Introdução ao Matplotlib

O matplotlib é a biblioteca mais popular para visualização de dados em Python. Ele permite a geração de diferentes tipos de gráfico, como gráficos de linha, de barras, de pizza, de dispersão e histogramas. É possível personalizar os gráficos de diversas maneiras, acrescentando múltiplas fontes de dados, trocando cor, formato dos marcadores ou gerando diversos gráficos na mesma imagem.

1. Instalação

Se você usa Anaconda, o matplotlib já está instalado. Caso contrário, abra o terminal de seu sistema operaciona(*Prompt de Comando* no Windows) e digite:

```
pip install matplotlib
```

2. Gráficos simples

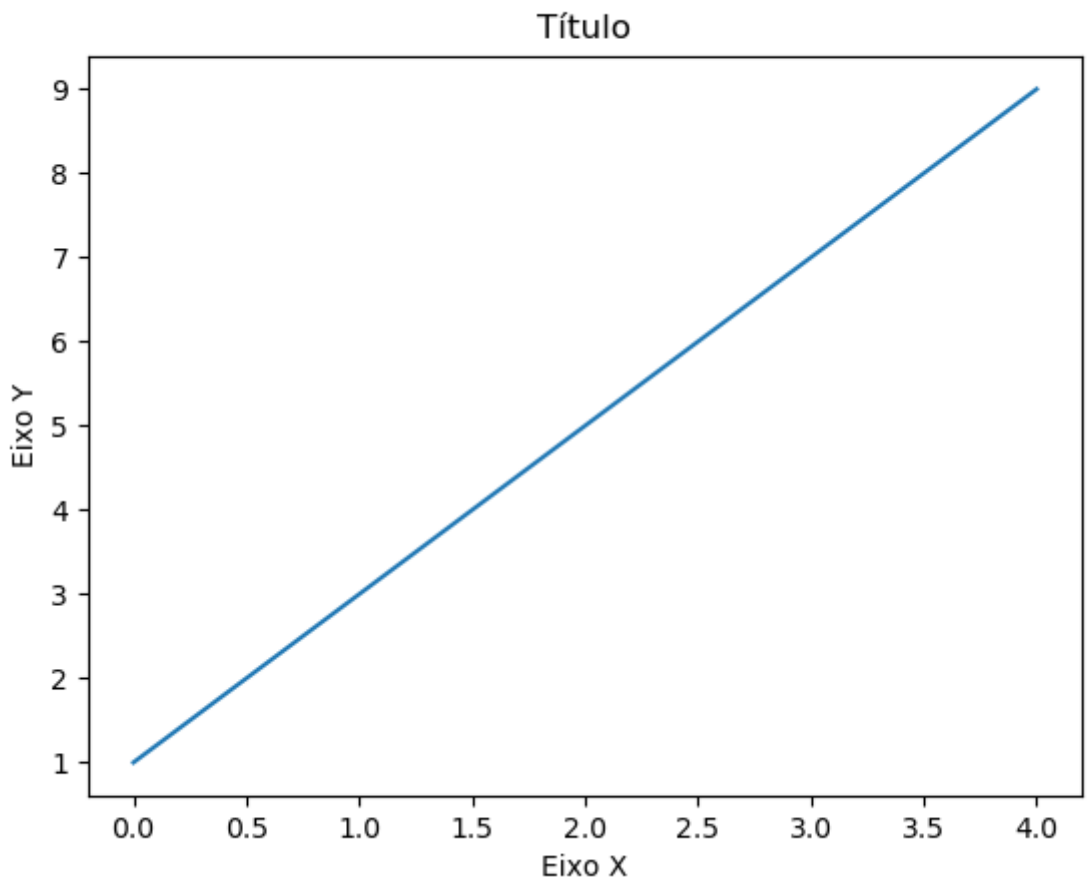
Para utilizar o matplotlib, é muito comum importá-lo com o nome `plt` utilizando a seguinte linha:

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

Vejamos um exemplo simples:

```
import matplotlib.pyplot as plt

dados_x = [0, 1, 2, 3, 4]
dados_y = [1, 3, 5, 7, 9]

plt.plot(dados_x, dados_y)
plt.title('Título')
plt.xlabel('Eixo X')
plt.ylabel('Eixo Y')

plt.show()
```

O código acima gera o seguinte gráfico:



Vamos entender o que cada linha faz:

- `plt.plot(dados_x, dados_y)` gera um gráfico de linha. Os dois parâmetros obrigatórios são coleções iteráveis, como listas, tuplas, numpy arrays etc correspondendo, respectivamente, aos valores do eixo X e os valores do eixo Y.
- `plt.title('Título')` insere um título no gráfico.
- `plt.xlabel('Eixo X')` e `plt.ylabel('Eixo Y')` colocam títulos nos eixos.
- `plt.show()` faz com que o gráfico apareça na tela.

> No Jupyter Notebook não é necessário utilizar `plt.show`. Ao invés disso, basta incluir a seguinte linha no código para que todos os gráficos sejam exibidos automaticamente:
>
> ```
> %matplotlib inline
> ```

2.1. Personalizando nossos gráficos

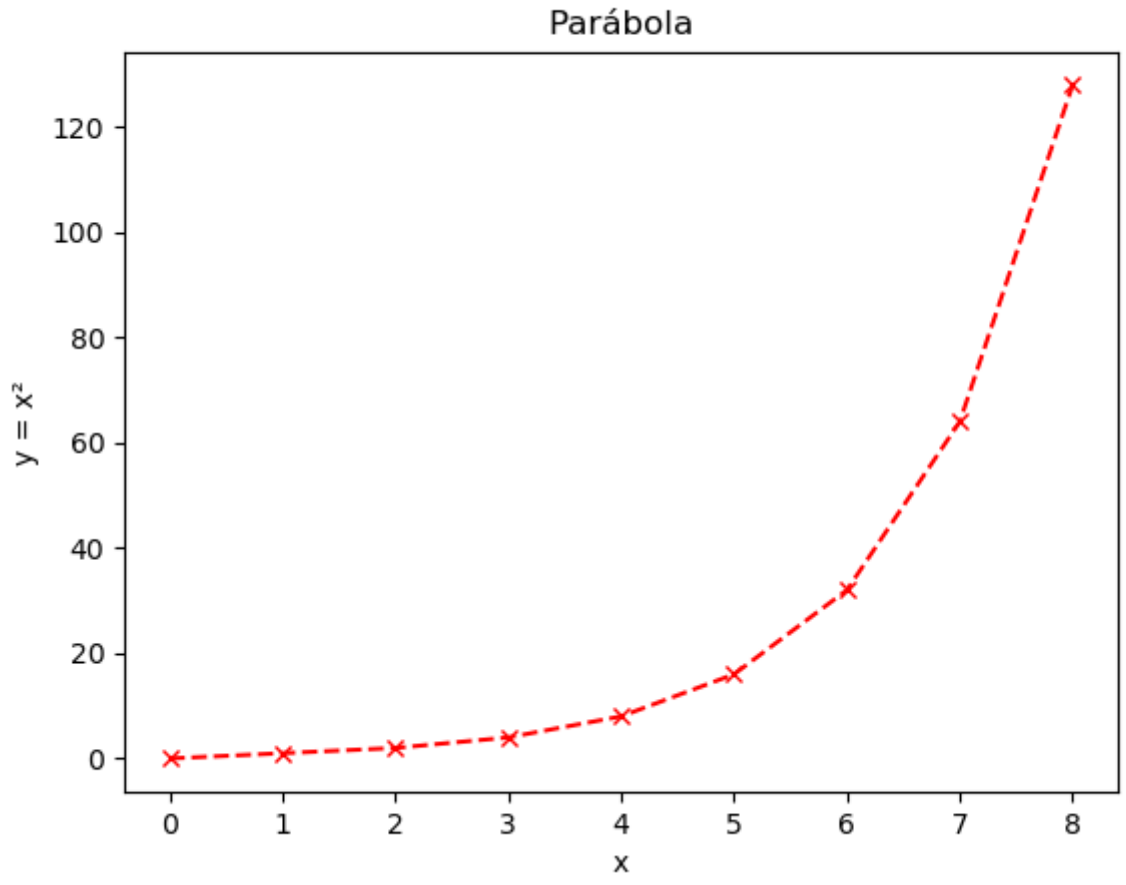
Podemos utilizar parâmetros opcionais para personalizar os gráficos. É possível ajustar estilo da linha, marcadores, cores, entre várias outras coisas. Vejamos alguns dos parâmetros mais comuns:

Parâmetro	Descrição
color	nomes de cor em inglês
alpha	transparência - valor entre 0.0 e 1.0, onde 1.0 é completamente visível, e 0.0 é completamente transparente
linestyle (ou ls)	estilo da linha: '-' (sólida), '--' (tracejada), '-.' (traço-ponto), ':' (pontilhada), '' (não desenhar)
marker	estilo dos pontos no gráfico: 'o' (bolinha), 'x' (formato de x), '+' (cruz), 's' (quadrado)

> A lista completa dos parâmetros e de seus possíveis valores pode ser encontrada no site oficial da matplotlib: https://matplotlib.org/3.2.1/api/_as_gen/matplotlib.pyplot.plot.html

Execute o código abaixo para visualizar um gráfico vermelho, tracejado e com marcadores em "x":

```
import matplotlib.pyplot as plt
# remover a linha abaixo se não estiver no Jupyter
%matplotlib inline

dados_x = [0, 1, 2, 3, 4, 5, 6, 7, 8]
dados_y = [0, 1, 2, 4, 8, 16, 32, 64, 128]

plt.plot(dados_x, dados_y, marker='x', color='red', ls='--')
plt.title('Parábola')
plt.xlabel('x')
plt.ylabel('y = x²')

# incluir a linha abaixo se não estiver no Jupyter
# plt.show()
```

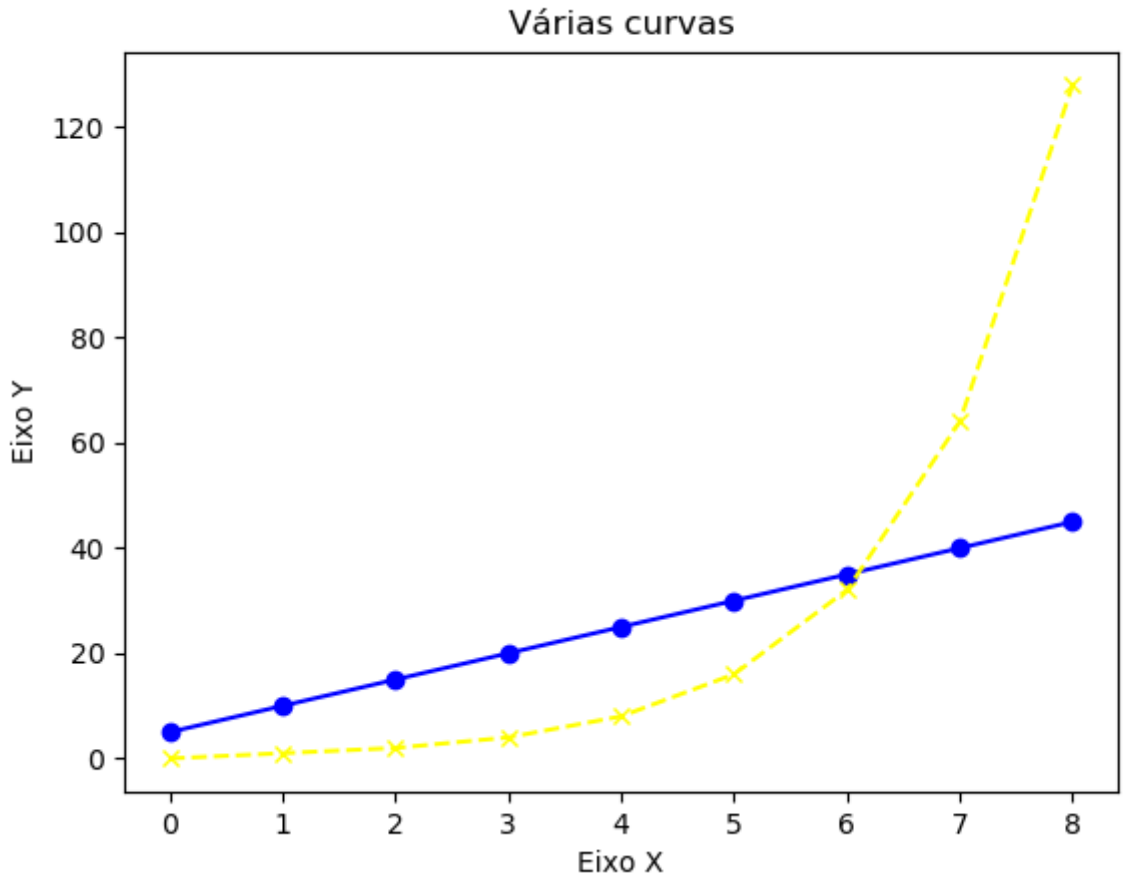


2.2. Múltiplos dados no mesmo gráfico

Podemos utilizar `plt.plot` diversas vezes para ir adicionando novas curvas aos nossos gráficos. Quando o gráfico for exibido, ambas as curvas aparecerão sobrepostas.

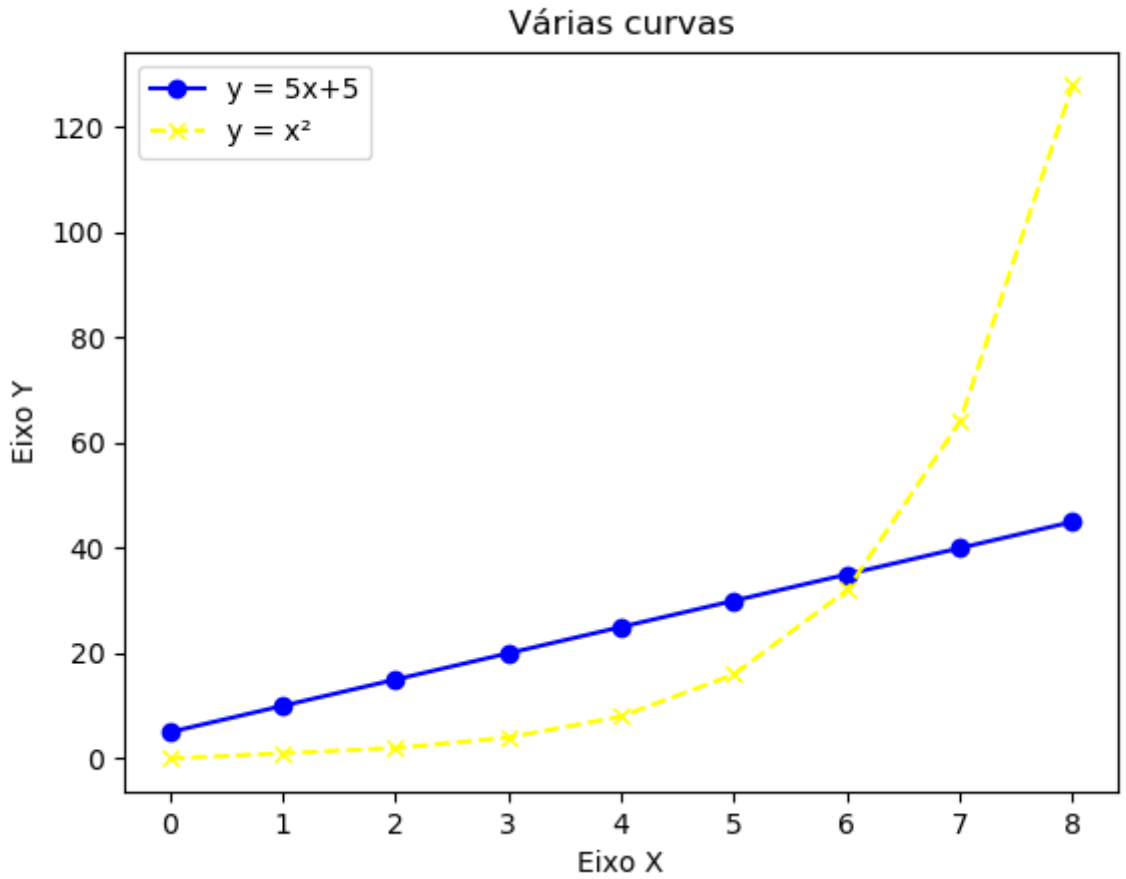
```
import matplotlib.pyplot as plt
# remover a linha abaixo se não estiver no Jupyter
%matplotlib inline

dados_x1 = [0, 1, 2, 3, 4, 5, 6, 7, 8]
dados_y1 = [5, 10, 15, 20, 25, 30, 35, 40, 45]

dados_x2 = [0, 1, 2, 3, 4, 5, 6, 7, 8]
dados_y2 = [0, 1, 2, 4, 8, 16, 32, 64, 128]

plt.plot(dados_x1, dados_y1, marker='o', color='blue', ls='-')
plt.plot(dados_x2, dados_y2, marker='x', color='yellow', ls='--')
```

```
plt.title('Várias curvas')
plt.xlabel('Eixo X')
plt.ylabel('Eixo Y')

# incluir a linha abaixo se não estiver no Jupyter
#plt.show()
```



Note que agora ficou difícil de identificar o que cada reta/curva representa, já que temos diferentes dados representados nos mesmos eixos. Podemos utilizar o `plt.legend` para acrescentar uma pequena legenda ao gráfico. Essa função recebe uma lista de strings. Adicione a linha abaixo ao código anterior:

```
plt.legend(['y = 5x+5', 'y = x²'])
```



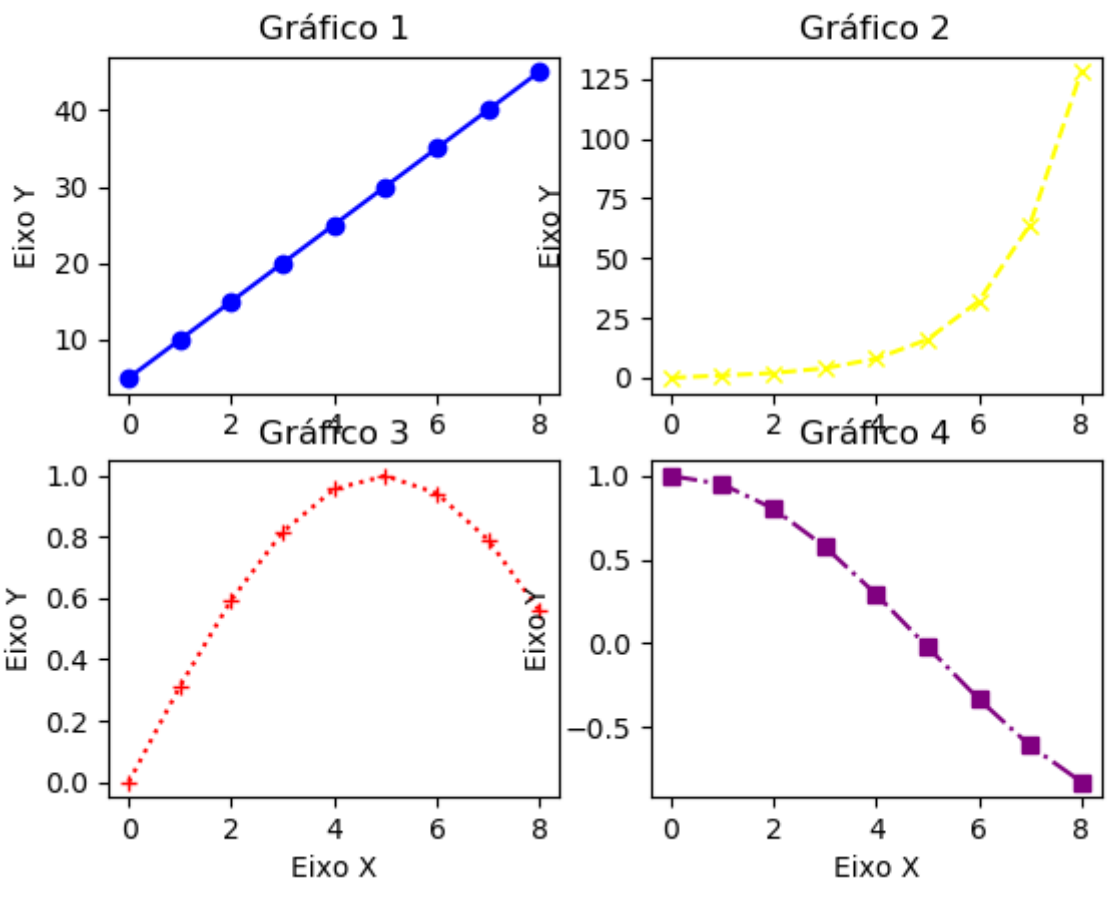
3. Múltiplos gráficos na mesma figura

Às vezes não estamos interessados em exibir todos os dados em um mesmo gráfico. Porém, gostaríamos que os diferentes gráficos estivessem próximos por serem dados relacionados, que gostaríamos de visualizar simultaneamente e comparar. Para isso utilizaremos o `plt.subplot`, que gera uma "tabela" de gráficos na figura.

Essa função como parâmetro um número com 3 dígitos: o primeiro diz quantas linhas teremos, o segundo diz quantas colunas, e o terceiro diz em qual gráfico estamos. O contador de gráficos começa em 1, sendo que o primeiro gráfico é o do canto superior esquerdo. O índice cresce da esquerda para a direita, e de cima para baixo.

Para editar cada gráfico individual, basta chamar a `plt.subplot` passando o índice do gráfico desejado. Daquele ponto em diante, todas as operações serão sobre aquele gráfico. Veja o exemplo abaixo, que plota 4 gráficos:

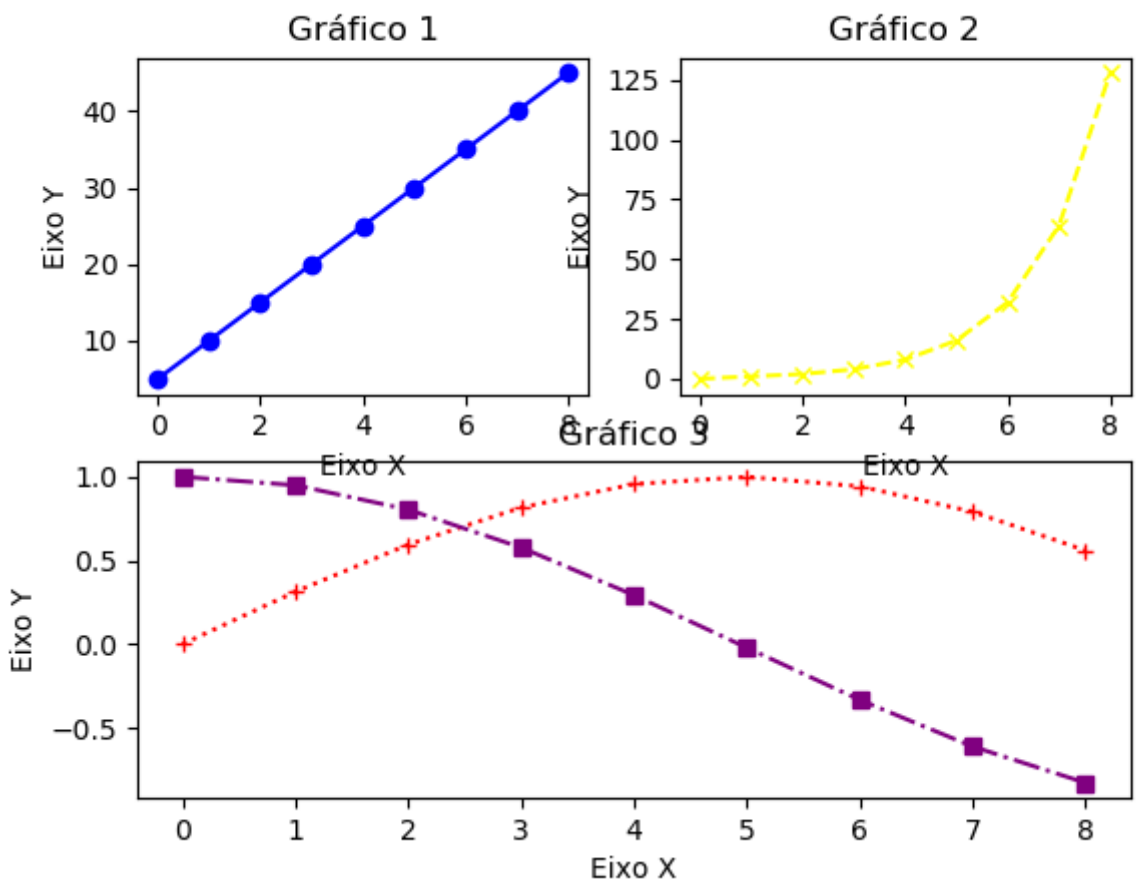
```
import matplotlib.pyplot as plt
# remover a linha abaixo se não estiver no Jupyter
%matplotlib inline

dados_x = [0, 1, 2, 3, 4, 5, 6, 7, 8]
dados_y1 = [5, 10, 15, 20, 25, 30, 35, 40, 45]
dados_y2 = [0, 1, 2, 4, 8, 16, 32, 64, 128]
dados_y3 = [0.0, 0.3131151331286599, 0.5947403868365128, 0.8165528078619212, 0.9562447986054508, 0.9997675844179418, 0
dados_y4 = [1.0, 0.9497151748844606, 0.8039178268116431, 0.5772707440818498, 0.29256774453111023, -0.02155869075601868,

# 2 linhas, 2 colunas, estamos no gráfico 1
plt.subplot(221)
plt.plot(dados_x, dados_y1, marker='o', color='blue', ls='-')
plt.title('Gráfico 1')
#plt.xlabel('Eixo X')
plt.ylabel('Eixo Y')

# 2 linhas, 2 colunas, estamos no gráfico 2
plt.subplot(222)
plt.plot(dados_x, dados_y2, marker='x', color='yellow', ls='--')
plt.title('Gráfico 2')
#plt.xlabel('Eixo X')
plt.ylabel('Eixo Y')

# 2 linhas, 2 colunas, estamos no gráfico 3
plt.subplot(223)
plt.plot(dados_x, dados_y3, marker='+', color='red', ls=':')
plt.title('Gráfico 3')
plt.xlabel('Eixo X')
plt.ylabel('Eixo Y')

# 2 linhas, 2 colunas, estamos no gráfico 4
plt.subplot(224)
plt.plot(dados_x, dados_y4, marker='s', color='purple', ls='-.')
plt.title('Gráfico 4')
plt.xlabel('Eixo X')
plt.ylabel('Eixo Y')

# incluir a linha abaixo se não estiver no Jupyter
#plt.show()
```



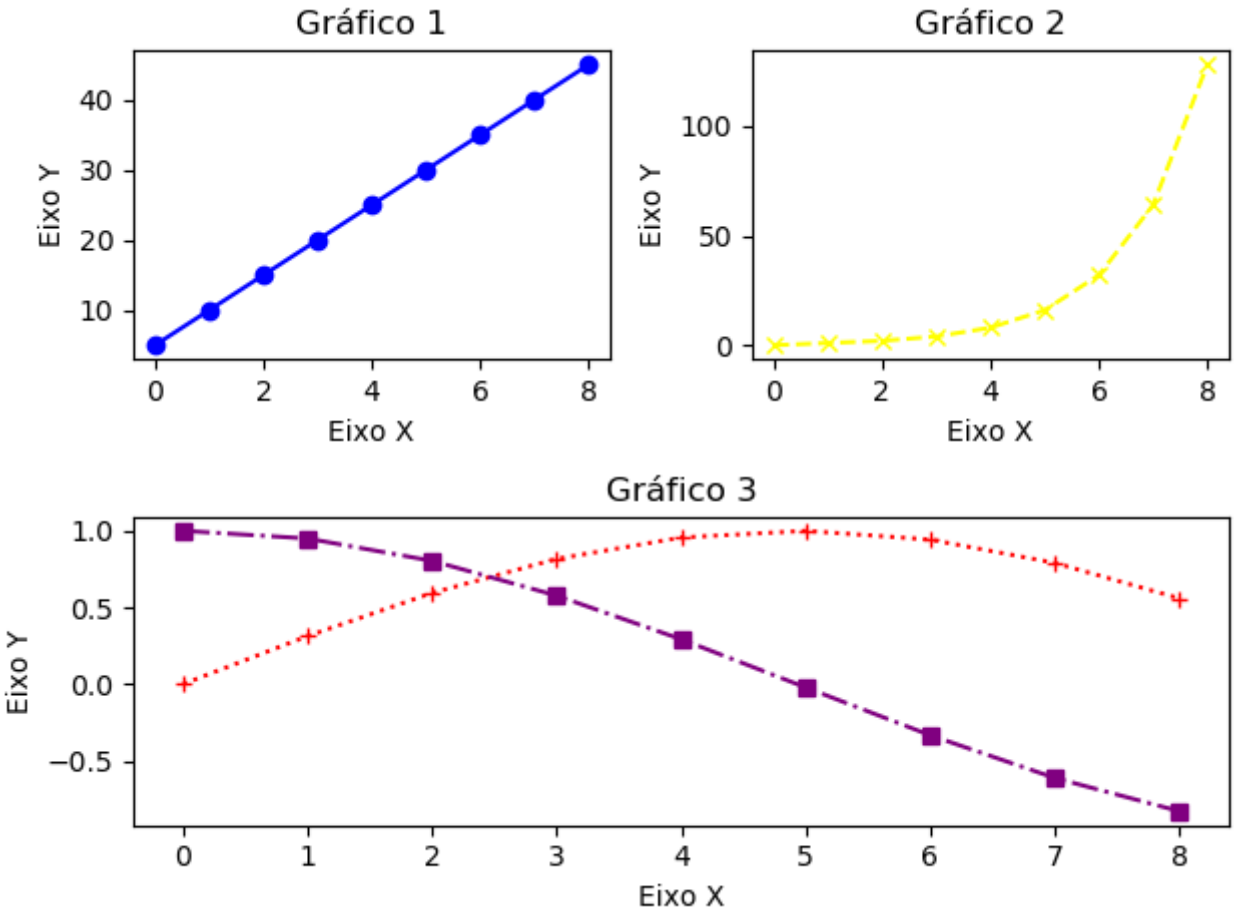
É possível fazer com que um gráfico ocupe múltiplas linhas ou múltiplas colunas. Imagine, por exemplo, que gostaríamos de plotar 3 gráficos, sendo 2 em uma mesma linha e um terceiro na linha seguinte sozinho, ocupando todo o espaço disponível. Podemos começar chamando `plt.subplot(212)` - ou seja, estamos dizendo que tem 2 linhas de 1 coluna cada e gostaríamos de começar pelo segundo gráfico. Após desenhá-lo, podemos ir para o primeiro e para o segundo gráfico fazendo `plt.subplot(221)` e `plt.subplot(222)`, respectivamente. Ou seja, após termos desenhado o gráfico da segunda linha, voltamos à primeira e redefinimos o número de colunas para 2! Agora, ao nos referimos ao gráfico 2, estamos falando do gráfico na segunda coluna da primeira linha, e não mais do gráfico solitário da segunda linha.

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

```
import matplotlib.pyplot as plt
# remover a linha abaixo se não estiver no Jupyter

%matplotlib inline

dados_x = [0, 1, 2, 3, 4, 5, 6, 7, 8]
dados_y1 = [5, 10, 15, 20, 25, 30, 35, 40, 45]
dados_y2 = [0, 1, 2, 4, 8, 16, 32, 64, 128]
dados_y3 = [0.0, 0.3131151331286599, 0.5947403868365128, 0.8165528078619212, 0.9562447986054508, 0.9997675844179418, 0
dados_y4 = [1.0, 0.9497151748844606, 0.8039178268116431, 0.5772707440818498, 0.29256774453111023, -0.02155869075601868,


# 2 linhas, 1 coluna, estamos no gráfico 3
# note que falamos para o Python que estamos no gráfico 2
# isso ocorre porque estamos afirmando que teremos 2 linhas de 1 coluna cada
# o que será o nosso gráfico 3, por hora, é visto pelo Python como gráfico 2
plt.subplot(212)
plt.plot(dados_x, dados_y3, marker='+', color='red', ls=':')
plt.plot(dados_x, dados_y4, marker='s', color='purple', ls='-.')
plt.title('Gráfico 3')
plt.xlabel('Eixo X')
plt.ylabel('Eixo Y')

# 2 linhas, 2 colunas, estamos no gráfico 1
# note que antes havíamos dito para o Python que teria 1 coluna, e agora falamos em 2 colunas...
plt.subplot(221)
plt.plot(dados_x, dados_y1, marker='o', color='blue', ls='-')
plt.title('Gráfico 1')
plt.xlabel('Eixo X')
plt.ylabel('Eixo Y')

# 2 linhas, 2 colunas, estamos no gráfico 2
# ué, mas já não havíamos usado o gráfico 2 antes?
# agora que redefinimos o número de colunas para 2, o Python considera a posição 2
# como sendo a segunda coluna da primeira linha...
# nosso antigo gráfico 2 agora é visto pelo Python como gráfico 3
plt.subplot(222)
plt.plot(dados_x, dados_y2, marker='x', color='yellow', ls='--')
plt.title('Gráfico 2')
plt.xlabel('Eixo X')
plt.ylabel('Eixo Y')

# incluir a linha abaixo se não estiver no Jupyter
#plt.show()
```



O excesso de informação na mesma figura "encavalou" o título dos gráficos inferiores com os eixos dos gráficos superiores. Podemos consertar isso! Inclua a linha `plt.tight_layout()` logo antes de `plt.show()` e veja o resultado:



4. Ajustes nos eixos

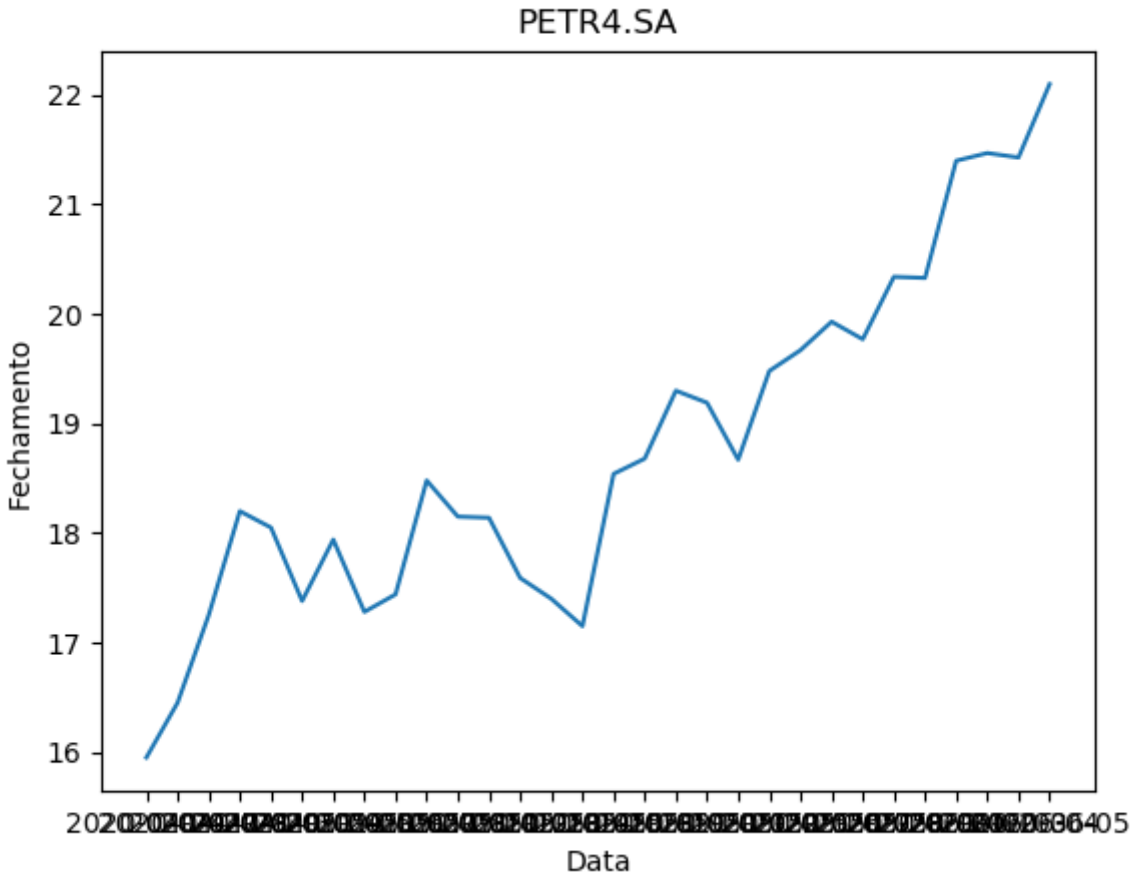
Agora que vimos o `plt.tight_layout`, convém falar um pouco sobre os eixos.

Além da curva desenhada, é possível personalizar também os eixos: podemos determinar a posição individual das marcações (*ticks*), os rótulos que aparecerão nas marcações e a rotação dos rótulos.

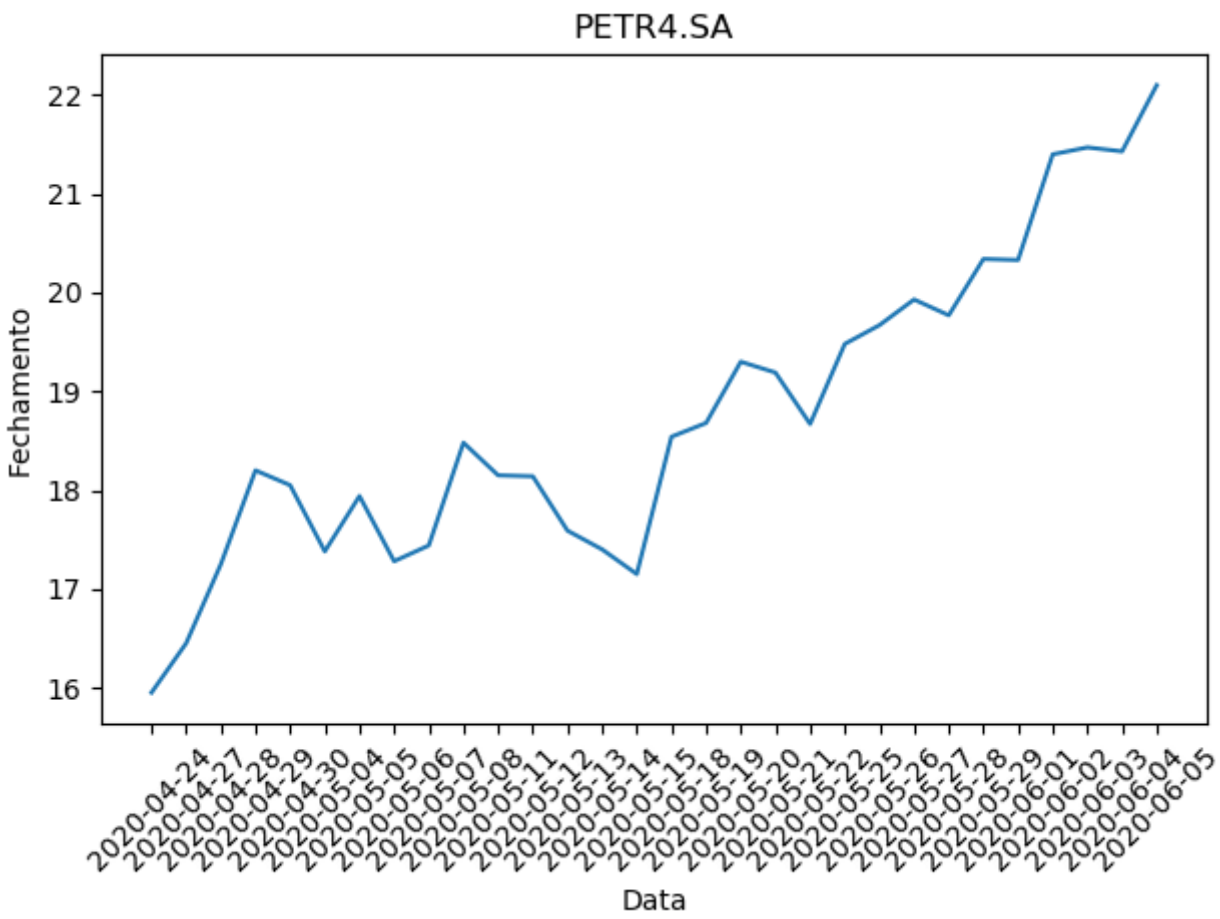
Considere o exemplo abaixo:

```
import matplotlib.pyplot as plt
# remover a linha abaixo se não estiver no Jupyter
%matplotlib inline

datas = ['2020-04-24', '2020-04-27', '2020-04-28', '2020-04-29', '2020-04-30',
'2020-05-04', '2020-05-05', '2020-05-06', '2020-05-07', '2020-05-08', '2020-05-11',
'2020-05-12', '2020-05-13', '2020-05-14', '2020-05-15', '2020-05-18', '2020-05-19',
'2020-05-20', '2020-05-21', '2020-05-22', '2020-05-25', '2020-05-26', '2020-05-27',
'2020-05-28', '2020-05-29', '2020-06-01', '2020-06-02', '2020-06-03', '2020-06-04', '2020-06-05']
fechamentos = [15.95, 16.45, 17.25, 18.2, 18.05, 17.38, 17.94, 17.28, 17.44, 18.48, 18.15, 18.14,
17.59, 17.4, 17.15, 18.54, 18.68, 19.3, 19.19, 18.67, 19.48, 19.67, 19.93, 19.77, 20.34, 20.33,
21.4, 21.47, 21.43, 22.1]

plt.plot(datas, fechamentos)
plt.title('PETR4.SA')
plt.xlabel('Data')
plt.ylabel('Fechamento')
# incluir a linha abaixo se não estiver no Jupyter
#plt.show()
```

A informação do eixo X é um conjunto de datas. Porém, por falta de espaço, elas acabam ficando ilegíveis:



Após as duas linhas onde ajustamos *xlabel* e *ylabel*, adicione a seguinte linha: `plt.xticks(rotation = 45)`. Ela irá rotacionar em 45 graus as informações do eixo X. Acrescente também o `plt.tight_layout()` para que haja espaço suficiente na vertical para comportar as datas. Vejamos o resultado:



> Falaremos um pouquinho mais sobre a personalização dos *ticks* no próximo capítulo, ao explicar outros tipos de gráficos.