**Trabalhando com Matplotlib**

* O matplotlib é uma biblioteca de gráficos 2D.
* Ideal para plotagem de gráficos **simples**.
* Você consegue plotar gráficos interessantes com poucas linhas de comando.
* Aumenta a produtividade do Cientista de dados.
* Fornece uma interfaçe próxima do Matlab, o que é interessante para gráficos com expressões matemáticas.
* A maior desvantagem é o **layout**, o seu padrão não entrega gráficos tão atraentes.
* Você vai precisar gastar um tempinho a mais para customizar e deixar seus gráficos com um visual bacana.

**Quando utilizar?**

* Útil para análise e exploração de dados.

**Instalação da Biblioteca**

* Caso não esteja usando anaconda instale o matplotlib com o gerenciador de pacotes pip ou usando o conda.

!pip install matplotlib

ou

conda install matplotlib

Se estiver usando o Anaconda, o matplotlib já está disponível por padrão.

Talvez seja necessário usar o %matplotlib inline ou %matplotlib notebook para que os gráficos sejam renderizados no jupyter notebook.

#Importando o pyplot

from matplotlib import pyplot as plt

No pacote pyplot se encontra todos os recursos dessa biblioteca.

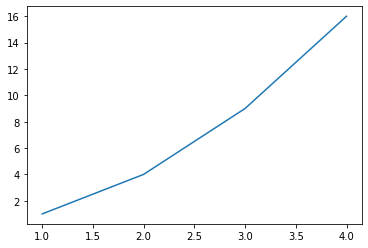
**Gráfico de Linhas**

* Gráfico normalmente utilizado para visualização de dados em uma linha do tempo.
* Normalmente utilizado com eixo X relacionado a dados temporais e eixo Y a dados quantitativos.
* Cuidado ao plotar várias linhas, a leitura pode ficar prejudicada.

# Eixo\_x, Eixo\_y

plt.plot([1, 2, 3, 4], [1, 4, 9, 16])

plt.show()



O eixo x recebe a primeira lista de valores e o y a segunda lista.

**Definindo label aos eixos**:

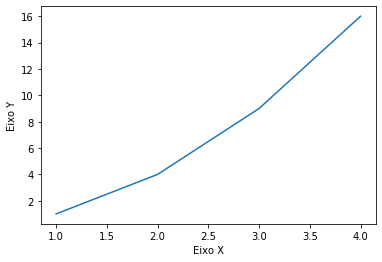
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4],[1, 4, 9, 16])

plt.ylabel('Eixo Y')

plt.xlabel('Eixo X')

plt.show()



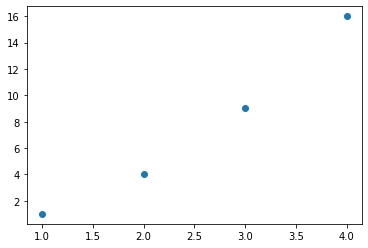
Permiti definir um rótulo qualquer para cada um dos eixos.

**Alterando a forma dos pontos**:

# plot(x, y)

plt.plot([1, 2, 3, 4], [1, 4, 9, 16],'o')

plt.show()



Anteriormente usamos o marcador padrão da biblioteca que são as linhas, mas podemos plotar esses mesmos dados usando pontos, é só usar o **‘o’** após a lista de valores.

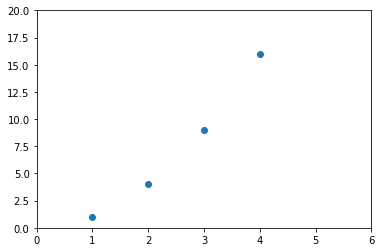
Esse marcador é muito utilizado em gráficos de dispersão.

**Definindo limites de visualização dos eixos x e y**:

plt.plot([1, 2, 3, 4], [1, 4, 9, 16],'o')

plt.axis([0, 6, 0, 20]) # [xmin, xmax, ymin, ymax]

plt.show()



o método **axis** permite customizar a escala dos eixos, os dois primeiros valores são do eixo x, indicando que a escala começa em 0 e termina em 6.

O funcionamento é o mesmo para o eixo y, a escala começa em 0 e termina em 20.

**Importa a biblioteca numpy**:

import numpy as np

**Plot duas séries de dados, no caso x1 e y2 e x3 e y3**:

x1=np.array([1,2,3,4,5,6,7,8,9,11])

x2 = x1

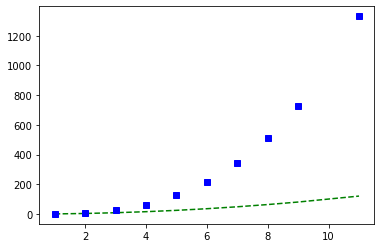
y2 = x1\*\*2 # Numero elevado ao quadrado

x3 = x1

y3 = x1\*\*3 # Numero elevado cubo

plt.plot(x1, y2 ,'g--', x2, y3, 'bs')

plt.show()



Então temos:

**x1**= [ 1 2 3 4 5 6 7 8 9 11]

**y2** = [ 1 4 9 16 25 36 49 64 81 121]

**x2**= [ 1 2 3 4 5 6 7 8 9 11]

**y3** = [ 1 8 27 64 125 216 343 512 729 1331]

No gráfico a plotagem fica da seguinte maneira: o tracejado da cor verde (g de green) representa os valores da séria x1,y2 e os quadradinhos de cor azul representam os valores os valores da série x2,y3.

**Plotando dados categóricos**

* Trabalhando com gráficos de barras.
* Ideal para quando estamos trabalhando com **comparação de valores**.
* Normalmente quer se visualizar frequência ou percentual de valores.

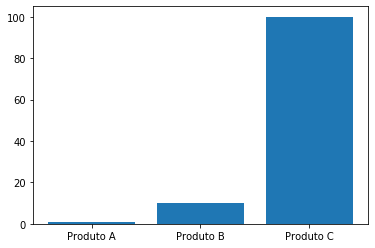
**Plotando gráfico de barras verticais o gráfico de colunas**:

grupos = ['Produto A', 'Produto B', 'Produto C']

valores = [1, 10, 100]

plt.bar(grupos, valores)

plt.show()



Para plotar um gráfico de barras usamos o método **bar** que recebe 2 parâmetros, a lista com as categorias dos produtos (x) e uma lista com os seus respectivos valores (y).

**Plota gráficos de Barras juntas**

**Customizando layouts**:

# Define o stilo para ggplot

plt.style.use("ggplot")

Consulte a galeria de estilos do matplotlib: [https://](https://tonysyu.github.io/raw_content/matplotlib-style-gallery/gallery.html)[tonysyu.github.io](https://tonysyu.github.io/raw_content/matplotlib-style-gallery/gallery.html)[/raw\_content/matplotlib-style-gallery/gallery.html](https://tonysyu.github.io/raw_content/matplotlib-style-gallery/gallery.html).

# Define as configurações dos plots

# Cada plot terá o mesmo tamanho de figuras (10,5)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,5))

# Dados para cada subplot

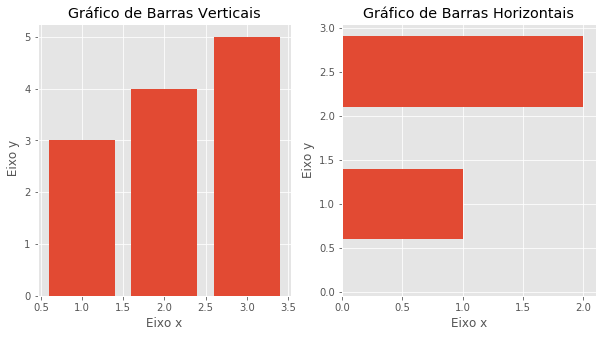
ax1.bar([1,2,3],[3,4,5])

ax2.barh([0.5,1,2.5],[0,1,2])

ax1.set(title="Gráfico de Barras Verticais", xlabel="Eixo x", ylabel=" Eixo y")

ax2.set(title="Gráfico de Barras Horizontais", xlabel="Eixo x", ylabel="Eixo y")

plt.show()



No método **subplot**informamos que queremos uma única figura com dois elementos ax1 para o primeiro gráfio e ax2 para o segundo, e essa figura vai ter uma largura de 10 polegadas e uma altura de 5 polegadas.

O método **barh** é responsável por gerar gráficos de barras horizontais. Cada gráfico tem seu próprio título e um rótulo para cada eixo.

Note que não temos mais aquele fundo branco, agora temos um fundo com grids, essa é umas das características do estilo que estamos utilizando.

**Matplotlib funciona também com cores em RGB. Para fazer isso usamos o parâmetro \*color\***:

# Define as configurações dos plots

# Cada plot terá o mesmo tamanho de figuras (10,5)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,5))

# Dados para cada subplot

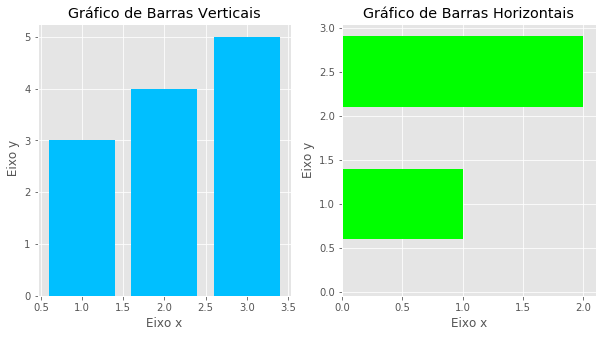
ax1.bar([1,2,3],[3,4,5],color='#00BFFF')

ax2.barh([0.5,1,2.5],[0,1,2], color='#00FF00')

ax1.set(title="Gráfico de Barras Verticais", xlabel="Eixo x", ylabel=" Eixo y")

ax2.set(title="Gráfico de Barras Horizontais", xlabel="Eixo x", ylabel="Eixo y")

plt.show()



Temos o mesmo gráfico, a diferença é que definimos uma cor para cada gráfico usando um código RGB.

Não sabe qual código usar? Então aqui vai mais um link para te ajudar nessa tarefa <https://celke.com.br/artigo/tabela-de-cores-html-nome-hexadecimal-rgb>.

**Gráfico de barras aninhadas**

# Quantidade de vendas para o Produto A

valores\_produto\_A = [6,7,8,4,4]

# Quantidade de vendas para o Produto B

valores\_produto\_B = [3,12,3,4.1,6]

# Cria eixo x para produto A e produto B com uma separação de 0.25 entre as barras

x1 = np.arange(len(valores\_produto\_A))

x2 = [x + 0.25 for x in x1]

# Plota as barras

plt.bar(x1, valores\_produto\_A, width=0.25, label = 'Produto A', color = 'b')

plt.bar(x2, valores\_produto\_B, width=0.25, label = 'Produto B', color = 'y')

# coloca o nome dos meses como label do eixo x

meses = ['Agosto','Setembro','Outubro','Novembro','Dezembro']

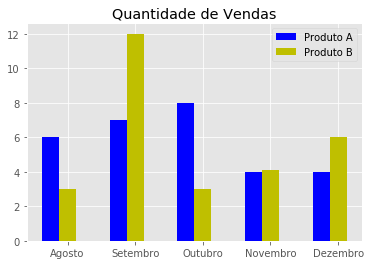
plt.xticks([x + 0.25 for x in range(len(valores\_produto\_A))], meses)

# inseri uma legenda no gráfico

plt.legend()

plt.title("Quantidade de Vendas")

plt.show()



Podemos notar facilmente que no mês de setembro o grupo A teve 12 produtos vendidos enquanto o grupo B teve 7.

Fizemos algumas customizações como o espaço entre as barras e a largura delas, o ponto mais importante dessa customização é a legenda que criamos indicando qual a cor de cada grupo.

Para o parâmetro color passamos as inicias das cores que queremos em inglês, ‘b’ (blue) e ‘y’ (yellow).

**Gráfico de barras horizontais**

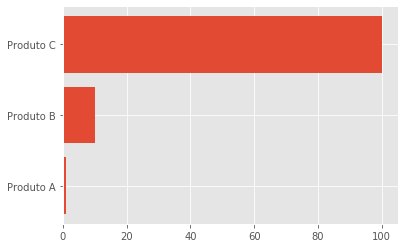
* Gráfico muito usado quando queremos fazer **comparações**.
* Pode ser usado para expressar uma ideia cronológica.
* Leitura muito simples e intuitiva.
* Funciona bem quando os labels dos eixos são longos.

grupos = ['Produto A', 'Produto B', 'Produto C']

valores = [1, 10, 100]

plt.barh(grupos, valores)

plt.show()



**Rotacionando Labels**:

grupos = ['Produto A', 'Produto B', 'Produto C']

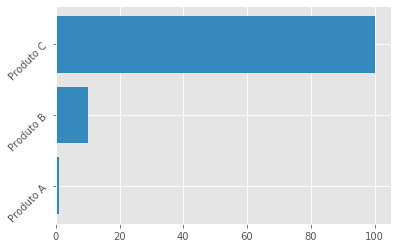
valores = [1, 10, 100]

plt.barh(grupos, valores)

plt.barh (grupos, valores)

plt.yticks(rotation=45)

plt.show()



Quando temos um gráfico que possui muitas barras os rótulos costumam se sobrepor, para evitar isso podemos simplesmente rotacionar os labels.

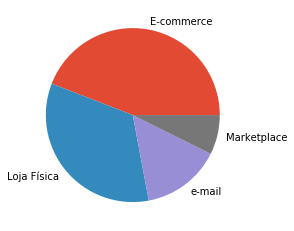
**Gráfico de Pizza**

vendas = [3000, 2300, 1000, 500]

labels = ['E-commerce', 'Loja Física', 'e-mail', 'Marketplace']

plt.pie(vendas, labels=labels)

plt.show()



No nosso exemplo estamos mostrando a proporção de vendas de cada categoria. Fica mais fácil de visualizar quais foram as categorias que tiveram um maior número de vendas.

Ele é muito utilizado para mostrar a proporção de sexo (feminino / masculino) e para faixa etária (criança, jovem, adulto, idoso).

Mas se você tiver muitos dados (muitas categorias por exemplo) evite usar esse tipo de gráfico, pois a visualização fica muito confusa.

**Customizando gráfico de pizza**:

* Adicionando informações de percentual, sombra, separação das partes e legenda.

vendas = [3000, 2300, 1000, 500]

labels = ['E-commerce', 'Loja Física', 'e-mail', 'Marketplace']

# define o nível de separabilidade entre as partes, ordem do vetor representa as partes

explode = (0.1, 0, 0, 0)

# define o formato de visualização com saída em 1.1%%, sombras e a separação entre as partes

plt.pie(vendas, labels=labels, autopct='%1.1f%%', shadow=True, explode=explode)

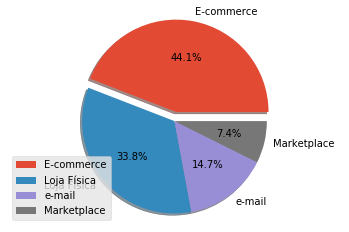
# inseri a legenda e a localização da legenda.

plt.legend(labels, loc=3)

# define que o gráfico será plotado em circulo

plt.axis('equal')

plt.show()



Colocamos a quantidade em valores percentuais dando destaque (**explode=explode**) para a categoria que teve o maior valor. O método**shadow=True**é responsável pelo sombreamento dando um aspecto de 3D.

A localização da legenda é definida dentro de um quadrante:

* loc=1 → canto superior direito
* loc=2 → canto superior esquerdo
* loc=3 → canto inferior esquerdo
* loc=1 → canto inferior direito

**Adiciona textos em gráficos**

# 100 valores no intervalo de 0 a 2.

x = np.linspace(0, 2, num=100)

# define a legenda e tamanho de linhas

plt.plot(x, x, label='linear')

plt.plot(x, x\*\*2, label='quadratico', linewidth=5)

plt.xlabel('x label')

plt.ylabel('y label')

plt.title("Gráfico de Linhas Simples")

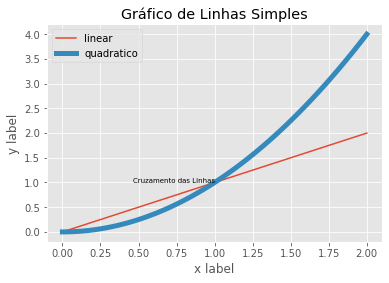
# Exibe a legenda e por padrão usa o label de cada plot.

plt.legend()

# Configurações do texto

plt.text(1.00, 1.0, "Cruzamento das Linhas", fontsize=8, horizontalalignment='right')

plt.show()



Plotamos duas linhas no mesmo gráfico com cores e espessuras diferentes para facilitar a visualização, além disso estamos informando que as linhas se cruzam no ponto 1.0 do eixo x e do eixo y através de um texto que também pode ser customizado.

**Plotando Histogramas**

* Histogramas são úteis quando queremos visualizar a distribuição de **uma** variável.
* Esse é um gráfico muito utilizado em estatítica descritiva.
* Facilita a visualização de dados extremos.
* É útil em tarefas de **análise** e **exploração** de dados.

**Gera dados aleatórios com média em torno de 0 e desvio padrão em torno de 1**:

d = np.random.normal(0, 1, size=500)

**Plota histograma com 10 bins**:

plt.hist(d, bins=10,color='r')

# Saída:

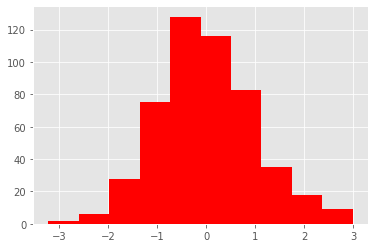
(array([ 2., 6., 28., 75., 128., 116., 83., 35., 18., 9.]),

array([-3.21494489, -2.59489258, -1.97484026, -1.35478795, -0.73473564,

-0.11468332, 0.50536899, 1.1254213 , 1.74547362, 2.36552593,

2.98557824]),

<a list of 10 Patch objects>)



Apesar de ser um gráfico bem simples e muito utilizado para mostrar a distribuição dos dados.

Temos os dados distribuídos em 10 barras agrupadas com uma maior concentração entre os valores de -1 e 0.