível definir a propriedade xscale ou vscale do objeto axes como 'log'.

Às vezes, também é necessário mostrar alguma distância adicional entre os números dos eixos e o rótulo dos eixos. A propriedade labelpad de qualquer eixo (x ou y ou ambos) pode ser definida com o valor desejado.

Ambos os recursos acima são demonstrados com a ajuda do exemplo a seguir. A subtrama à direita possui escala logarítmica e a da esquerda possui eixo x com rótulo a maior distância.

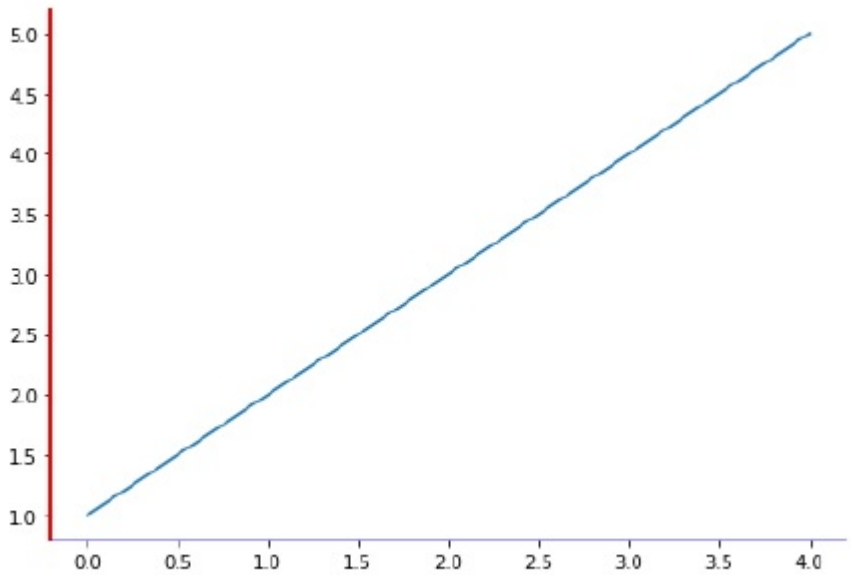
```
import matplotlib.pyplot as plt
import numpy as np
fig, axes = plt.subplots(1, 2, figsize=(10,4))
x = np.arange(1,5)
axes[0].plot( x, np.exp(x))
axes[0].plot(x,x**2)
axes[0].set_title("Normal scale")
axes[1].plot (x, np.exp(x))
axes[1].plot(x, x**2)
axes[1].set_yscale("log")
axes[1].set_title("Logarithmic scale (y)")
axes[0].set_xlabel("x axis")
axes[0].set_ylabel("y axis")
axes[0].xaxis.labelpad = 10
axes[1].set_xlabel("x axis")
axes[1].set_ylabel("y axis")
plt.show()
```



As espinhas do eixo são as linhas que conectam as marcas do eixo que demarcam os limites da área de plotagem. O objeto de eixos possui espinhos localizados na parte superior, inferior, esquerda e direita.

Cada lombada pode ser formatada especificando cor e largura. Qualquer borda pode ficar invisível se sua cor estiver definida como nenhuma.

```
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.spines['bottom'].set_color('blue')
ax.spines['left'].set_color('red')
ax.spines['left'].set_linewidth(2)
ax.spines['right'].set_color(None)
ax.spines['top'].set_color(None)
ax.plot([1,2,3,4,5])
plt.show()
```

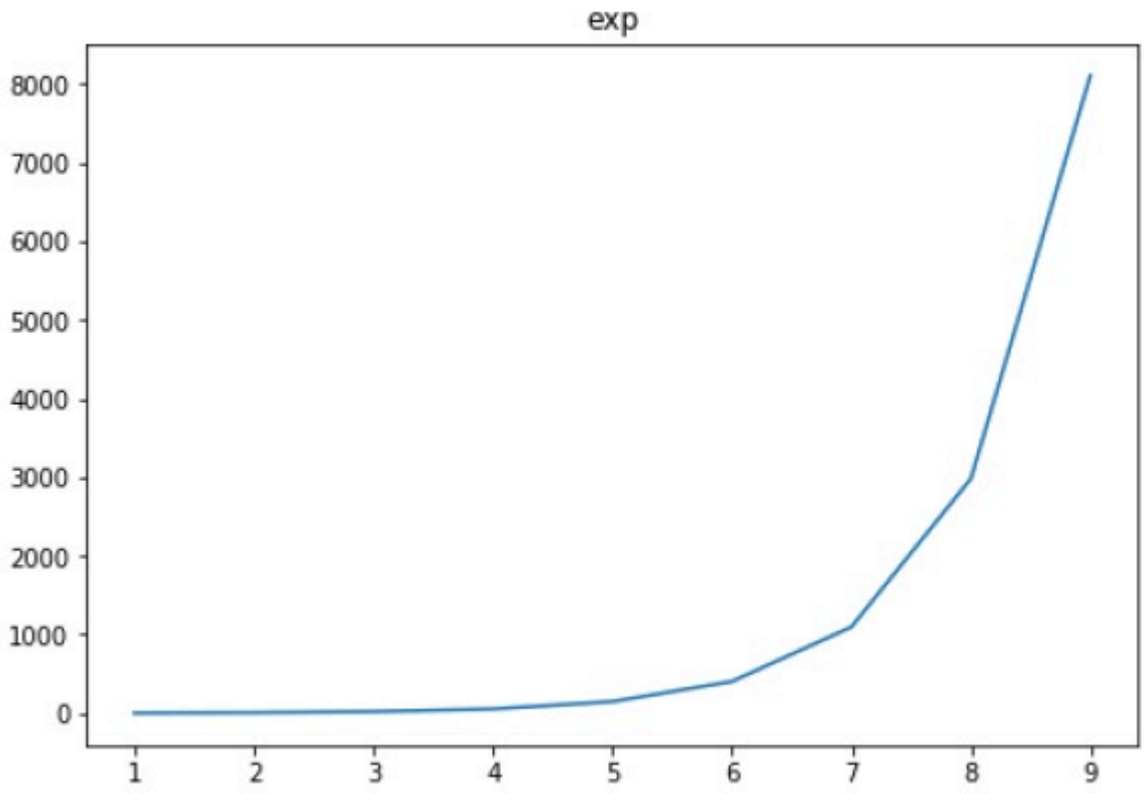


Matplotlib - Definindo Limites

Matplotlib chega automaticamente aos valores mínimo e máximo das variáveis a serem exibidas ao longo dos eixos x, y (e z no caso de gráfico 3D) de um gráfico. No entanto, é possível definir os limites explicitamente usando as funções **set_xlim()** e **set_ylim()** .

No gráfico a seguir, os limites de escala automática dos eixos xey são mostrados -

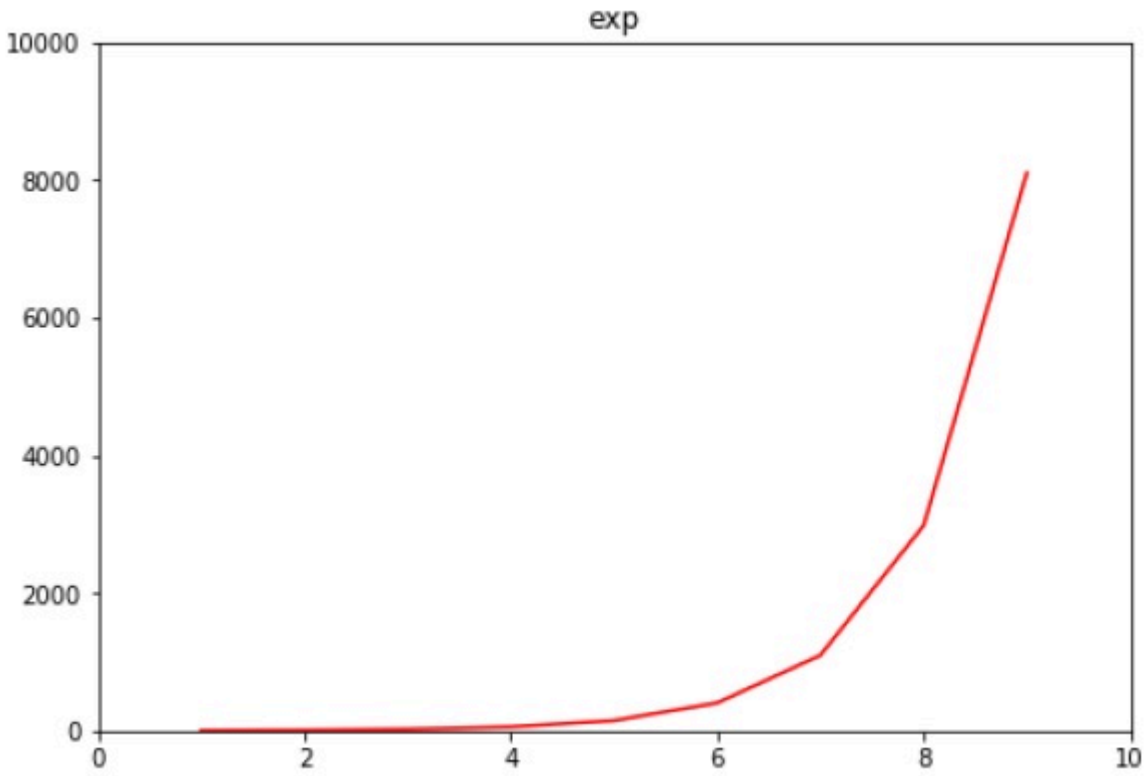
```
import matplotlib.pyplot as plt
fig = plt.figure()
a1 = fig.add_axes([0,0,1,1])
import numpy as np
x = np.arange(1,10)
a1.plot(x, np.exp(x))
a1.set_title('exp')
plt.show()
```



Agora formatamos os limites no eixo x para (0 a 10) e no eixo y (0 a 10000) -

```
import matplotlib.pyplot as plt
fig = plt.figure()
a1 = fig.add_axes([0,0,1,1])
import numpy as np
x = np.arange(1,10)
a1.plot(x, np.exp(x),'r')
a1.set_title('exp')
a1.set_ylim(0,10000)
```

```
a1.set_xlim(0,10)
plt.show()
```



Matplotlib - Configurando ticks e rótulos de ticks

Ticks are the markers denoting data points on axes. Matplotlib has so far - in all our previous examples - automatically taken over the task of spacing points on the axis.Matplotlib's default tick locators and formatters are designed to be generally sufficient in many common situations. Position and labels of ticks can be explicitly mentioned to suit specific requirements.

The **xticks()** and **yticks()** function takes a list object as argument. The elements in the list denote the positions on corresponding action where ticks will be displayed.

```
ax.set_xticks([2,4,6,8,10])
```

This method will mark the data points at the given positions with ticks.

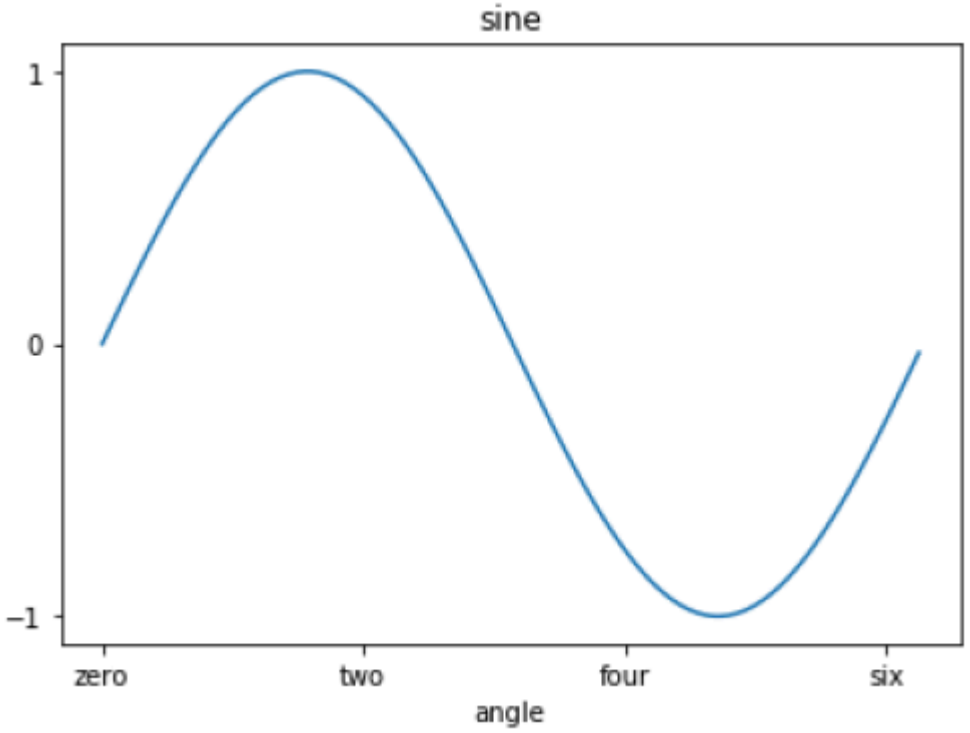
Similarly, labels corresponding to tick marks can be set by **set_xlabels()** and **set_ylabels()** functions respectively.

```
ax.set_xlabels(['two', 'four','six', 'eight', 'ten'])
```

This will display the text labels below the markers on the x axis.

Following example demonstrates the use of ticks and labels.

```
import matplotlib.pyplot as plt
import numpy as np
import math
x = np.arange(0, math.pi*2, 0.05)
fig = plt.figure()
ax = fig.add_axes([0.1, 0.1, 0.8, 0.8]) # main axes
y = np.sin(x)
ax.plot(x, y)
ax.set_xlabel('angle')
ax.set_title('sine')
ax.set_xticks([0,2,4,6])
ax.set_xticklabels(['zero','two','four','six'])
ax.set_yticks([-1,0,1])
plt.show()
```



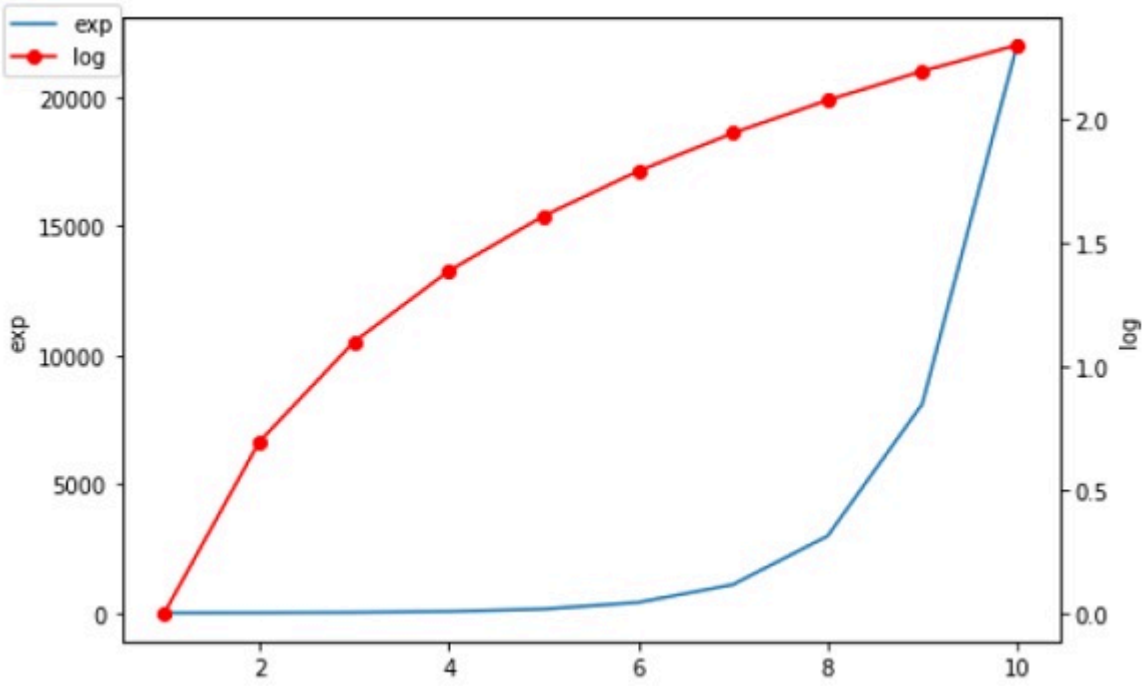
Matplotlib - Twin Axes

It is considered useful to have dual x or y axes in a figure. Moreso, when plotting curves with different units together. Matplotlib supports this with the twinxand twiny functions.

In the following example, the plot has dual y axes, one showing exp(x) and the other showing log(x) −

```
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure()
a1 = fig.add_axes([0,0,1,1])
x = np.arange(1,11)
```

```
a1.plot(x,np.exp(x))
a1.set_ylabel('exp')
a2 = a1.twinx()
a2.plot(x, np.log(x),'ro-')
a2.set_ylabel('log')
fig.legend(labels = ('exp','log'),loc='upper left')
plt.show()
```



Matplotlib - Bar Plot

A bar chart or bar graph is a chart or graph that presents categorical data with rectangular bars with heights or lengths proportional to the values that they represent. The bars can be plotted vertically or horizontally.

A bar graph shows comparisons among discrete categories. One axis of the chart shows the specific categories being compared, and the other axis represents a measured value.

Matplotlib API provides the **bar()** function that can be used in the MATLAB style use as well as object oriented API. The signature of bar() function to be used with axes object is as follows −

```
ax.bar(x, height, width, bottom, align)
```

The function makes a bar plot with the bound rectangle of size (x −width = 2; x + width=2; bottom; bottom + height).

The parameters to the function are −

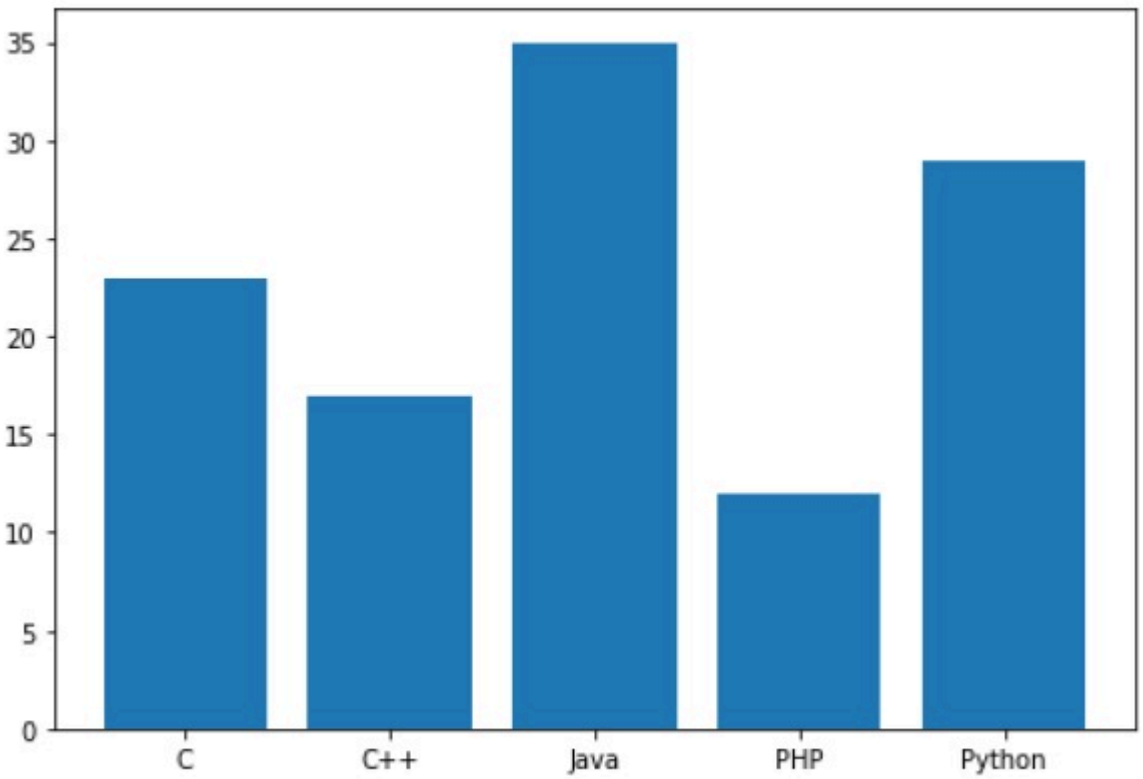
x	sequence of scalars representing the x coordinates of the bars. align controls if x is the bar center (default) or left edge.

height	scalar or sequence of scalars representing the height(s) of the bars.
width	scalar or array-like, optional. the width(s) of the bars default 0.8
bottom	scalar or array-like, optional. the y coordinate(s) of the bars default None.
align	{‘center’, ‘edge’}, optional, default ‘center’

The function returns a Matplotlib container object with all bars.

Following is a simple example of the Matplotlib bar plot. It shows the number of students enrolled for various courses offered at an institute.

```
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
langs = ['C', 'C++', 'Java', 'Python', 'PHP']
students = [23,17,35,29,12]
ax.bar(langs,students)
plt.show()
```

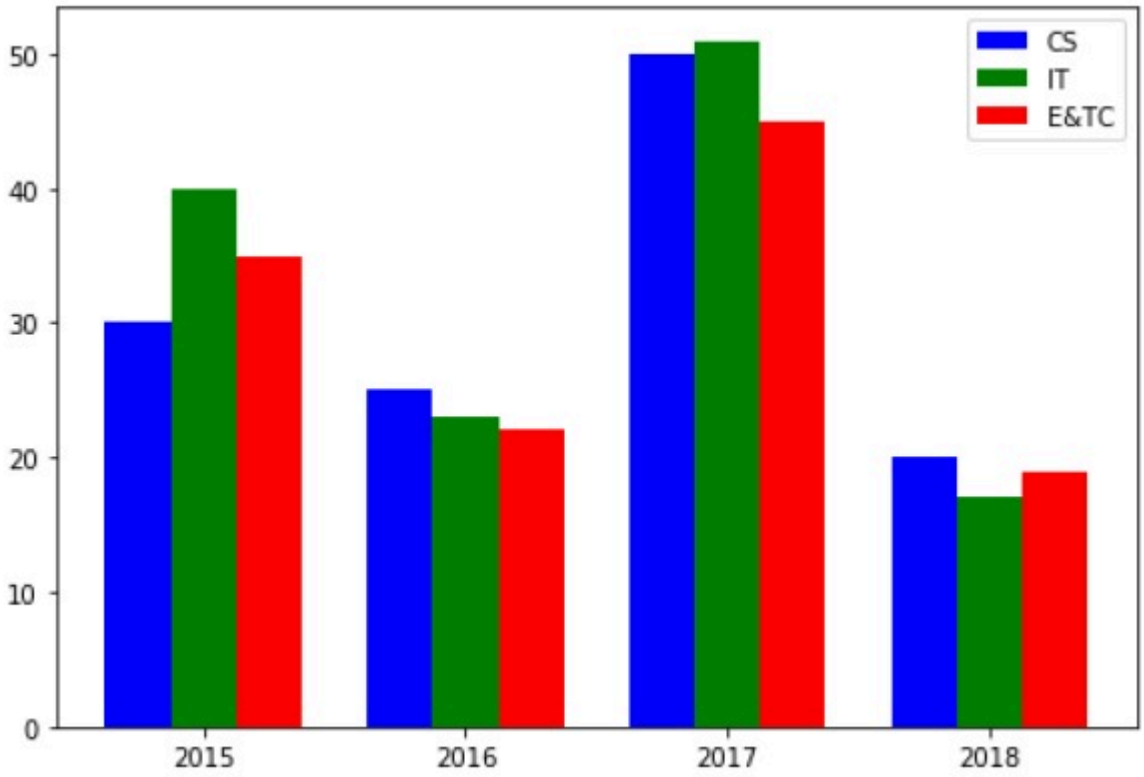


When comparing several quantities and when changing one variable, we might want a bar chart where we have bars of one color for one quantity value.

Podemos traçar vários gráficos de barras brincando com a espessura e as posições das barras. A variável de dados contém três séries de quatro valores. O script a seguir mostrará três gráficos de barras de quatro barras. As barras terão espessura de 0,25 unidades. Cada gráfico de barras será deslocado 0,25 unidades em relação ao anterior. O objeto de dados é um

multidicto contendo o número de alunos aprovados em três ramos de uma faculdade de engenharia nos últimos quatro anos.

```
import numpy as np
import matplotlib.pyplot as plt
data = [[30, 25, 50, 20],
[40, 23, 51, 17],
[35, 22, 45, 19]]
X = np.arange(4)
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.bar(X + 0.00, data[0], color = 'b', width = 0.25)
ax.bar(X + 0.25, data[1], color = 'g', width = 0.25)
ax.bar(X + 0.50, data[2], color = 'r', width = 0.25)
```

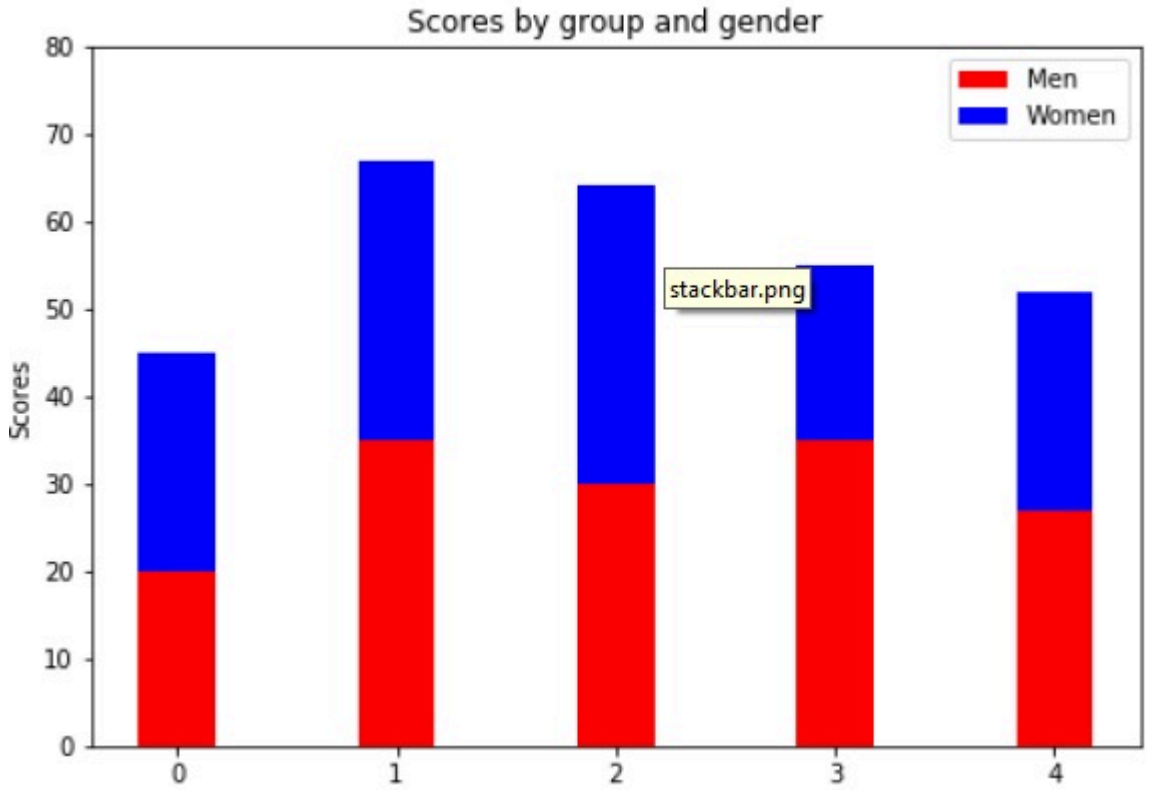


O gráfico de barras empilhadas empilha barras que representam grupos diferentes umas sobre as outras. A altura da barra resultante mostra o resultado combinado dos grupos.

O parâmetro inferior opcional da função **pyplot.bar()** permite especificar um valor inicial para uma barra. Em vez de ir de zero a um valor, ele irá de baixo para o valor. A primeira chamada para pyplot.bar() traça as barras azuis. A segunda chamada para pyplot.bar() traça as barras vermelhas, com a parte inferior das barras azuis no topo das barras vermelhas.

```
import numpy as np
import matplotlib.pyplot as plt
N = 5
menMeans = (20, 35, 30, 35, 27)
```

```
womenMeans = (25, 32, 34, 20, 25)
ind = np.arange(N) # the x locations for the groups
width = 0.35
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.bar(ind, menMeans, width, color='r')
ax.bar(ind, womenMeans, width,bottom=menMeans, color='b')
ax.set_ylabel('Scores')
ax.set_title('Scores by group and gender')
ax.set_xticks(ind, ('G1', 'G2', 'G3', 'G4', 'G5'))
ax.set_yticks(np.arange(0, 81, 10))
ax.legend(labels=['Men', 'Women'])
plt.show()
```



Matplotlib - Histograma

Um histograma é uma representação precisa da distribuição de dados numéricos. É uma estimativa da distribuição de probabilidade de uma variável contínua. É uma espécie de gráfico de barras.

Para construir um histograma, siga estas etapas -

- **Bin** o intervalo de valores.
- Divida todo o intervalo de valores em uma série de intervalos.
- Conte quantos valores caem em cada intervalo.

Os compartimentos geralmente são especificados como intervalos consecutivos e não sobrepostos de uma variável.

A função **matplotlib.pyplot.hist()** traça um histograma. Ele calcula e desenha o histograma de x.

Parâmetros

A tabela a seguir lista os parâmetros de um histograma -

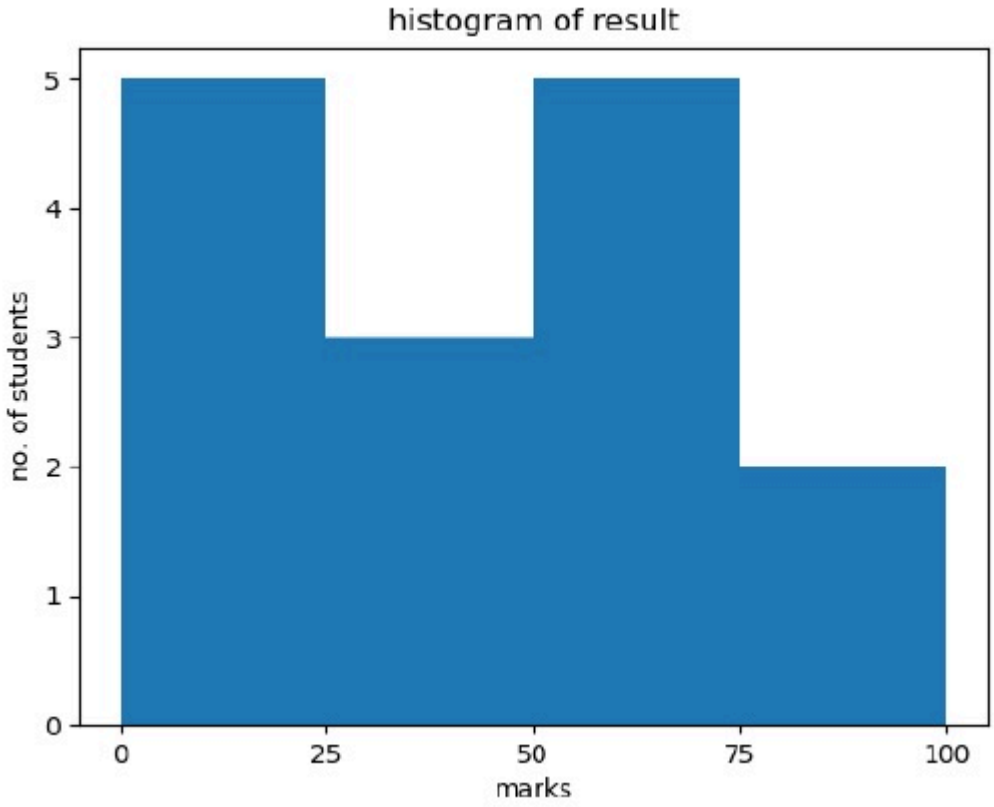
x	matriz ou sequência de matrizes
caixas	inteiro ou sequência ou 'auto', opcional
parâmetros opcionais	
faixa	A faixa inferior e superior das caixas.
densidade	Se True, o primeiro elemento da tupla de retorno serão as contagens normalizadas para formar uma densidade de probabilidade
cumulativo	Se for True, um histograma será calculado onde cada compartimento fornece as contagens nesse compartimento mais todos os compartimentos para valores menores.
tipo de histórico	O tipo de histograma a ser desenhado. O padrão é 'barra' • 'bar' é um histograma tradicional do tipo barra. Se forem fornecidos vários dados, as barras serão organizadas lado a lado. • 'barstacked' é um histograma tipo barra onde vários dados são empilhados uns sobre os outros. • 'step' gera um gráfico de linhas que por padrão não é preenchido. • 'stepfilled' gera um gráfico de linha que é preenchido por padrão.

O exemplo a seguir traça um histograma de notas obtidas pelos alunos em uma turma. Quatro compartimentos, 0-25, 26-50, 51-75 e 76-100 são definidos. O histograma mostra o número de alunos que se enquadram nesta faixa.

```
from matplotlib import pyplot as plt
import numpy as np
fig,ax = plt.subplots(1,1)
a = np.array([22,87,5,43,56,73,55,54,11,20,51,5,79,31,27])
ax.hist(a, bins = [0,25,50,75,100])
ax.set_title("histogram of result")
ax.set_xticks([0,25,50,75,100])
```

```
ax.set_xlabel('marks')
ax.set_ylabel('no. of students')
plt.show()
```

O gráfico aparece conforme mostrado abaixo -



Matplotlib - Gráfico de pizza

Um gráfico de pizza só pode exibir uma série de dados. Os gráficos de pizza mostram o tamanho dos itens (chamados de cunha) em uma série de dados, proporcional à soma dos itens. Os pontos de dados em um gráfico de pizza são mostrados como uma porcentagem de toda a pizza.

A API Matplotlib possui uma função **pie()** que gera um diagrama de pizza representando dados em um array. A área fracionária de cada cunha é dada por **x/sum(x)** . Se soma (x) <1, então os valores de x fornecem a área fracionária diretamente e a matriz não será normalizada. A torta resultante terá uma fatia vazia de tamanho 1 - soma(x).

O gráfico de pizza fica melhor se a figura e os eixos forem quadrados ou se o aspecto dos Eixos for igual.

Parâmetros

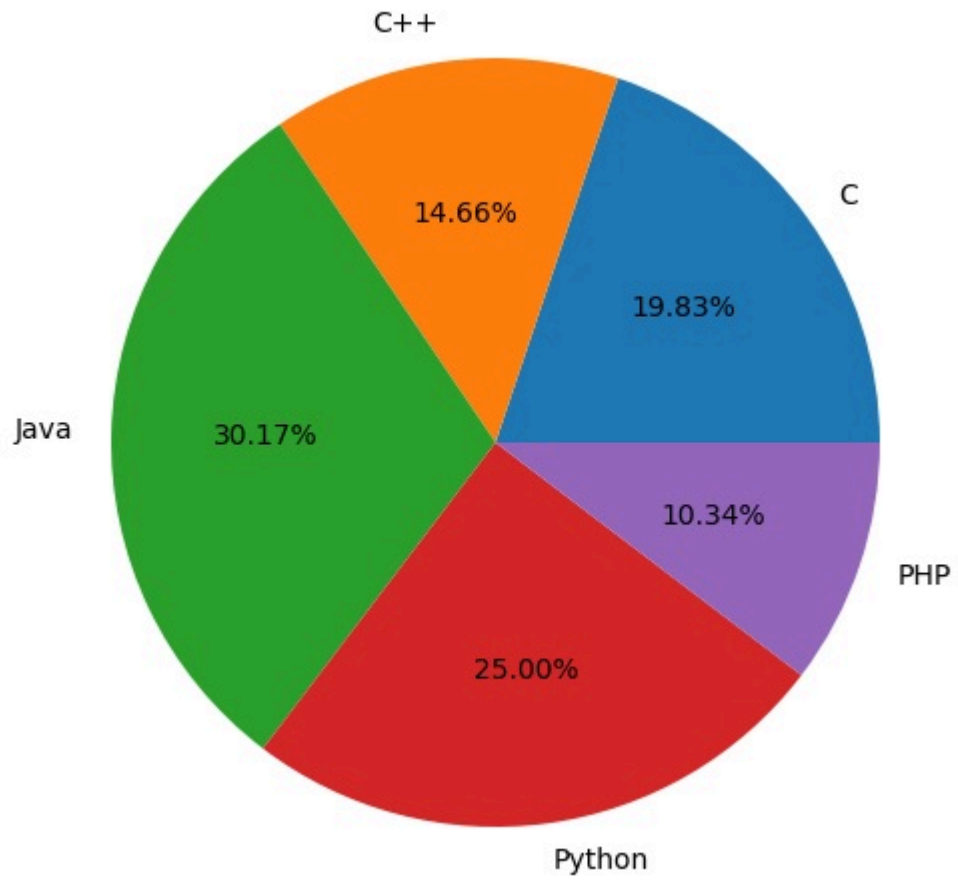
A tabela a seguir lista os parâmetros de um gráfico de pizza -

x	semelhante a uma matriz. Os tamanhos das cunhas.
rótulos	lista. Uma sequência de strings que fornece os rótulos para cada fatia.

Cores	Uma sequência de matplotlibcolorargs pela qual o gráfico de pizza percorrerá. Se Nenhum, usará as cores do ciclo atualmente ativo.
Autopct	string, usada para rotular as fatias com seu valor numérico. A etiqueta será colocada dentro da cunha. A string de formato será fmt%pct.

O código a seguir usa a função pie() para exibir o gráfico de pizza da lista de alunos matriculados em vários cursos de linguagem de informática. A porcentagem proporcional é exibida dentro da respectiva fatia com a ajuda do parâmetro **autopct** que está definido como% 1,2f%.

```
from matplotlib import pyplot as plt
import numpy as np
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.axis('equal')
langs = ['C', 'C++', 'Java', 'Python', 'PHP']
students = [23,17,35,29,12]
ax.pie(students, labels = langs,autopct='%1.2f%%')
plt.show()
```

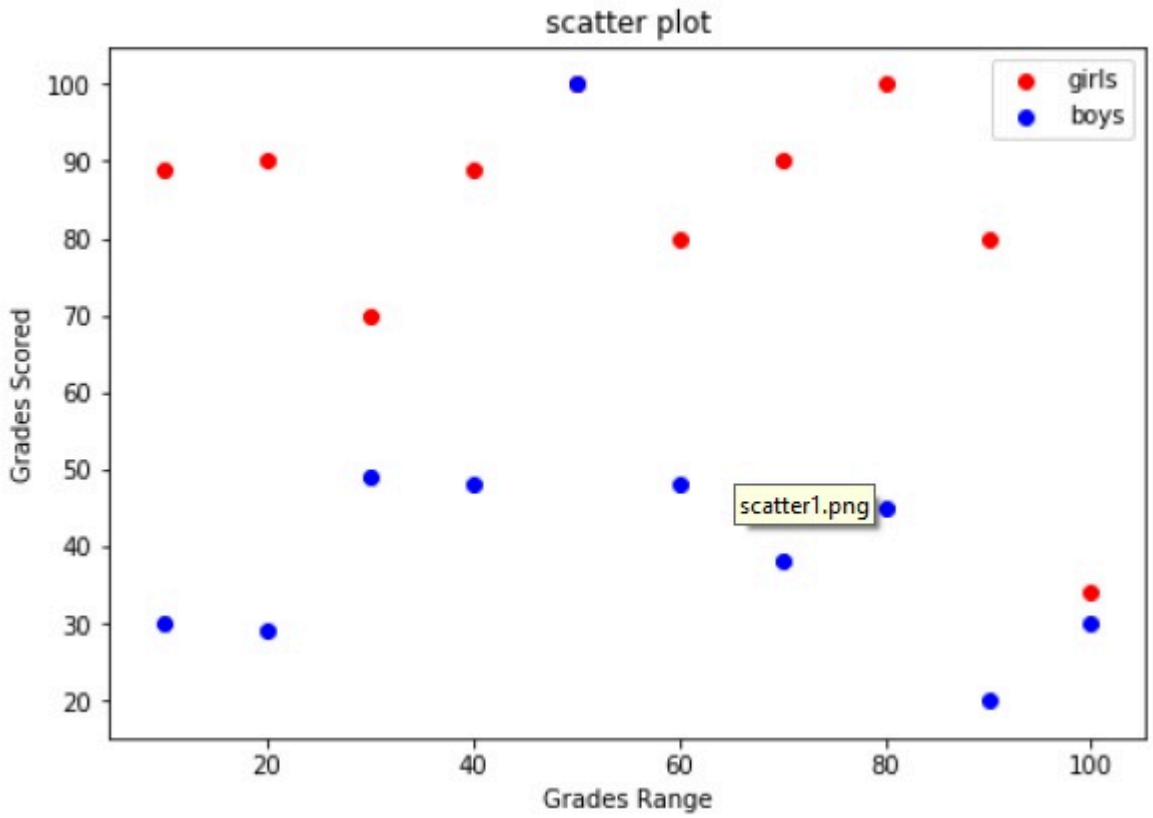


Matplotlib - gráfico de dispersão

Os gráficos de dispersão são usados para traçar pontos de dados nos eixos horizontal e vertical na tentativa de mostrar o quanto uma variável é afetada por outra. Cada linha da tabela de dados é representada por um marcador cuja posição depende de seus valores nas colunas definidas nos eixos X e Y. Uma terceira variável pode ser definida para corresponder à cor ou tamanho dos marcadores, acrescentando assim mais uma dimensão ao gráfico.

O script abaixo traça um diagrama de dispersão da faixa de notas versus as notas de meninos e meninas em duas cores diferentes.

```
import matplotlib.pyplot as plt
girls_grades = [89, 90, 70, 89, 100, 80, 90, 100, 80, 34]
boys_grades = [30, 29, 49, 48, 100, 48, 38, 45, 20, 30]
grades_range = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
fig=plt.figure()
ax=fig.add_axes([0,0,1,1])
ax.scatter(grades_range, girls_grades, color='r')
ax.scatter(grades_range, boys_grades, color='b')
ax.set_xlabel('Grades Range')
ax.set_ylabel('Grades Scored')
ax.set_title('scatter plot')
plt.show()
```



Matplotlib - Gráfico de contorno

Gráficos de contorno (às vezes chamados de Gráficos de Nível) são uma forma de mostrar uma superfície tridimensional em um plano bidimensional. Ele representa graficamente duas

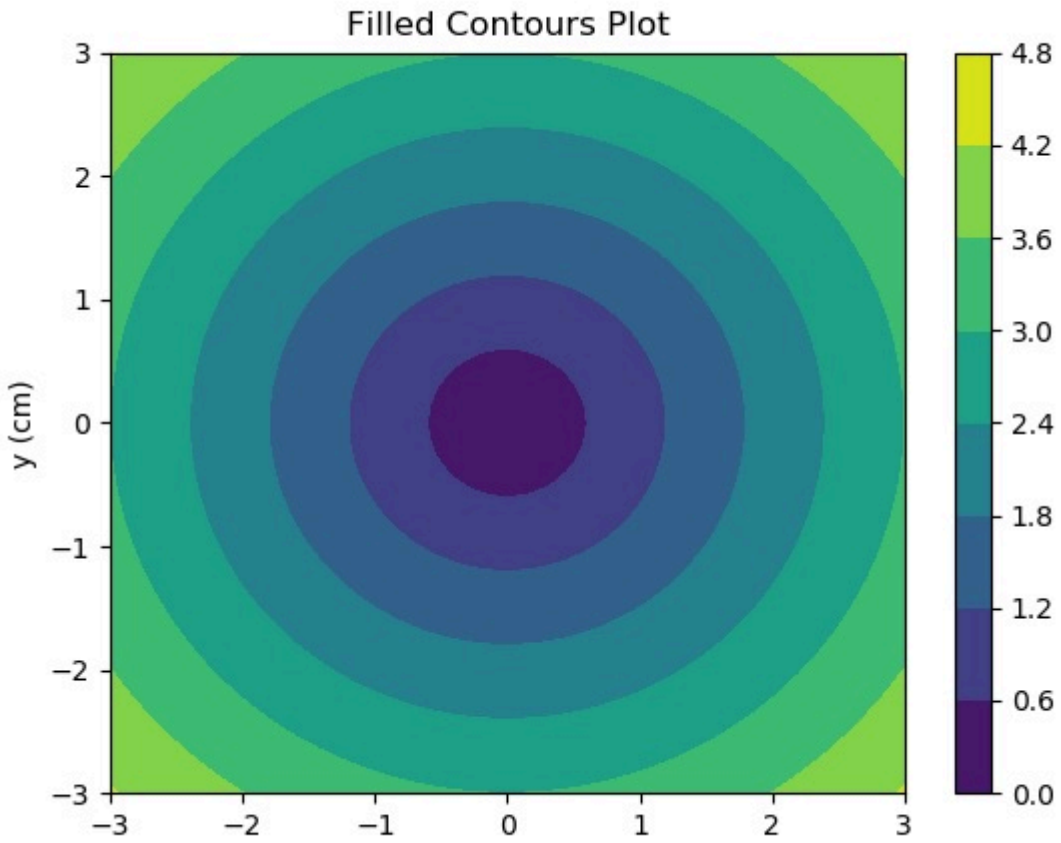
variáveis preditoras XY no eixo y e uma variável de resposta Z como contornos. Esses contornos são às vezes chamados de fatias z ou valores de resposta iso.

Um gráfico de contorno é apropriado se você quiser ver como o valor Z muda em função de duas entradas X e Y, de modo que Z = f(X,Y). Uma linha de contorno ou isolinha de uma função de duas variáveis é uma curva ao longo da qual a função tem um valor constante.

As variáveis independentes xey são geralmente restritas a uma grade regular chamada meshgrid. O numpy.meshgrid cria uma grade retangular a partir de uma matriz de valores x e uma matriz de valores y.

A API Matplotlib contém funções contorno() e contornof() que desenham linhas de contorno e contornos preenchidos, respectivamente. Ambas as funções precisam de três parâmetros x, y e z.

```
import numpy as np
import matplotlib.pyplot as plt
xlist = np.linspace(-3.0, 3.0, 100)
ylist = np.linspace(-3.0, 3.0, 100)
X, Y = np.meshgrid(xlist, ylist)
Z = np.sqrt(X**2 + Y**2)
fig,ax=plt.subplots(1,1)
cp = ax.contourf(X, Y, Z)
fig.colorbar(cp) # Add a colorbar to a plot
ax.set_title('Filled Contours Plot')
#ax.set_xlabel('x (cm)')
ax.set_ylabel('y (cm)')
plt.show()
```



Matplotlib - Gráfico Quiver

Um gráfico de quiver exibe os vetores de velocidade como setas com componentes (u,v) nos pontos (x,y).

```
quiver(x,y,u,v)
```

O comando acima plota vetores como setas nas coordenadas especificadas em cada par correspondente de elementos em x e y.

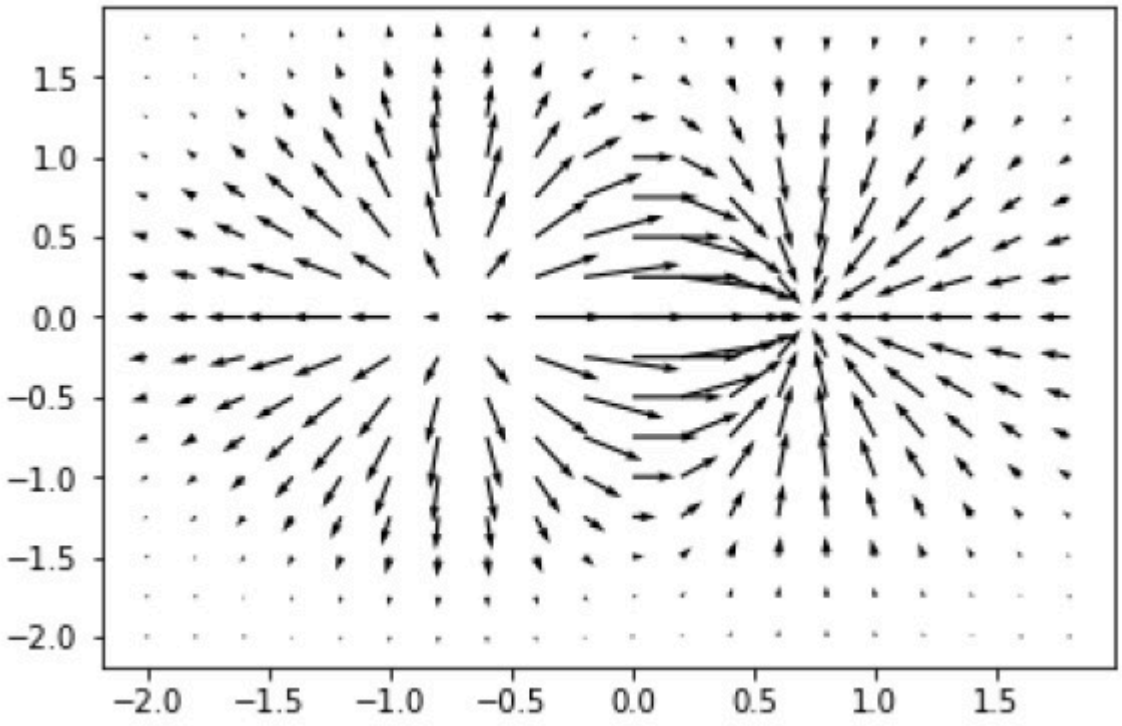
Parâmetros

A tabela a seguir lista os diferentes parâmetros para o gráfico Quiver -

x	Matriz 1D ou 2D, sequência. As coordenadas x das localizações das setas
sim	Matriz 1D ou 2D, sequência. As coordenadas y das localizações das setas
você	Matriz 1D ou 2D, sequência. Os componentes x dos vetores seta
v	Matriz 1D ou 2D, sequência. Os componentes y dos vetores seta
c	Matriz 1D ou 2D, sequência. As cores das setas

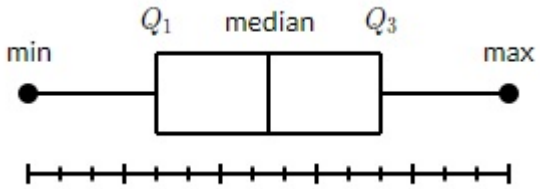
O código a seguir desenha um gráfico de aljava simples -

```
import matplotlib.pyplot as plt
import numpy as np
x,y = np.meshgrid(np.arange(-2, 2, .2), np.arange(-2, 2, .25))
z = x*np.exp(-x**2 - y**2)
v, u = np.gradient(z, .2, .2)
fig, ax = plt.subplots()
q = ax.quiver(x,y,u,v)
plt.show()
```



Matplotlib - Gráfico de caixa

Um box plot, também conhecido como bigode, exibe um resumo de um conjunto de dados contendo o mínimo, primeiro quartil, mediana, terceiro quartil e máximo. Em um box plot, desenhamos uma caixa do primeiro ao terceiro quartil. Uma linha vertical passa pela caixa na mediana. Os bigodes vão de cada quartil ao mínimo ou máximo.



Vamos criar os dados para os boxplots. Usamos a função **numpy.random.normal()** para criar os dados falsos. São necessários três argumentos, média e desvio padrão da distribuição normal e o número de valores desejados.

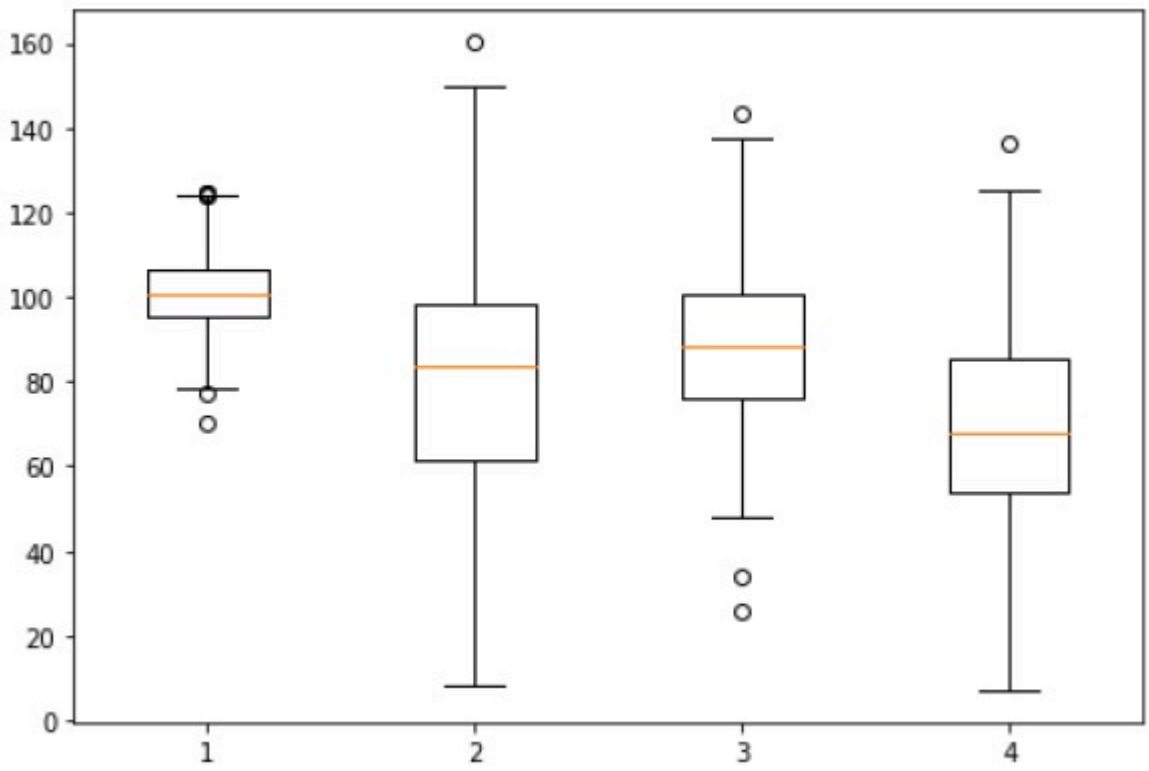
```
np.random.seed(10)
collectn_1 = np.random.normal(100, 10, 200)
```

```
collectn_2 = np.random.normal(80, 30, 200)
collectn_3 = np.random.normal(90, 20, 200)
collectn_4 = np.random.normal(70, 25, 200)
```

A lista de arrays que criamos acima é a única entrada necessária para a criação do boxplot. Usando a linha de código **data_to_plot** , podemos criar o boxplot com o seguinte código -

```
fig = plt.figure()
# Create an axes instance
ax = fig.add_axes([0,0,1,1])
# Create the boxplot
bp = ax.boxplot(data_to_plot)
plt.show()
```

A linha de código acima irá gerar a seguinte saída -

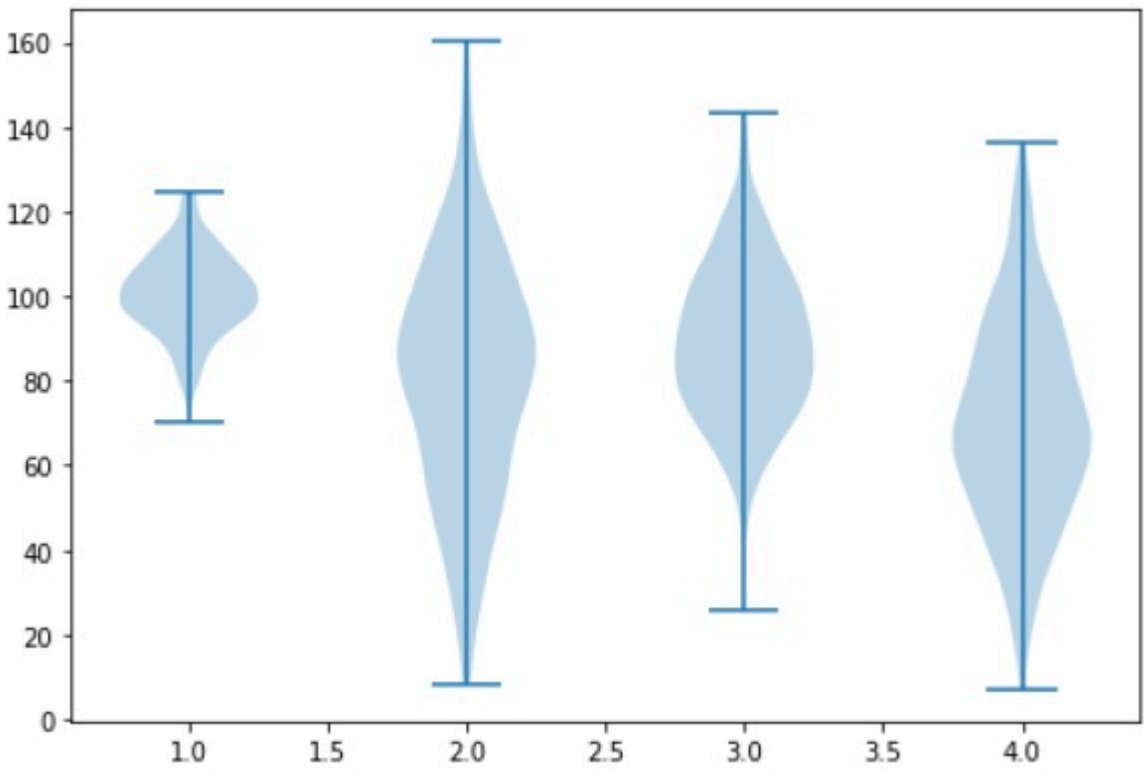


Matplotlib - Enredo de Violino

Os gráficos de violino são semelhantes aos gráficos de caixa, exceto que também mostram a densidade de probabilidade dos dados em valores diferentes. Esses gráficos incluem um marcador para a mediana dos dados e uma caixa indicando o intervalo interquartil, como nos box plots padrão. Sobreposto neste box plot está uma estimativa de densidade do kernel. Assim como os gráficos de caixa, os gráficos de violino são usados para representar a comparação de uma distribuição variável (ou distribuição de amostra) em diferentes "categorias".

Um gráfico de violino é mais informativo do que um gráfico de caixa simples. Na verdade, enquanto um gráfico de caixa mostra apenas estatísticas resumidas, como média/mediana e intervalos interquartis, o gráfico de violino mostra a distribuição completa dos dados.

```
importar matplotlib . pyplot como plt
 importar numpy como np

np . aleatório . semente ( 10 )
coletarn_1 = np . aleatório . normal ( 100 , 10 , 200 )
coletan_2 = np . aleatório . normal ( 80 , 30 , 200 )
coletan_3 = np . aleatório . normal ( 90 , 20 , 200 )
coletan_4 = np . aleatório . normais ( 70 , 25 , 200 )

## combine essas diferentes coleções em uma lista
data_to_plot = [ collectn_1 , collectn_2 , collectn_3 , collectn_4 ]

# Crie uma instância de figura
fig = plt . figura ()

# Crie uma instância de eixos
ax = fig . add_axes ([ 0 , 0 , 1 , 1 ])

# Crie o boxplot
bp = ax . violinplot ( data_to_plot )
plt . mostrar ()
```



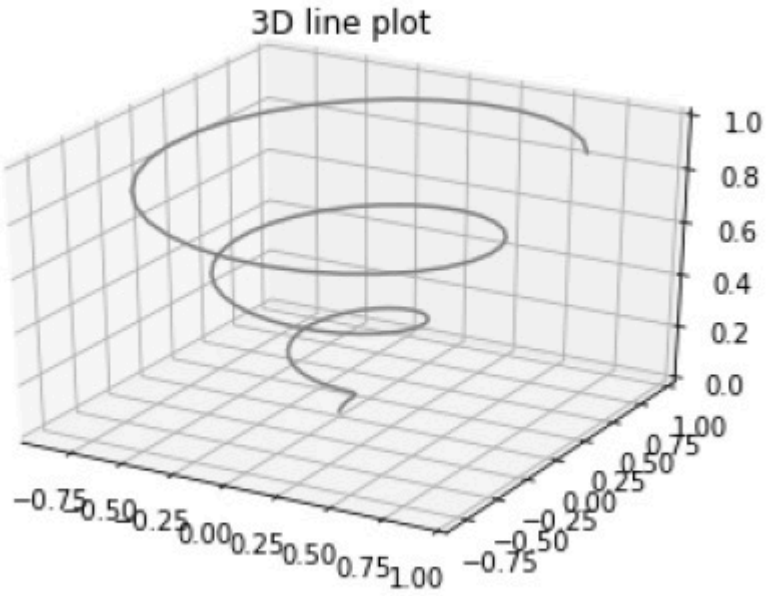
Matplotlib - plotagem tridimensional

Embora o Matplotlib tenha sido inicialmente projetado com apenas plotagem bidimensional em mente, alguns utilitários de plotagem tridimensional foram construídos sobre a exibição bidimensional do Matplotlib em versões posteriores, para fornecer um conjunto de ferramentas para visualização de dados tridimensionais. Gráficos tridimensionais são habilitados importando o **kit de ferramentas mplot3d** , incluído no pacote Matplotlib.

Eixos tridimensionais podem ser criados passando a palavra-chave project='3d' para qualquer uma das rotinas normais de criação de eixos.

```
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
fig = plt.figure()
ax = plt.axes(projection='3d')
z = np.linspace(0, 1, 100)
x = z * np.sin(20 * z)
y = z * np.cos(20 * z)
ax.plot3D(x, y, z, 'gray')
ax.set_title('3D line plot')
plt.show()
```

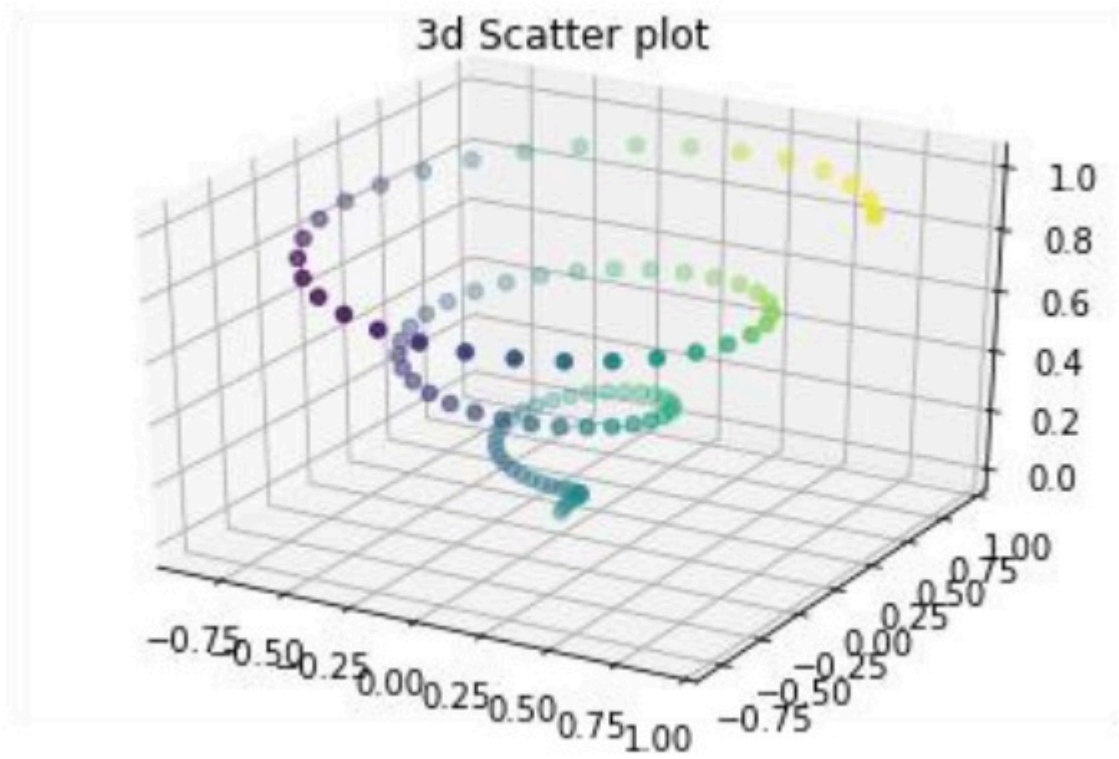
Agora podemos traçar uma variedade de tipos de gráficos tridimensionais. O gráfico tridimensional mais básico é um **gráfico de linha 3D** criado a partir de conjuntos de triplos (x, y, z). Isso pode ser criado usando a função ax.plot3D.



O gráfico de dispersão 3D é gerado usando a função **ax.scatter3D** .

```
from mpl_toolkits import mplot3d
import numpy as np
```

```
import matplotlib.pyplot as plt
fig = plt.figure()
ax = plt.axes(projection='3d')
z = np.linspace(0, 1, 100)
x = z * np.sin(20 * z)
y = z * np.cos(20 * z)
c = x + y
ax.scatter(x, y, z, c=c)
ax.set_title('3d Scatter plot')
plt.show()
```

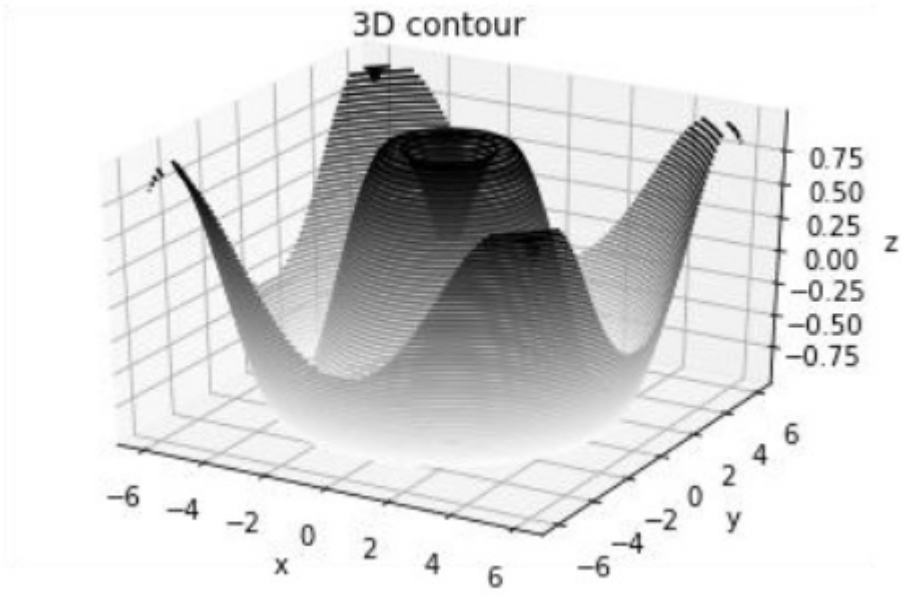


Matplotlib - Gráfico de contorno 3D

A função **ax.contour3D()** cria um gráfico de contorno tridimensional. Requer que todos os dados de entrada estejam na forma de grades regulares bidimensionais, com os dados Z avaliados em cada ponto. Aqui, mostraremos um diagrama de contorno tridimensional de uma função senoidal tridimensional.

```
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
def f(x, y):
   return np.sin(np.sqrt(x ** 2 + y ** 2))

x = np.linspace(-6, 6, 30)
y = np.linspace(-6, 6, 30)
```

```
X, Y = np.meshgrid(x, y)
Z = f(X, Y)

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.contour3D(X, Y, Z, 50, cmap='binary')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
ax.set_title('3D contour')
plt.show()
```



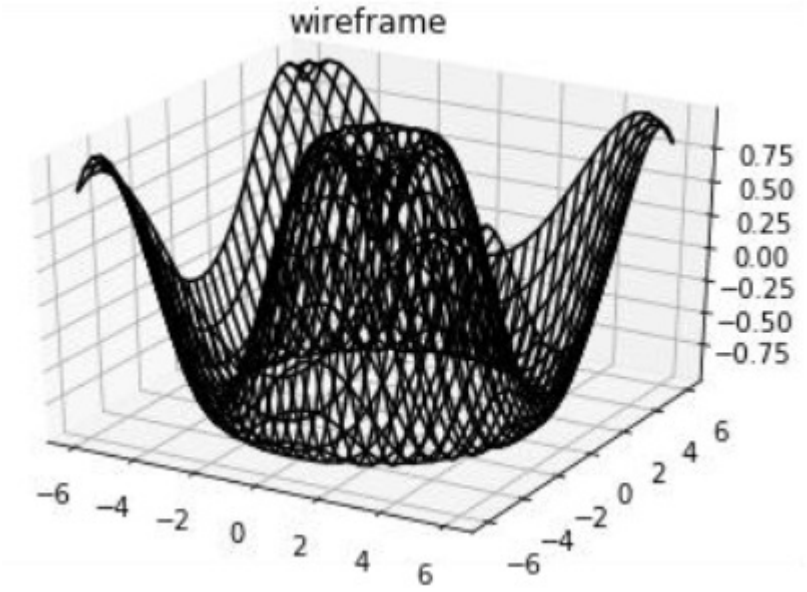
Matplotlib - gráfico de wireframe 3D

O gráfico wireframe pega uma grade de valores e a projeta na superfície tridimensional especificada e pode tornar as formas tridimensionais resultantes bastante fáceis de visualizar. A função **plot_wireframe()** é usada para esse propósito -

```
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
def f(x, y):
   return np.sin(np.sqrt(x ** 2 + y ** 2))

x = np.linspace(-6, 6, 30)
y = np.linspace(-6, 6, 30)

X, Y = np.meshgrid(x, y)
Z = f(X, Y)
```

```
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_wireframe(X, Y, Z, color='black')
ax.set_title('wireframe')
plt.show()
```

A linha de código acima irá gerar a seguinte saída -

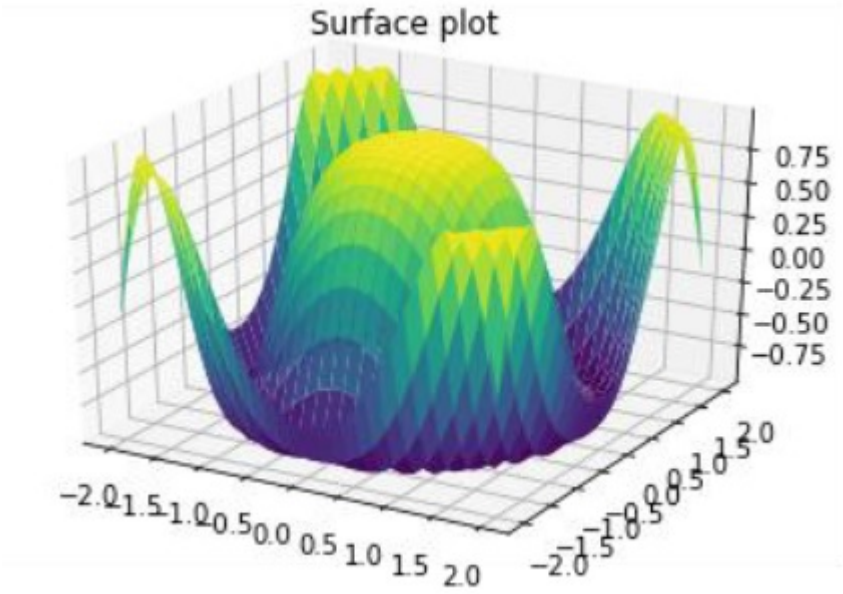


Matplotlib - gráfico de superfície 3D

O gráfico de superfície mostra uma relação funcional entre uma variável dependente designada (Y) e duas variáveis independentes (X e Z). O gráfico é um gráfico complementar ao gráfico de contorno. Um gráfico de superfície é como um gráfico de modelo de arame, mas cada face do modelo de arame é um polígono preenchido. Isso pode ajudar na percepção da topologia da superfície que está sendo visualizada. A função **plot_surface()** x,y e z como argumentos.

```
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
x = np.outer(np.linspace(-2, 2, 30), np.ones(30))
y = x.copy().T # transpose
z = np.cos(x ** 2 + y ** 2)

fig = plt.figure()
ax = plt.axes(projection='3d')

ax.plot_surface(x, y, z,cmap='viridis', edgecolor='none')
ax.set_title('Surface plot')
plt.show()
```

A linha de código acima irá gerar a seguinte saída -



Matplotlib - Trabalhando com Texto

Matplotlib tem amplo suporte a texto, incluindo suporte para expressões matemáticas, suporte **TrueType** para saídas raster e vetoriais, texto separado por nova linha com rotações arbitrárias e suporte Unicode. Matplotlib inclui seu próprio matplotlib.font_manager que implementa um algoritmo de localização de fontes compatível com W3C de plataforma cruzada.

O usuário tem grande controle sobre as propriedades do texto (tamanho da fonte, espessura da fonte, localização e cor do texto, etc.). Matplotlib implementa um grande número de símbolos e comandos matemáticos TeX.

A seguinte lista de comandos é usada para criar texto na interface Pyplot -

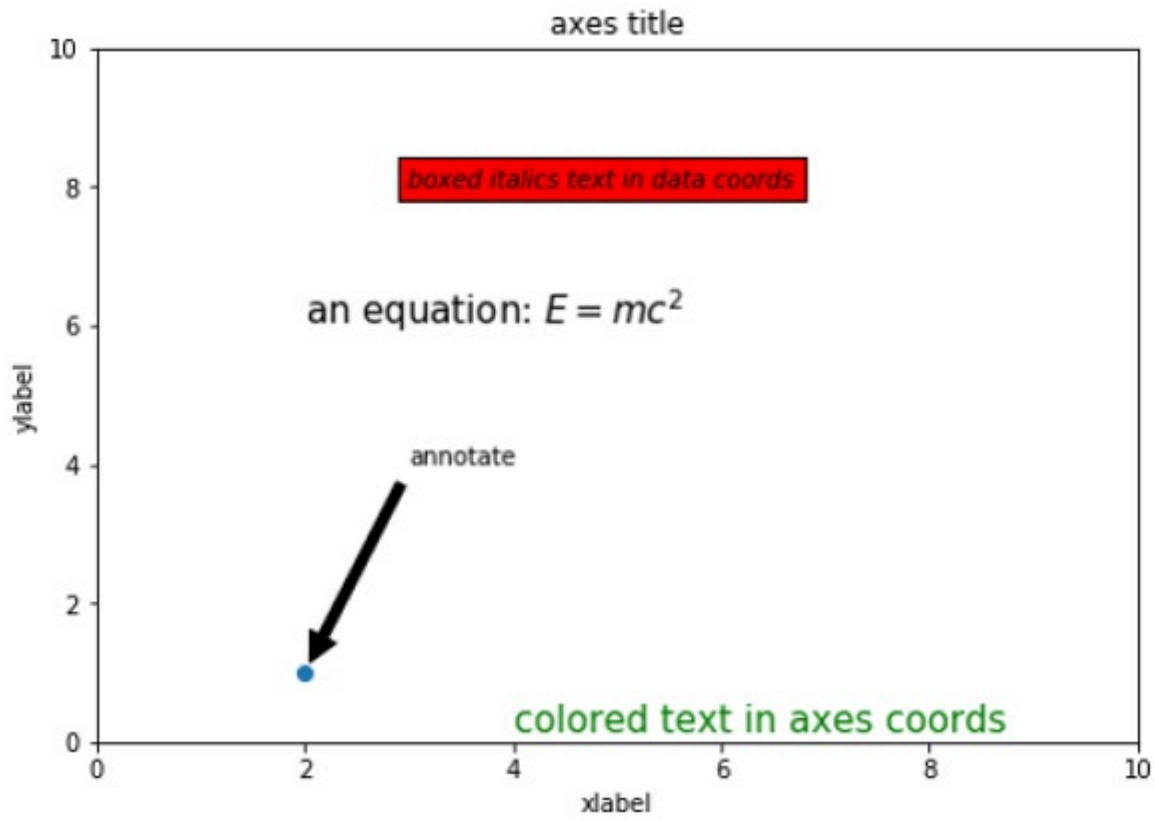
texto	Adicione texto em um local arbitrário dos eixos.
anotar	Adicione uma anotação, com uma seta opcional, em um local arbitrário dos Eixos.
rótulo x	Adicione um rótulo ao eixo x dos eixos.
rótulo	Adicione um rótulo ao eixo y dos eixos.
título	Adicione um título aos eixos.
figtexto	Adicione texto em um local arbitrário da Figura.
legenda	Adicione um título à figura.

Todas essas funções criam e retornam uma instância **matplotlib.text.Text()** .

Os scripts a seguir demonstram o uso de algumas das funções acima -

```
import matplotlib.pyplot as plt
fig = plt.figure()

ax = fig.add_axes([0,0,1,1])

ax.set_title('axes title')
ax.set_xlabel('xlabel')
ax.set_ylabel('ylabel')
ax.text(3, 8, 'boxed italics text in data coords', style='italic',
bbox = {'facecolor': 'red'})
ax.text(2, 6, r'an equation: $E = mc^2$', fontsize = 15)
ax.text(4, 0.05, 'colored text in axes coords',
verticalalignment = 'bottom', color = 'green', fontsize = 15)
ax.plot([2], [1], 'o')
ax.annotate('annotate', xy = (2, 1), xytext = (3, 4),
arrowprops = dict(facecolor = 'black', shrink = 0.05))
ax.axis([0, 10, 0, 10])
plt.show()
```

A linha de código acima irá gerar a seguinte saída -



Matplotlib - Expressões Matemáticas

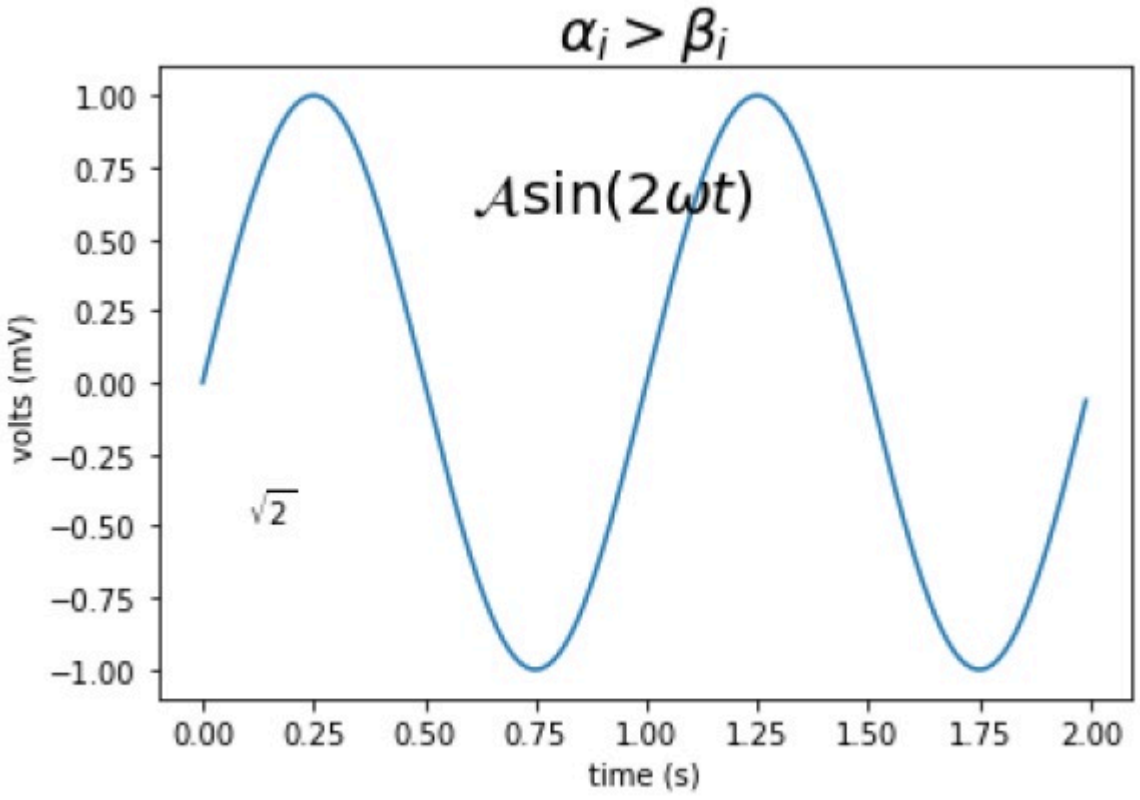
Você pode usar um subconjunto TeXmarkup em qualquer string de texto Matplotlib colocando-o dentro de um par de cifrões ($).

```
# math text
plt.title(r'$\alpha > \beta$')
```

Para fazer subscritos e sobrescritos, use os símbolos '_' e '^' -

```
r'$\alpha_i> \beta_i$'

import numpy as np
import matplotlib.pyplot as plt
t = np.arange(0.0, 2.0, 0.01)
s = np.sin(2*np.pi*t)

plt.plot(t,s)
plt.title(r'$\alpha_i> \beta_i$', fontsize=20)

plt.text(0.6, 0.6, r'$\mathcal{A}\mathrm{sin}(2 \omega t)$', fontsize = 20)
plt.text(0.1, -0.5, r'$\sqrt{2}$', fontsize=10)
plt.xlabel('time (s)')
plt.ylabel('volts (mV)')
plt.show()
```

A linha de código acima irá gerar a seguinte saída -



Matplotlib - Trabalhando com Imagens

O módulo de imagem no pacote Matplotlib fornece funcionalidades necessárias para carregar, redimensionar e exibir imagens.

O carregamento de dados de imagem é suportado pela biblioteca Pillow. Nativamente, o Matplotlib oferece suporte apenas a imagens PNG. Os comandos mostrados abaixo voltam ao Pillow se a leitura nativa falhar.

A imagem usada neste exemplo é um arquivo PNG, mas lembre-se do requisito do travesseiro para seus próprios dados. A função **imread()** é usada para ler dados de imagem em um objeto **ndarray** do tipo float32.

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
img = mpimg.imread('mtplogo.png')
```

Supondo que a imagem a seguir denominada **mtplogo.png** esteja presente no diretório de trabalho atual.

Qualquer array contendo dados de imagem pode ser salvo em um arquivo de disco executando a função **imsave()** . Aqui, uma versão invertida verticalmente do arquivo png original é salva fornecendo o parâmetro Origin como inferior.

```
plt.imsave("logo.png", img, cmap = 'gray', origin = 'lower')
```

A nova imagem aparece como abaixo se aberta em qualquer visualizador de imagens.

Para desenhar a imagem no visualizador Matplotlib, execute a função **imshow()** .

```
imgplot = plt.imshow(img)
```

Matplotlib - Transformações

O pacote matplotlib é construído sobre uma estrutura de transformação para mover-se facilmente entre sistemas de coordenadas. Quatro sistemas de coordenadas podem ser usados. Os sistemas são descritos resumidamente na tabela abaixo -

Coordenada	Objeto de transformação	Descrição
Dados	ax.transData	O sistema de coordenadas de dados terrestres do usuário. controlado pelo xlim e ylim
Eixos	ax.transAxes	O sistema de coordenadas dos eixos. (0,0) está no canto inferior esquerdo e (1,1) está no canto superior direito dos eixos.
Figura	fig.transFigura	O sistema de coordenadas da Figura. (0,0) está no canto inferior esquerdo e (1,1) está no canto superior direito da figura
mostrar	Nenhum	Este é o sistema de coordenadas de pixel da tela. (0,0) é o canto inferior esquerdo e (largura, altura) é o canto superior direito da exibição em pixels. Alternativamente, o(matplotlib.transforms.IdentityTransform()) pode ser usado em vez de Nenhum.

Considere o seguinte exemplo -

```
axes.text(x,y,"my label")
```

O texto é colocado na posição teórica de um ponto de dados (x,y). Assim falaríamos de "coordenadas de dados".

Usando outros objetos de transformação, o posicionamento pode ser controlado. Por exemplo, se o teste acima for colocado no sistema de coordenadas do centro dos eixos, execute a seguinte linha de código -

```
axes.text(0.5, 0.5, "middle of graph", transform=axes.transAxes)
```

Essas transformações podem ser usadas para qualquer tipo de objeto Matplotlib. A transformação padrão para **ax.text** é **ax.transData** e a transformação padrão para **fig.text** é **fig.transFigure.**

O sistema de coordenadas dos eixos é extremamente útil ao colocar texto em seus eixos. Muitas vezes você pode querer um balão de texto em um local fixo; por exemplo, no canto

superior esquerdo do painel de eixos e manter esse local fixo quando você faz panorâmica ou zoom.