https://vncoder.vn/bai-hoc/contour-plot-do-thi-duong-bao--510

1. Figure Class:

matplotlib.figure chứa lớp Figure. Nó là một vùng chứa cấp cao nhất cho tất cả các phần tử plot. Đối tượng Figure được khởi tạo bằng cách gọi hàm figure () từ mô-đun pyplot -

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

Cần lưu ý với các tham số sau :

Figsize	**(chiều rộng, chiều cao) tính bằng inch**
Dpi	Dấu chấm trên inch
Facecolor	Figure patch facecolor
Edgecolor	Figure patch edge color
Linewidth	Chiều rộng đường cạnh

2. Axes Class :

Đối tượng Axes là vùng ảnh có không gian dữ liệu. Một hình nhất định có thể chứa nhiều Trục, nhưng một đối tượng Trục đã cho chỉ có thể nằm trong một Hình. Axes chứa hai (hoặc ba trong trường hợp là 3D) đối tượng Axis. Lớp Axes và các hàm thành phần của nó là điểm đầu vào chính để làm việc với giao diện OO.

Đối tượng Axes được thêm vào hình bằng cách gọi phương thức add_axes (). Nó trả về đối tượng trục và thêm một trục ở vị trí trực tràng [trái, dưới, rộng, cao] nơi tất cả các đại lượng đều ở dạng nhỏ hơn chiều rộng và chiều cao của hình.

a. Parameter :

Sau đây là tham số cho lớp Axes

- Rect - Một chuỗi 4 chiều dài gồm các đại lượng [trái, dưới, rộng, cao]

b. Legend :

Phương thức **legend()** của lớp axes bổ sung thêm chú giải cho hình vẽ . Nó có ba tham số -

ax.legend(handles, labels, loc)
Trong đó các label là một chuỗi các chuỗi và xử lý liêu tục Line2D hoặc Patch. loc có thể là một chuỗi hoặc một số nguyên xác định vị trí chú giải.

Location string	**Location code**
Best	0
upper right	1
upper left	2
lower left	3
lower right	4
Right	5
Center left	6
Center right	7
lower center	8
upper center	9
Center	10

axes.plot()

Đây là phương thức cơ bản của lớp trục vẽ các giá trị của mảng này so với mảng khác dưới dạng dòng hoặc điểm đánh dấu. Phương thức plot () có thể có đối số chuỗi định dạng tùy chọn để chỉ định màu, kiểu và kích thước của dòng và điểm đánh dấu.

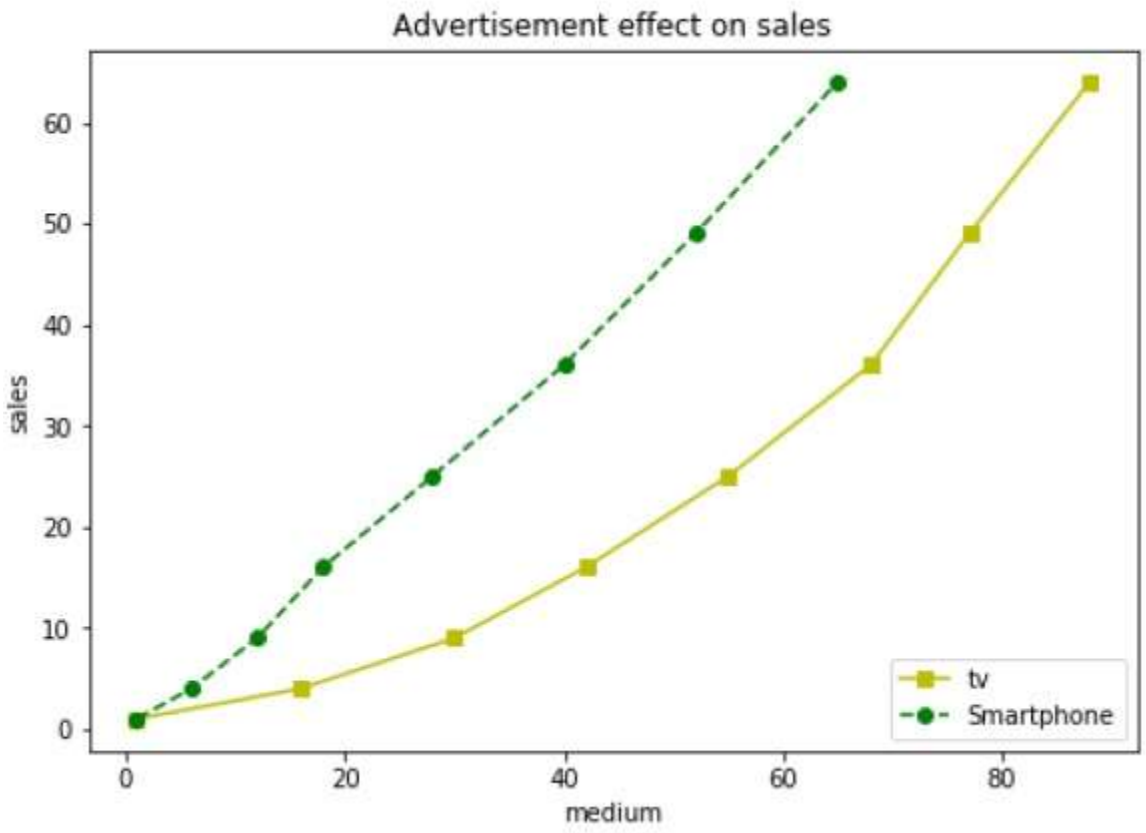
Mã màu :

Character	Color
‘b’	Blue
‘g’	Green
‘r’	Red
‘b’	Blue
‘c’	Cyan
‘m’	Magenta
‘y’	Yellow
‘k’	Black
‘b’	Blue
‘w’	White

Character	Description
'.'	Point marker
'o'	Circle marker
'x'	X marker
'D'	Diamond marker
'H'	Hexagon marker
's'	Square marker
'+'	Plus marker

Character	Description
'-'	Solid line
'—'	Dashed line
'-.'	Dash-dot line
':'	Dotted line
'H'	Hexagon marker

Ví dụ sau cho thấy chi phí quảng cáo và số liệu bán hàng của TV và điện thoại thông minh dưới dạng biểu đồ đường thẳng. Đường thể hiện TV là đường liền nét với màu vàng và các điểm đánh dấu hình vuông, trong khi dòng điện thoại thông minh là đường đứt nét với màu xanh lá cây và điểm đánh dấu hình tròn.



Multiplots - Matplotib Cơ Bản
Trong bài này, ta sẽ tìm hiểu cách tạo nhiều plot trên cùng một khung vẽ.

Hàm subplot () trả về đối tượng trục tại một vị trí lưới nhất định. Call signature của hàm này là -

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

Trong hình hiện tại, hàm tạo và trả về một đối tượng Axes, tại chỉ số vị trí của lưới các nrows theo ncolsaxes. Các index đi từ 1 đến nrows * ncols, tăng dần theo thứ tự hàng-chính. Nếu nrows, ncol và chỉ mục đều nhỏ hơn 10. Các chỉ mục cũng có thể được cung cấp dưới dạng số đơn, nối, thứ tự

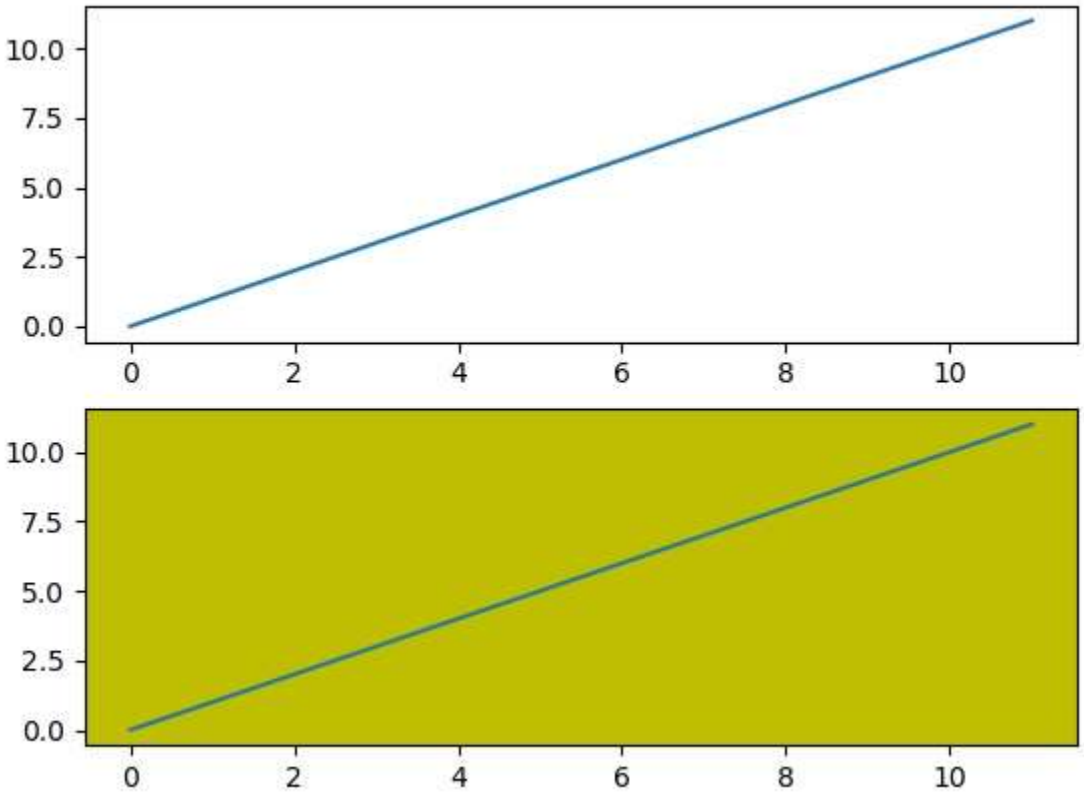
Ví dụ: subplot (2, 3, 3) và subplot (233) đều tạo Axes ở góc trên cùng bên phải của hình hiện tại, chiếm một nửa chiều cao hình và một phần ba chiều rộng hình.

Việc tạo một subplot sẽ xóa mọi subplot tồn tại trước đó chồng lên nó ngoài việc chia sẻ ranh giới.

```
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 column.
#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))
```

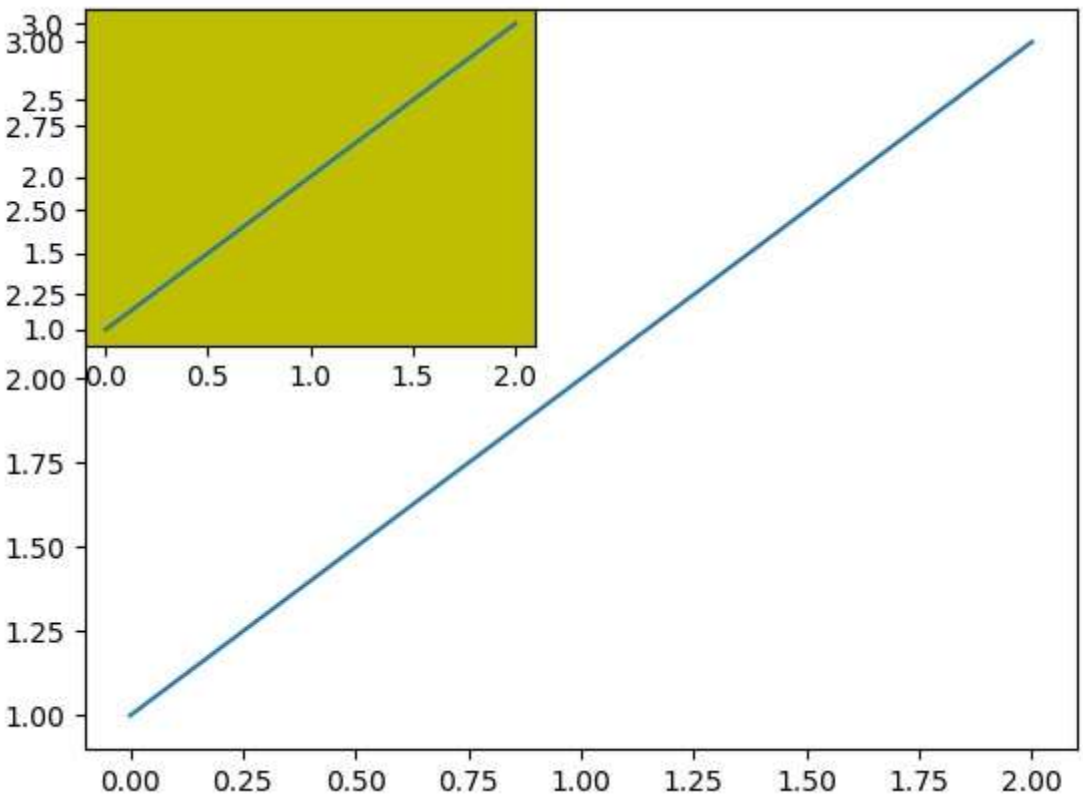
Kết quả như sau :



Hàm add_subplot () của lớp figure sẽ không ghi đè lên phần hiện có -

```
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])
```

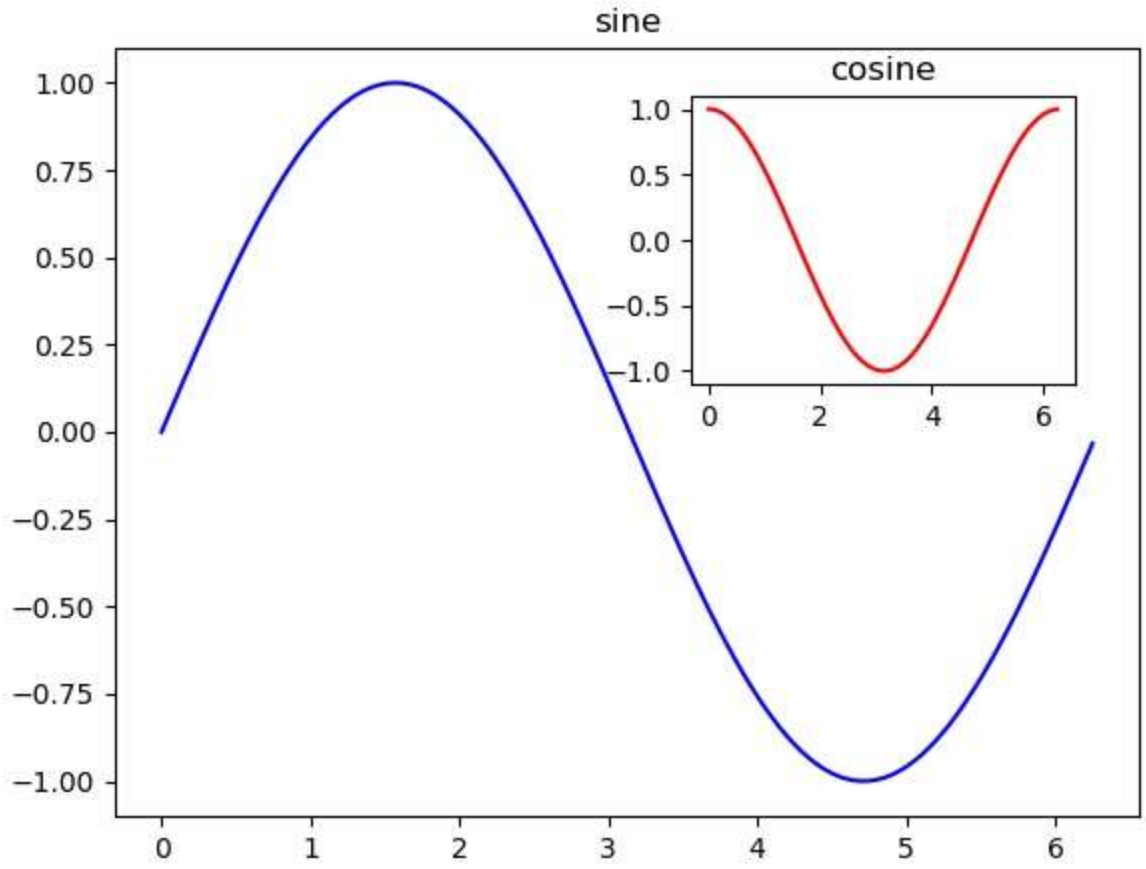
Khi dòng mã trên được thực thi, nó tạo ra kết quả sau:



Ta có thể thêm một plot chèn trong cùng một hình bằng cách thêm một trục khác trong cùng khung vẽ.

```
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()
```

Kết quả sau khi thực thi đoạn code :



BÀI TIẾP THEO: HÀM SUBPLOTS() VÀ SUBPLOT2GRID() >>

Bài 10: Hàm Subplots() và Subplot2grid() - Matplotib Cơ Bản

1. Subplots() :

Matplotlib'spyplot API có một hàm tiện lợi được gọi là subplots () hoạt động như một trình bao bọc tiện ích và giúp tạo các bố cục chung của các subplot, bao gồm cả figure bao quanh, trong một lệnh gọi.

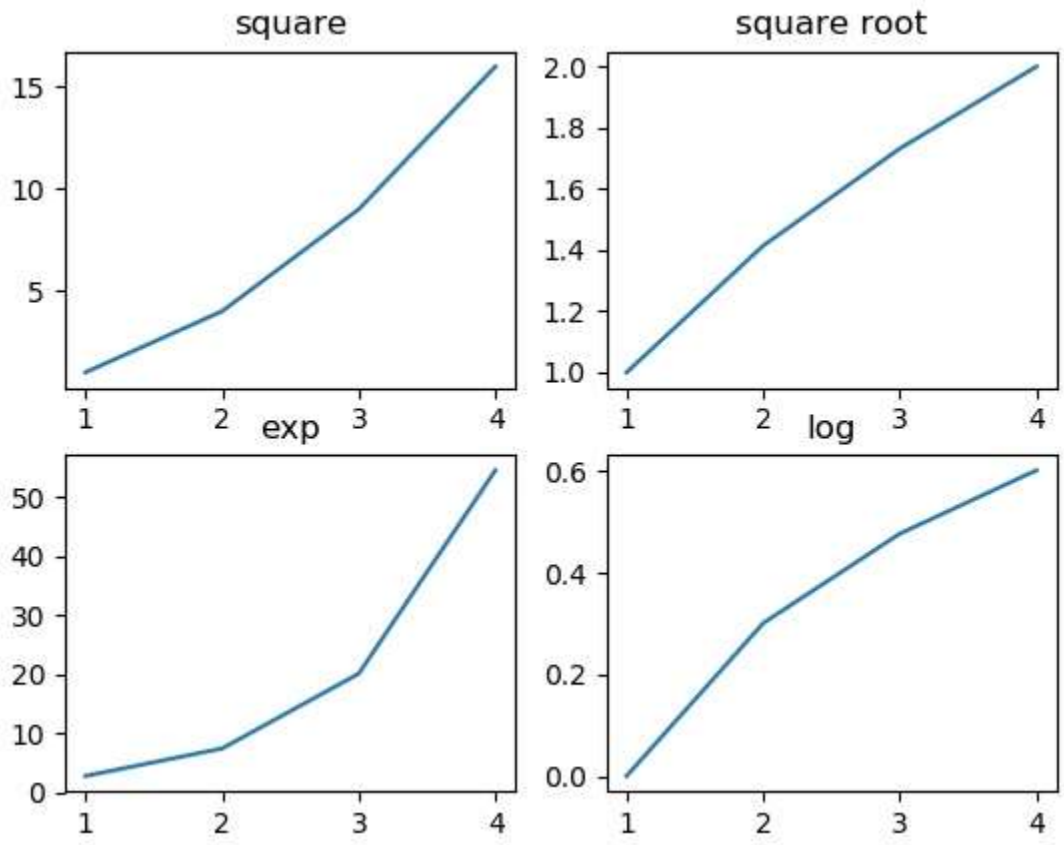
```
Plt.subplots(nrows, ncols)
```

Hai đối số nguyên cho hàm này chỉ định số hàng và cột của lưới subplot. Hàm trả về một figure và một bộ chứa các trục bằng nrows * ncols. Mỗi trục có thể truy cập được bằng index của nó. Ở đây tôi tạo một subplot gồm 2 hàng x 2 cột và hiển thị 4 ô khác nhau trong mỗi subplot.

```
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()
```

Kết quả như sau :



2. Subplot2grid() ;

Hàm này mang lại sự linh hoạt hơn trong việc tạo một trục tại một vị trí cụ thể của lưới. Nó cũng cho phép trục được kéo dài qua nhiều hàng hoặc cột.

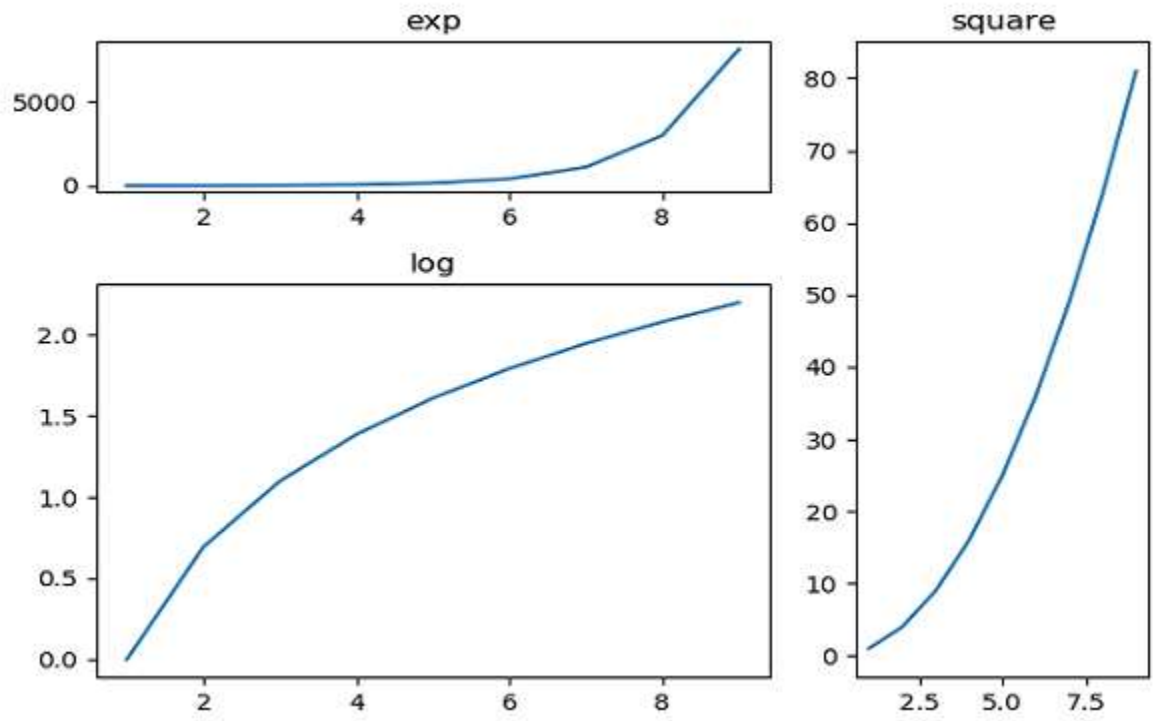
```
Plt.subplot2grid(shape, location, rowspan, colspan)
```

Trong ví dụ sau, một lưới 3X3 của figure được lấp đầy bởi các trục có kích thước khác nhau trong các khoảng hàng và cột, mỗi ô hiển thị một biểu đồ khác nhau.

```
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()
```

Kết quả :



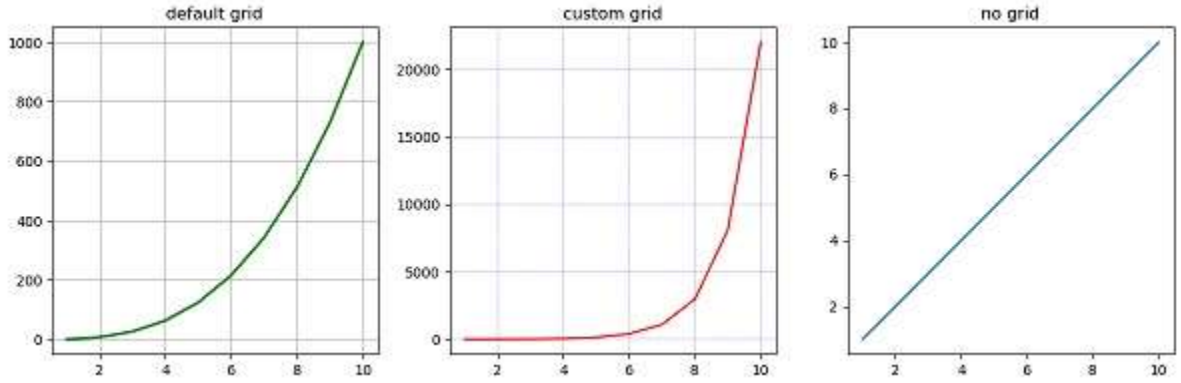
BÀI TIẾP THEO: GRIDS >>

Bài 11: Grids - Matplotib Cơ Bản
Đăng bởi: Admin | Lượt xem: 2705 | Chuyên mục: AI

1. Khái niệm cơ bản :

Hàm Grid () của trục được đặt hiển thị bên trong lưới để bật hoặc tắt. Bạn cũng có thể hiển thị các tích tắc chính / phụ (hoặc cả hai) của lưới. Ngoài ra, các thuộc tính màu sắc, kiểu đường kẻ và độ rộng đường thẳng có thể được đặt trong hàm 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()
```



Các tham số cần chú ý :

- **b** : bool hoặc None
- **which** : {'major', 'minor', 'both'} Các đường lưới để áp dụng các thay đổi trên.
- **axis** : {'both', 'x', 'y'} Trục áp dụng các thay đổi trên.

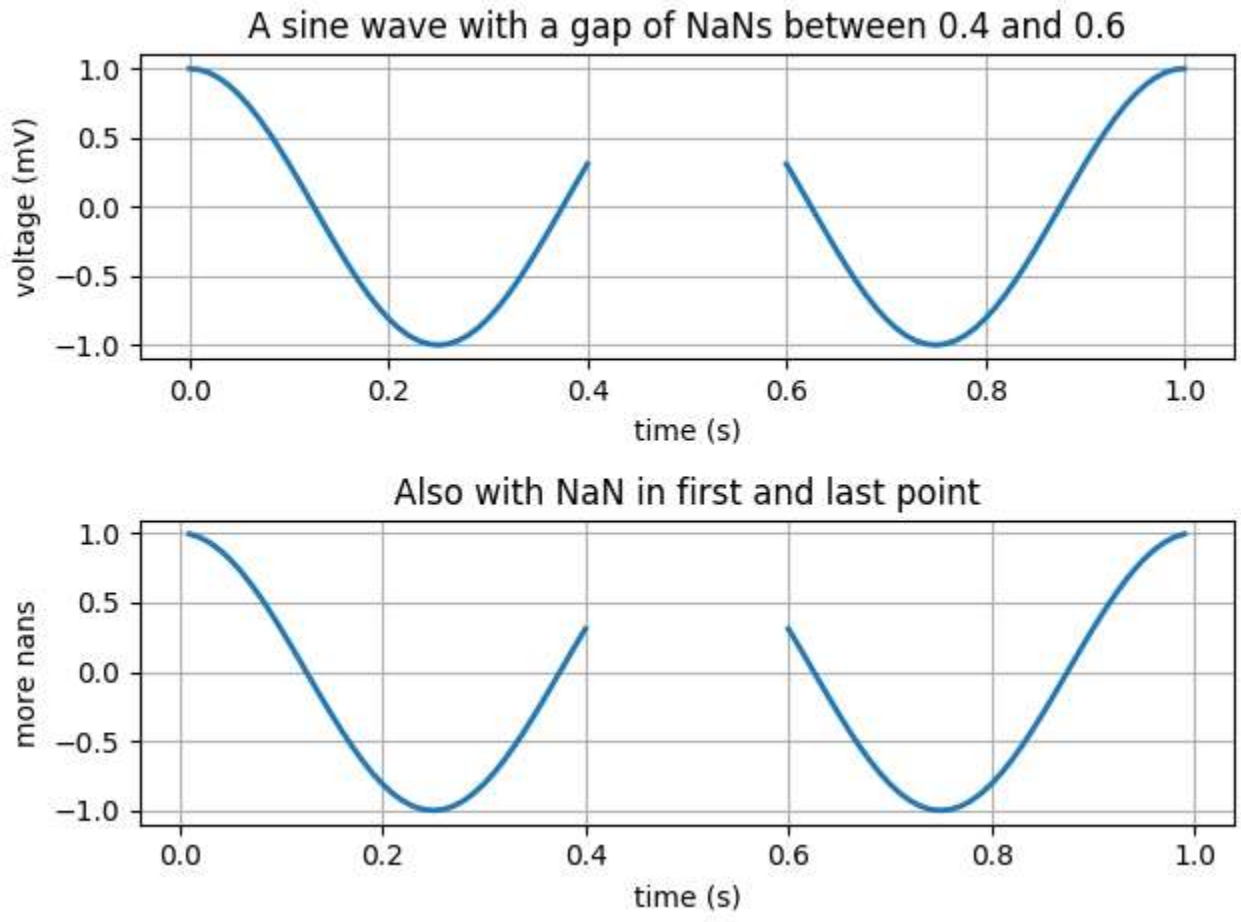
2. Các ví dụ :

Ví dụ 1 : Nan Test

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

t = np.arange(0.0, 1.0 + 0.01, 0.01)
s = np.cos(2 * 2*np.pi * t)
t[41:60] = np.nan

plt.subplot(2, 1, 1)
plt.plot(t, s, '-', lw=2)

plt.xlabel('time (s)')
```

```
plt.ylabel('voltage (mV)')
plt.title('A sine wave with a gap of NaNs between 0.4 and 0.6')
plt.grid(True)

plt.subplot(2, 1, 2)
t[0] = np.nan
t[-1] = np.nan
plt.plot(t, s, '-', lw=2)
plt.title('Also with NaN in first and last point')

plt.xlabel('time (s)')
plt.ylabel('more nans')
plt.grid(True)

plt.tight_layout()
plt.show()
```

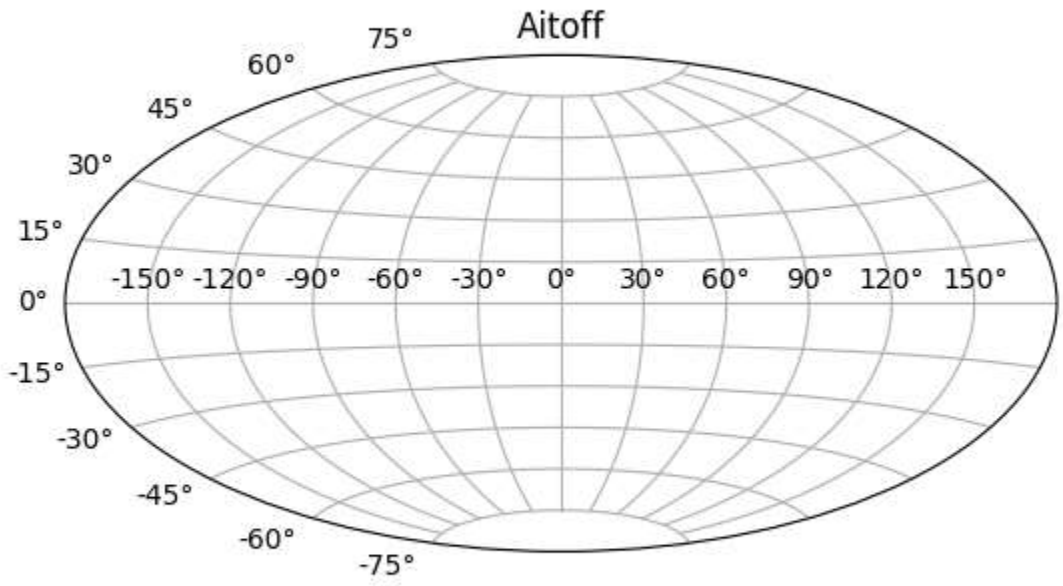
Kết quả :



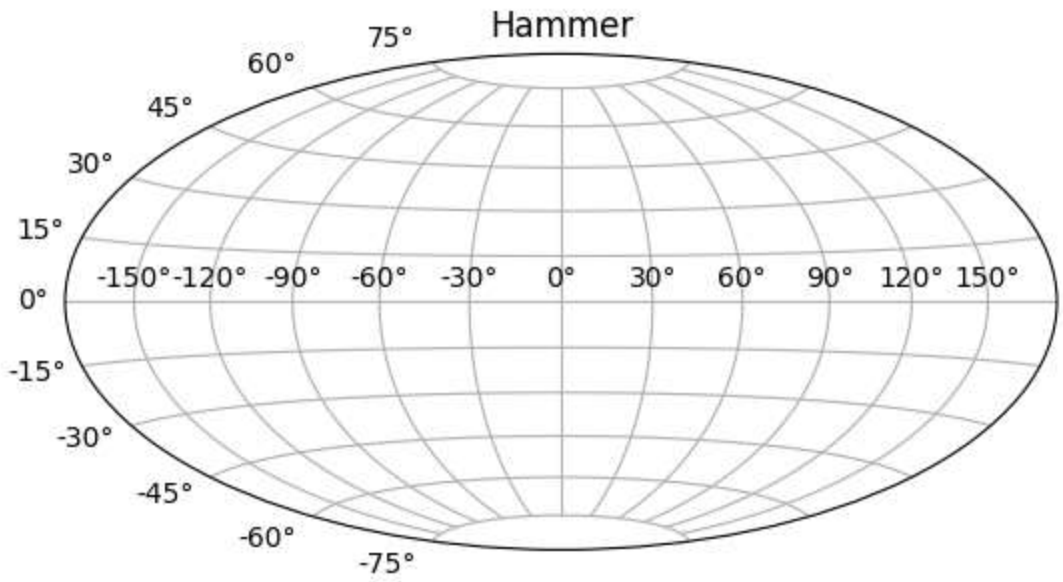
Ví dụ 2 : Geographic Projections

Điều này cho thấy 4 phép chiếu có thể sử dụng subplot. Matplotlib cũng hỗ trợ Basemaps Toolkit và Cartopy cho các phép chiếu địa lý.

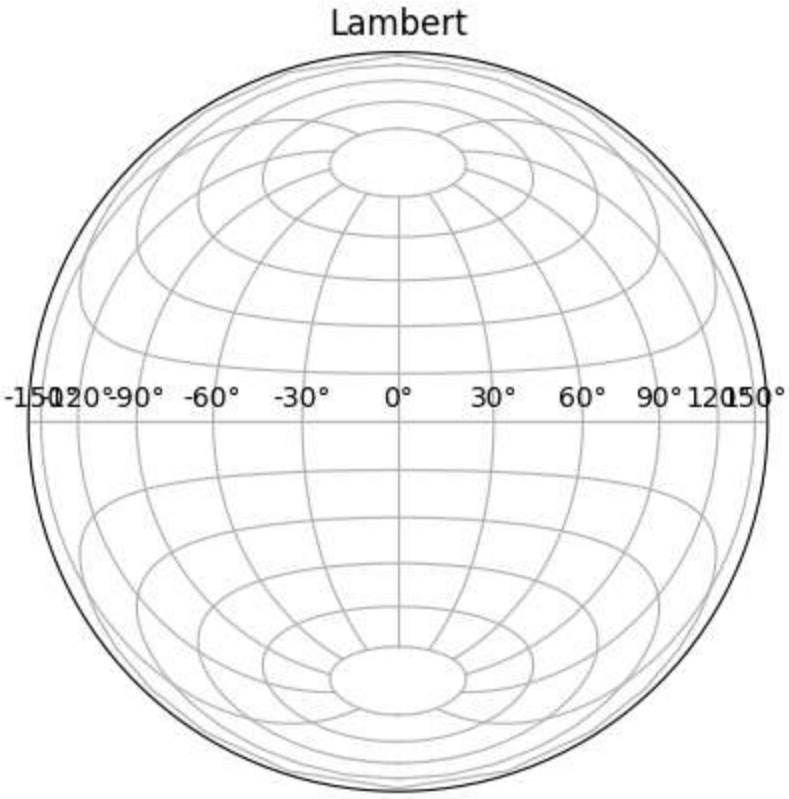
```
import matplotlib.pyplot as plt
plt.figure()
plt.subplot(111, projection="aitoff")
plt.title("Aitoff")
plt.grid(True)
```



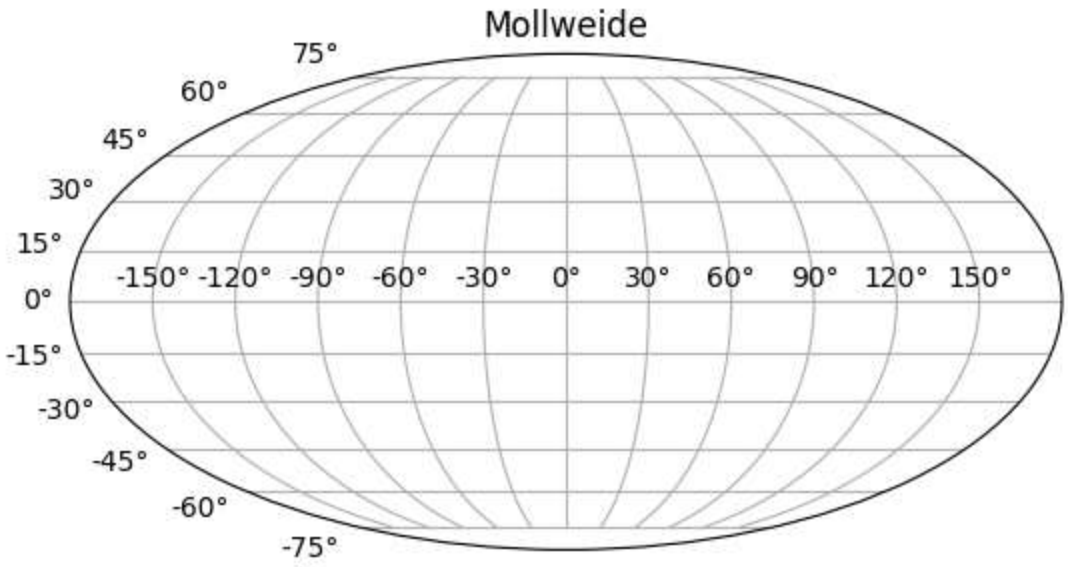

```
plt.figure()
plt.subplot(111, projection="hammer")
plt.title("Hammer")
plt.grid(True)
```




```
plt.figure()
plt.subplot(111, projection="lambert")
plt.title("Lambert")
plt.grid(True)
```




```
plt.figure()
plt.subplot(111, projection="mollweide")
plt.title("Mollweide")
plt.grid(True)

plt.show()
```



Python source code

Jupyter Notebook

Bài 12: Định dạng Axes - Matplotib Cơ Bản

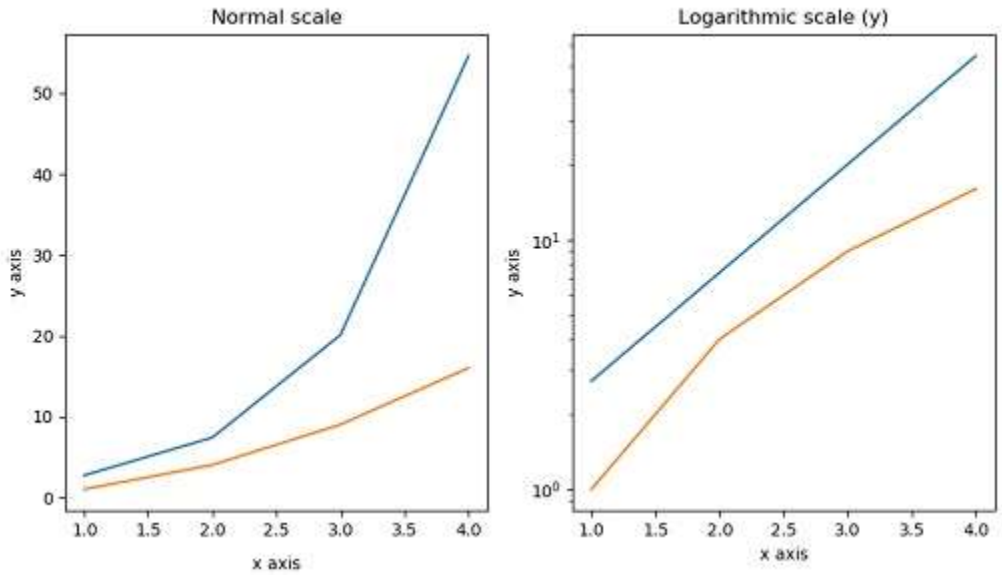
Đôi khi, các điểm lớn hơn nhiều so với phần lớn dữ liệu. Trong trường hợp này, tỷ lệ của một trục cần phải được đặt là logarit thay vì tỷ lệ bình thường. Đây là thang đo Logarit. Trong Matplotlib, có thể thực hiện bằng cách đặt thuộc tính xscale hoặc vscale của trục thành 'log'.

Yêu cầu hiển thị một số khoảng cách bổ sung giữa các số trục và label trục. Thuộc tính bảng label của một trong hai trục (x hoặc y hoặc cả hai) có thể được đặt thành giá trị mong muốn.

Xét ví dụ sau, Ô bên phải có thang đo logarit và ô bên trái có trục x có label ở khoảng cách xa hơn.

```
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))
```

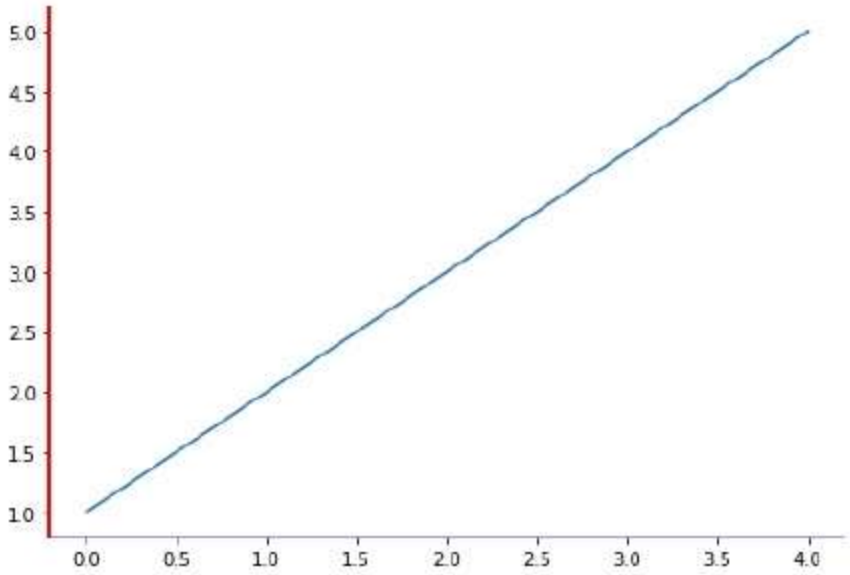
```
axes[0].plot(x,x**2)
axes[0].set_title("Normal scale")
axes[1].plot (x, np.exp(x))
axes[1].plot(x, x**2)
axes[1].set_yscale("log")
axes[1].set_title("Logarithmic scale (y)")
axes[0].set_xlabel("x axis")
axes[0].set_ylabel("y axis")
axes[0].xaxis.labelpad = 10
axes[1].set_xlabel("x axis")
axes[1].set_ylabel("y axis")
plt.show()
```



Axis spines là các đường nối trục đánh dấu phân chia ranh giới khu vực plot area. Đối tượng trục có các spine nằm ở trên, dưới, trái và phải.

Mỗi spine có thể được định dạng bằng cách chỉ định màu sắc và chiều rộng. Mọi cạnh có thể bị ẩn nếu màu của nó được đặt thành không.

```
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()
```



BÀI TIẾP THEO: ĐẶT GIỚI HẠN X VÀ Y >>

Bài 13: Đặt giới hạn X và Y - Matplotib Cơ Bản
Đăng bởi: Admin | Lượt xem: 1788 | Chuyên mục: AI

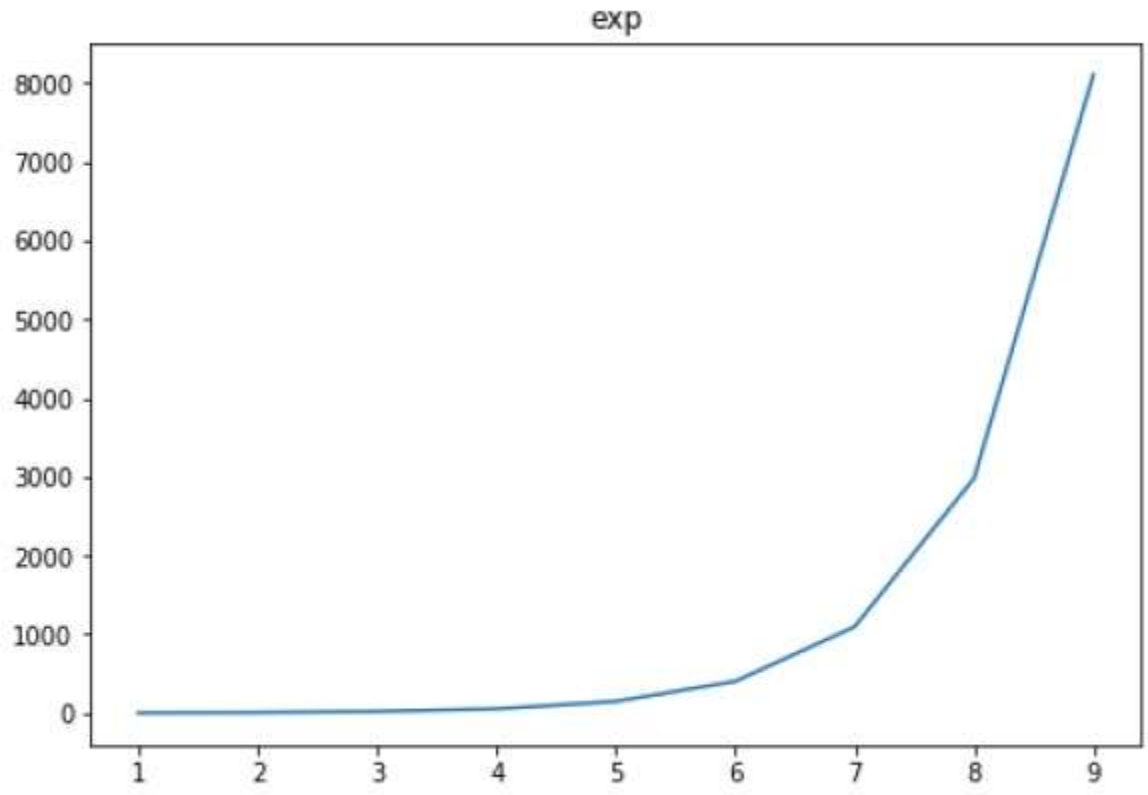
---

1. Khái niệm :

Matplotlib tự động đi từ các giá trị tối thiểu và lớn nhất của các biến được hiển thị dọc theo trục x, y (và trục z trong trường hợp biểu đồ 3D) của một biểu đồ. Tuy nhiên, có thể đặt giới hạn một cách rõ ràng bằng cách sử dụng hàm set_xlim () và set_ylim ().

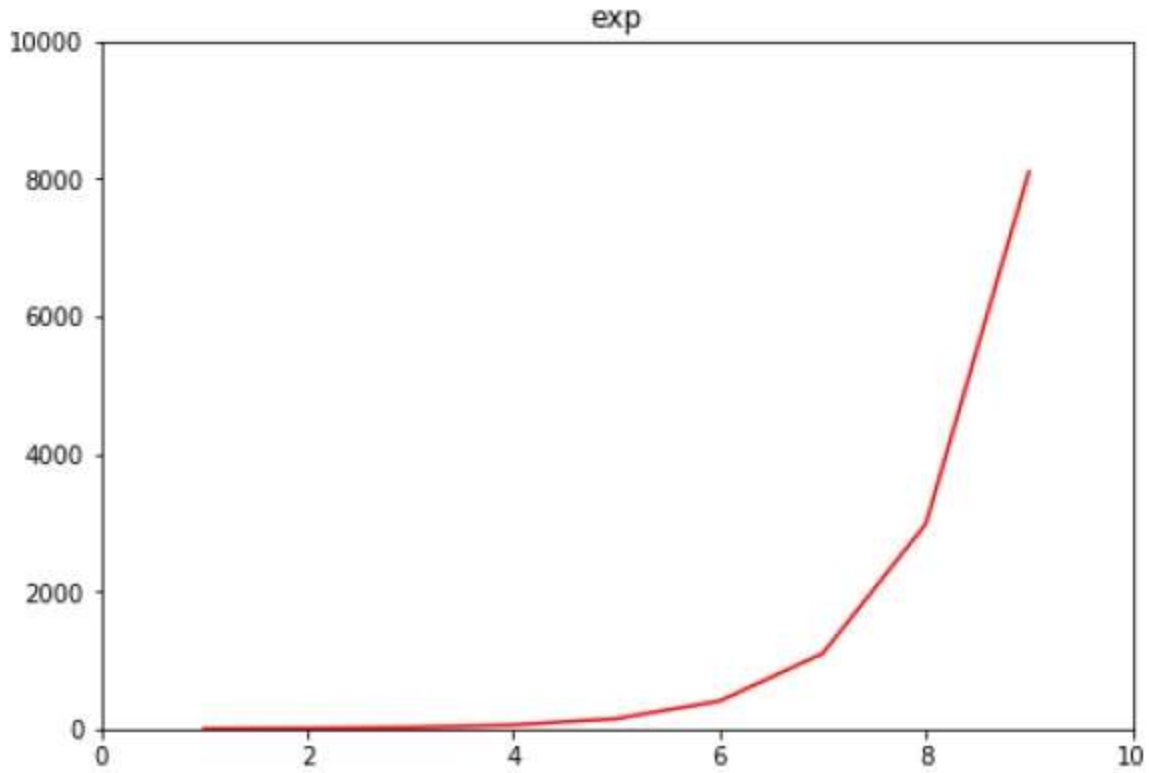
Trong biểu đồ sau, các giới hạn được tự động chia tỷ lệ của trục x và y được hiển thị:

```
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()
```



Bây giờ chỉnh giới hạn trên trục x thành (0 đến 10) và trục y (0 đến 10000) -

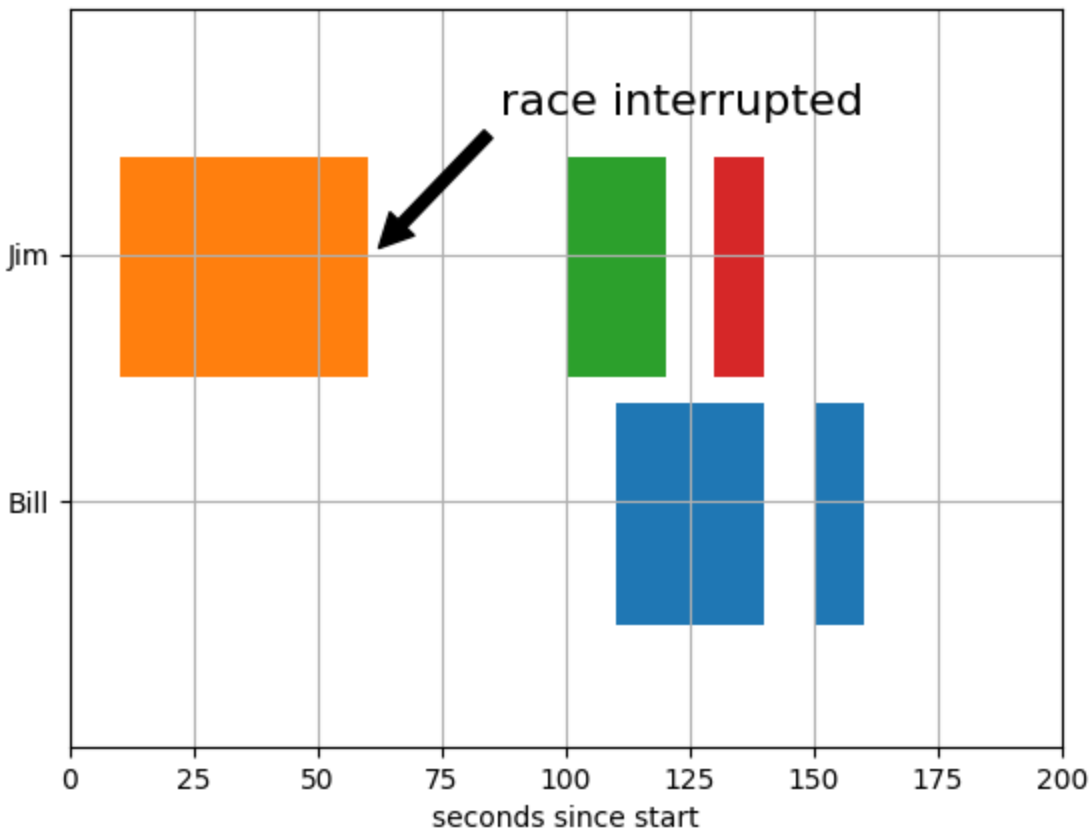
```
import matplotlib.pyplot as plt
fig = plt.figure()
a1 = fig.add_axes([0,0,1,1])
import numpy as np
x = np.arange(1,10)
a1.plot(x, np.exp(x),'r')
a1.set_title('exp')
a1.set_ylim(0,10000)
a1.set_xlim(0,10)
plt.show()
```



2. Ví dụ :

Ví dụ 1 : Broken Barh

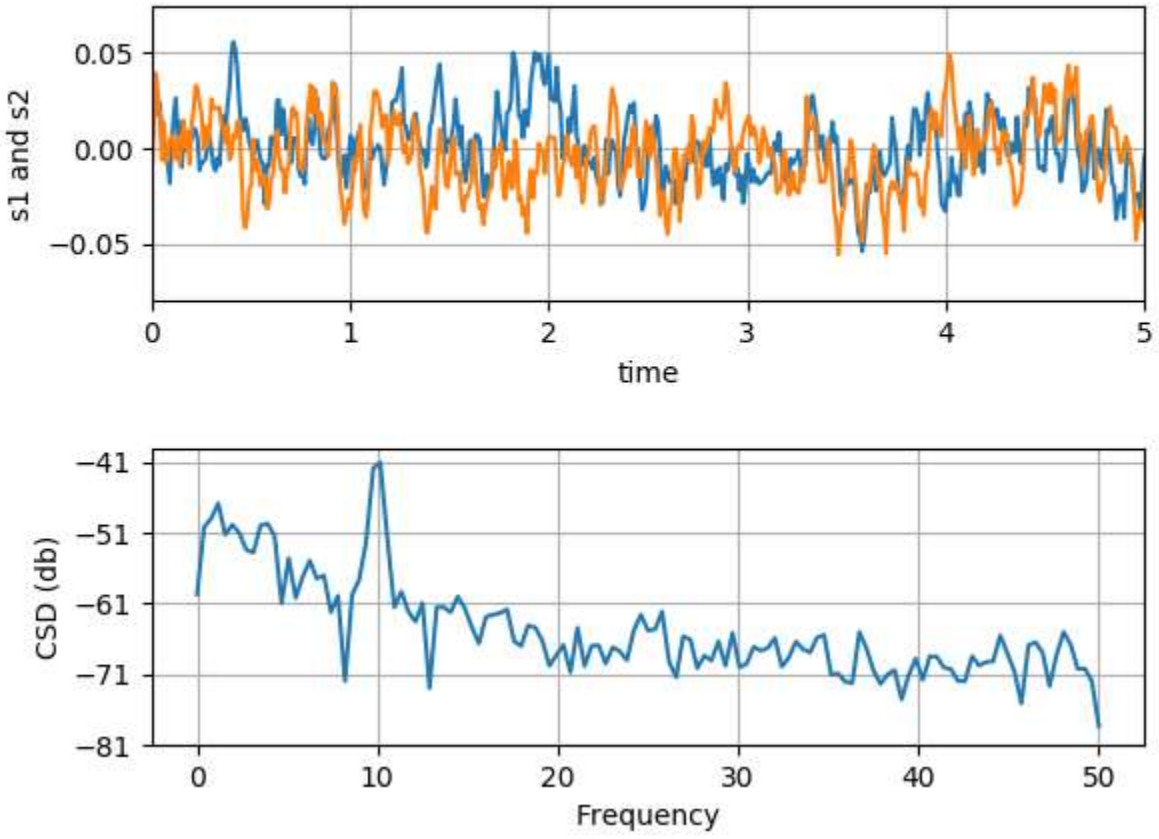
```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
ax.broken_barh([(110, 30), (150, 10)], (10, 9), facecolors='tab:blue')
ax.broken_barh([(10, 50), (100, 20), (130, 10)], (20, 9),
               facecolors=('tab:orange', 'tab:green', 'tab:red'))
ax.set_ylim(5, 35)
ax.set_xlim(0, 200)
ax.set_xlabel('seconds since start')
ax.set_yticks([15, 25])
ax.set_yticklabels(['Bill', 'Jim'])
ax.grid(True)
ax.annotate('race interrupted', (61, 25),
            xytext=(0.8, 0.9), textcoords='axes fraction',
            arrowprops=dict(facecolor='black', shrink=0.05),
            fontsize=16,
            horizontalalignment='right', verticalalignment='top')

plt.show()
```



Ví dụ 2 : CSD Demo

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


fig, (ax1, ax2) = plt.subplots(2, 1)
# make a little extra space between the subplots
fig.subplots_adjust(hspace=0.5)

dt = 0.01
t = np.arange(0, 30, dt)

# Fixing random state for reproducibility
np.random.seed(19680801)


nse1 = np.random.randn(len(t))                 # white noise 1
nse2 = np.random.randn(len(t))                 # white noise 2
r = np.exp(-t / 0.05)
```

```
cnse1 = np.convolve(nse1, r, mode='same') * dt   # colored noise 1
cnse2 = np.convolve(nse2, r, mode='same') * dt   # colored noise 2

# two signals with a coherent part and a random part
s1 = 0.01 * np.sin(2 * np.pi * 10 * t) + cnse1
s2 = 0.01 * np.sin(2 * np.pi * 10 * t) + cnse2

ax1.plot(t, s1, t, s2)
ax1.set_xlim(0, 5)
ax1.set_xlabel('time')
ax1.set_ylabel('s1 and s2')
ax1.grid(True)

cxy, f = ax2.csd(s1, s2, 256, 1. / dt)
ax2.set_ylabel('CSD (db)')
plt.show()
```

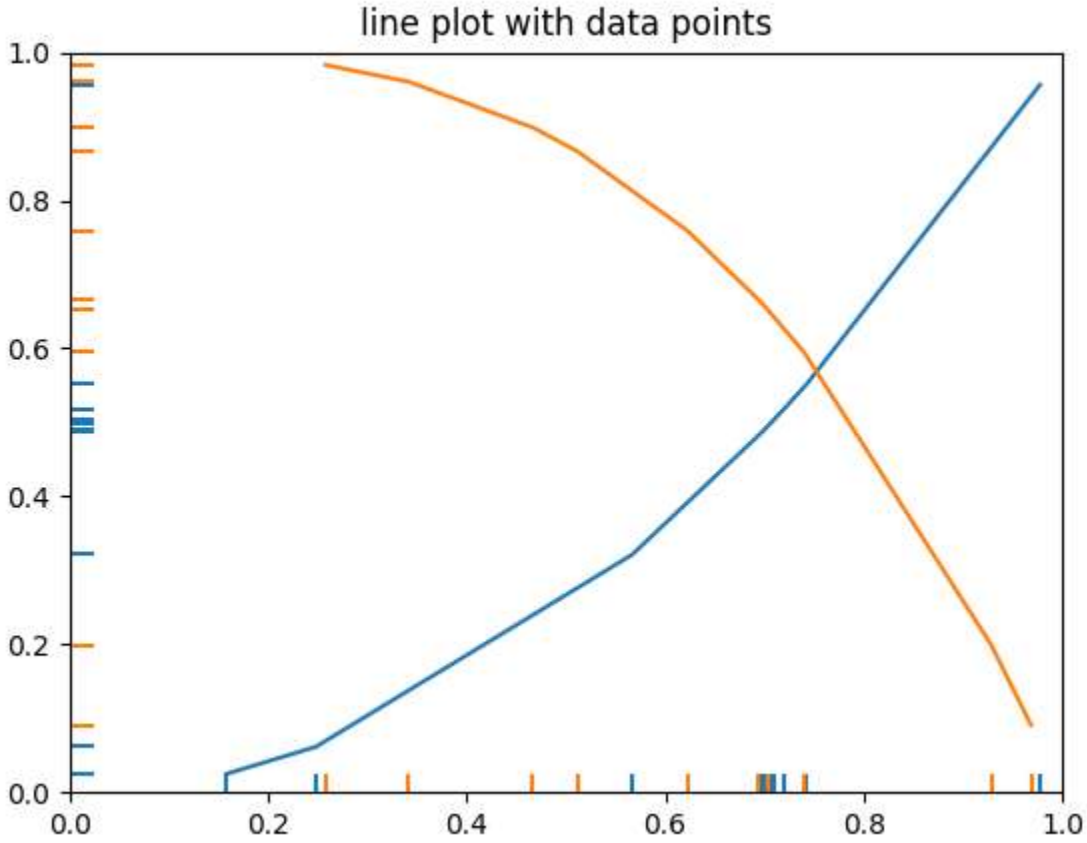


Ví dụ 3 : EventCollection Demo

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

```
from matplotlib.collections import EventCollection
import numpy as np

# Fixing random state for reproducibility
np.random.seed(19680801)

# create random data
xdata = np.random.random([2, 10])

# split the data into two parts
xdata1 = xdata[0, :]
xdata2 = xdata[1, :]

# sort the data so it makes clean curves
xdata1.sort()
xdata2.sort()

# create some y data points
ydata1 = xdata1 ** 2
ydata2 = 1 - xdata2 ** 3

# plot the data
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(xdata1, ydata1, color='tab:blue')
ax.plot(xdata2, ydata2, color='tab:orange')

# create the events marking the x data points
xevents1 = EventCollection(xdata1, color='tab:blue', linelength=0.05)
xevents2 = EventCollection(xdata2, color='tab:orange', linelength=0.05)

# create the events marking the y data points
yevents1 = EventCollection(ydata1, color='tab:blue', linelength=0.05,
                           orientation='vertical')
yevents2 = EventCollection(ydata2, color='tab:orange', linelength=0.05,
                           orientation='vertical')

# add the events to the axis
ax.add_collection(xevents1)
ax.add_collection(xevents2)
ax.add_collection(yevents1)
ax.add_collection(yevents2)

# set the limits
ax.set_xlim([0, 1])
ax.set_ylim([0, 1])
```

```
ax.set_title('line plot with data points')

# display the plot
plt.show()
```



Ví dụ 4 : Join Styles và Cap Styles

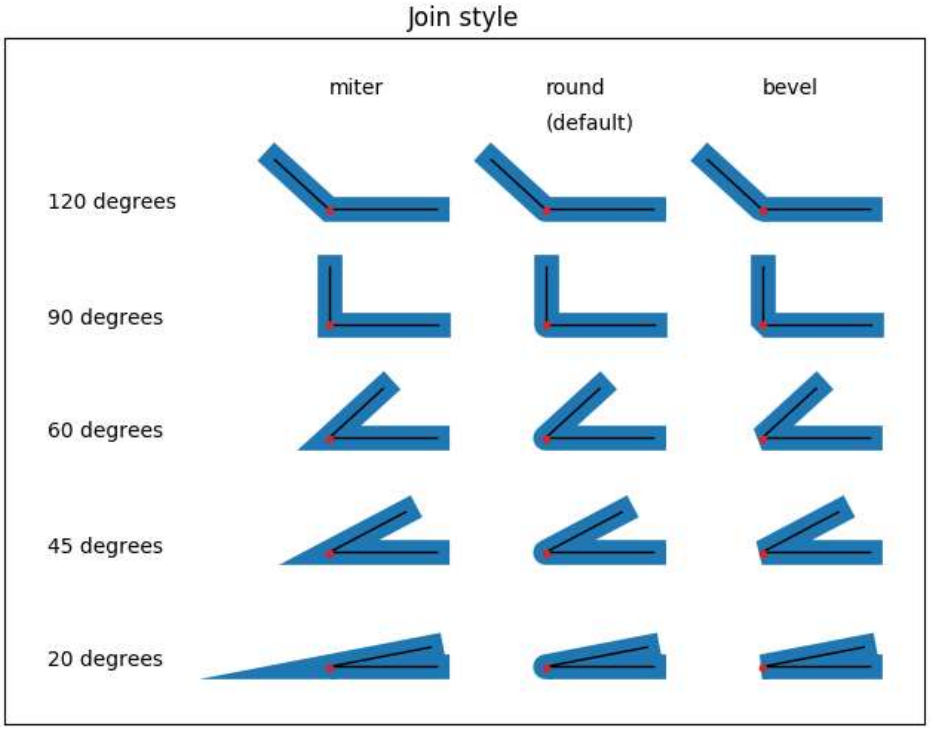
Ví dụ này minh họa các kiểu nối và kiểu mũ có sẵn.

Cả hai đều được sử dụng trong Line2D và các Bộ sưu tập khác nhau từ matplotlib.collections cũng như một số hàm tạo ra các bộ sưu tập này, ví dụ: âm mưu.

a. Join Styles :

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


def plot_angle(ax, x, y, angle, style):
    phi = np.radians(angle)
    xx = [x + .5, x, x + .5*np.cos(phi)]
```

```
    yy = [y, y, y + .5*np.sin(phi)]
    ax.plot(xx, yy, lw=12, color='tab:blue', solid_joinstyle=style)
    ax.plot(xx, yy, lw=1, color='black')
    ax.plot(xx[1], yy[1], 'o', color='tab:red', markersize=3)


fig, ax = plt.subplots(figsize=(8, 6))
ax.set_title('Join style')

for x, style in enumerate(['miter', 'round', 'bevel']):
    ax.text(x, 5, style)
    for y, angle in enumerate([20, 45, 60, 90, 120]):
        plot_angle(ax, x, y, angle, style)
        if x == 0:
            ax.text(-1.3, y, f'{angle} degrees')
ax.text(1, 4.7, '(default)')

ax.set_xlim(-1.5, 2.75)
ax.set_ylim(-.5, 5.5)
ax.xaxis.set_visible(False)
ax.yaxis.set_visible(False)
plt.show()
```



b. Cap styles :

```
fig, ax = plt.subplots(figsize=(8, 2))
ax.set_title('Cap style')

for x, style in enumerate(['butt', 'round', 'projecting']):
    ax.text(x, 1, style)
    xx = [x, x+0.5]
    yy = [0, 0]
    ax.plot(xx, yy, lw=12, color='tab:blue', solid_capstyle=style)
    ax.plot(xx, yy, lw=1, color='black')
    ax.plot(xx, yy, 'o', color='tab:red', markersize=3)
ax.text(2, 0.7, '(default)')

ax.set_ylim(-.5, 1.5)
ax.xaxis.set_visible(False)
ax.yaxis.set_visible(False)
```

BÀI TIẾP THEO: TRỤC ĐÔI >>

Bài 14: Trục đôi - Matplotib Cơ Bản

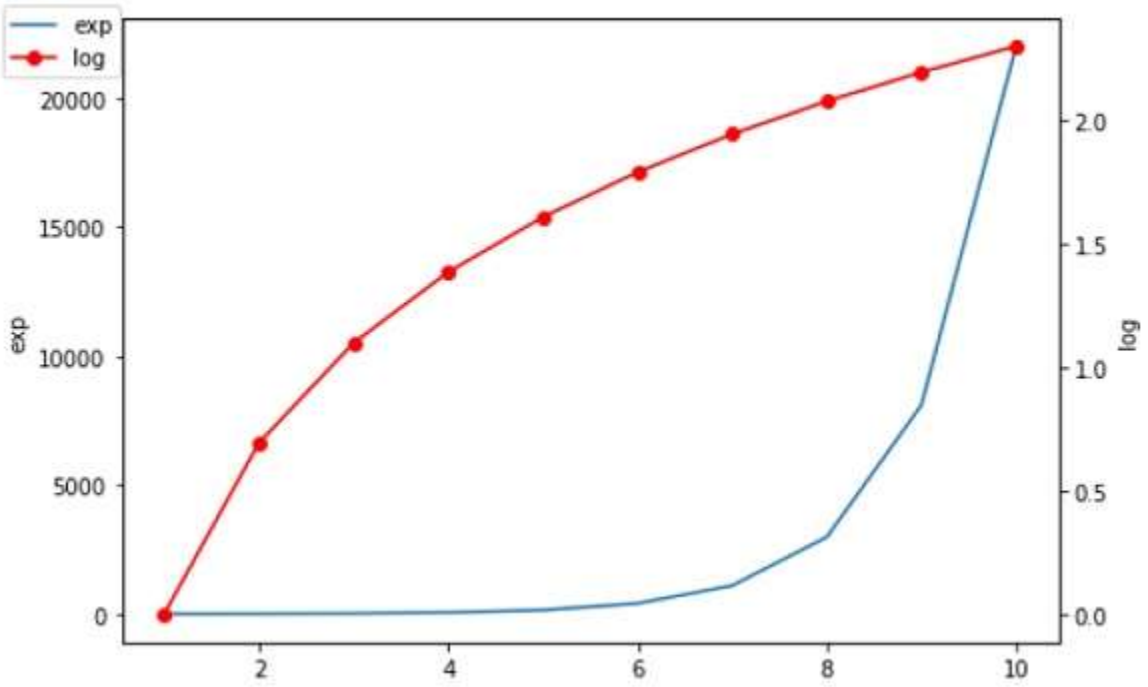
1. Khái niệm :

Trục đôi được coi là hữu ích khi có trục x hoặc y kép trong một hình. Ngoài ra, khi vẽ các đường cong với các đơn vị khác nhau v. Matplotlib hỗ trợ bằng hàm twinx và twiny.

Trong ví dụ sau, biểu đồ có trục y kép, một trục hiển thị exp (x) và trục kia hiển thị 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()
```



2. Ví dụ :

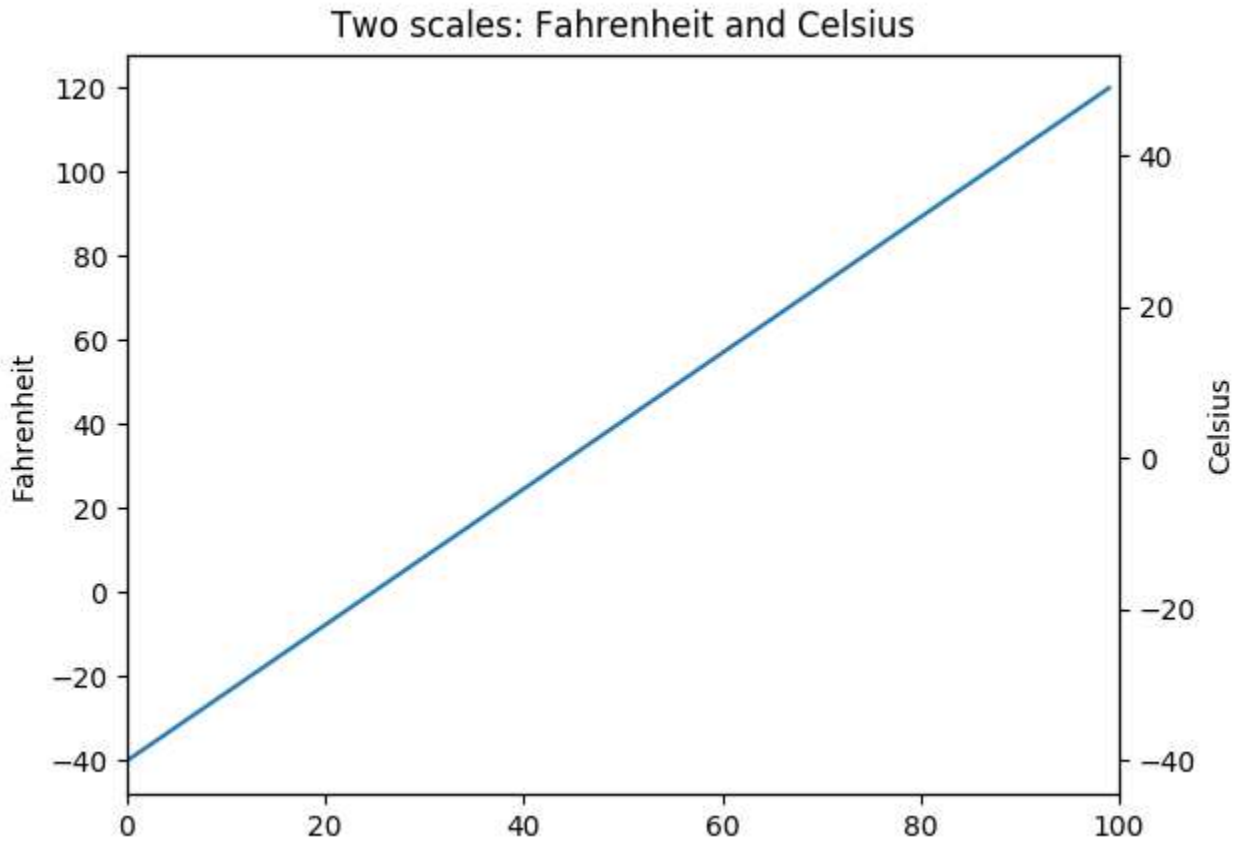
Ví dụ 1 : Các tỷ lệ khác nhau trên cùng một trục

Demo cách hiển thị hai thang đo trên trục y bên trái và bên phải.

Ví dụ này sử dụng thang đo độ F và độ C.

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


def fahrenheit2celsius(temp):
    """
    Returns temperature in Celsius.
    """
    return (5. / 9.) * (temp - 32)


def convert_ax_c_to_celsius(ax_f):
    """
    Update second axis according with first axis.
    """
```

```
    y1, y2 = ax_f.get_ylim()
    ax_c.set_ylim(fahrenheit2celsius(y1), fahrenheit2celsius(y2))
    ax_c.figure.canvas.draw()

fig, ax_f = plt.subplots()
ax_c = ax_f.twinx()

# automatically update ylim of ax2 when ylim of ax1 changes.
ax_f.callbacks.connect("ylim_changed", convert_ax_c_to_celsius)
ax_f.plot(np.linspace(-40, 120, 100))
ax_f.set_xlim(0, 100)

ax_f.set_title('Two scales: Fahrenheit and Celsius')
ax_f.set_ylabel('Fahrenheit')
ax_c.set_ylabel('Celsius')

plt.show()
```



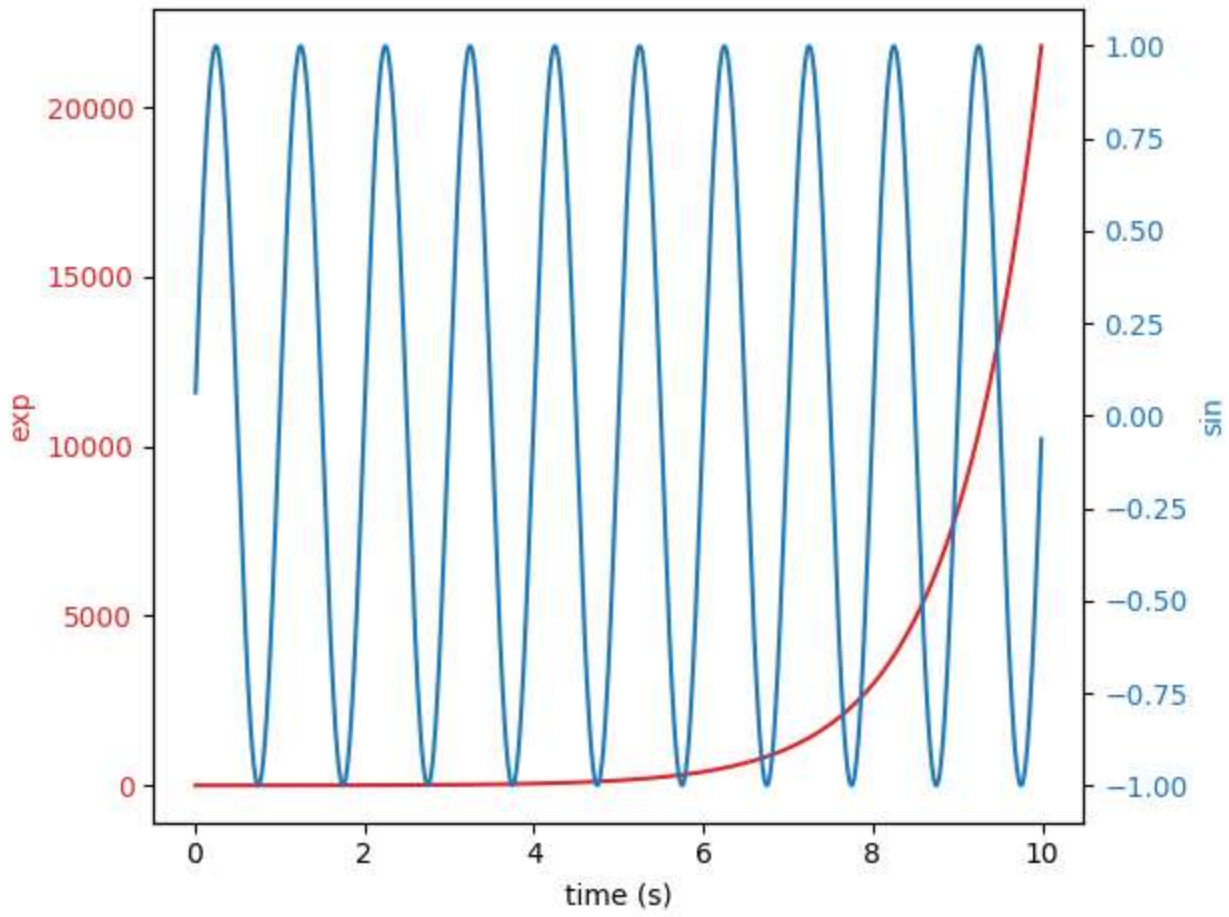
Ví dụ 2 : Các plot với các tỷ lệ khác nhau

Hai plot trên cùng một trục với tỷ lệ bên trái và bên phải khác nhau.

Bí quyết là sử dụng hai trục khác nhau có cùng trục x. Bạn có thể sử dụng các bộ định dạng và bộ định vị matplotlib.ticker riêng biệt vì hai trục là độc lập.

Các trục như vậy được tạo ra bằng cách gọi phương thức Axes.twinx (). Tương tự như vậy, Axes.twiny () có sẵn để tạo ra các trục có chung trục y nhưng có tỷ lệ trên và dưới khác nhau.

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

# Create some mock data
t = np.arange(0.01, 10.0, 0.01)
data1 = np.exp(t)
data2 = np.sin(2 * np.pi * t)

fig, ax1 = plt.subplots()

color = 'tab:red'
ax1.set_xlabel('time (s)')
ax1.set_ylabel('exp', color=color)
ax1.plot(t, data1, color=color)
ax1.tick_params(axis='y', labelcolor=color)

ax2 = ax1.twinx()  # instantiate a second axes that shares the same x-axis

color = 'tab:blue'
ax2.set_ylabel('sin', color=color)  # we already handled the x-label with ax1
ax2.plot(t, data2, color=color)
ax2.tick_params(axis='y', labelcolor=color)

fig.tight_layout()  # otherwise the right y-label is slightly clipped
plt.show()
```



Ví dụ 3 : Phần trăm dưới dạng biểu đồ thanh ngang

Biểu đồ thanh rất hữu ích để hiển thị số lượng hoặc thống kê tóm tắt với các thanh lỗi. Ngoài ra, xem biểu đồ thanh được nhóm có label hoặc ví dụ về biểu đồ thanh ngang để biết các phiên bản đơn giản hơn của các tính năng đó.

Ví dụ này xuất phát từ một ứng dụng trong đó giáo viên thể dục của lớp muốn có thể cho phụ huynh thấy con họ đã làm như thế nào qua một số bài kiểm tra thể lực và quan trọng là liên quan đến cách những đứa trẻ khác đã làm. Để trích xuất cho mục đích demo, ta sẽ chỉ tạo một số dữ liệu nhỏ.

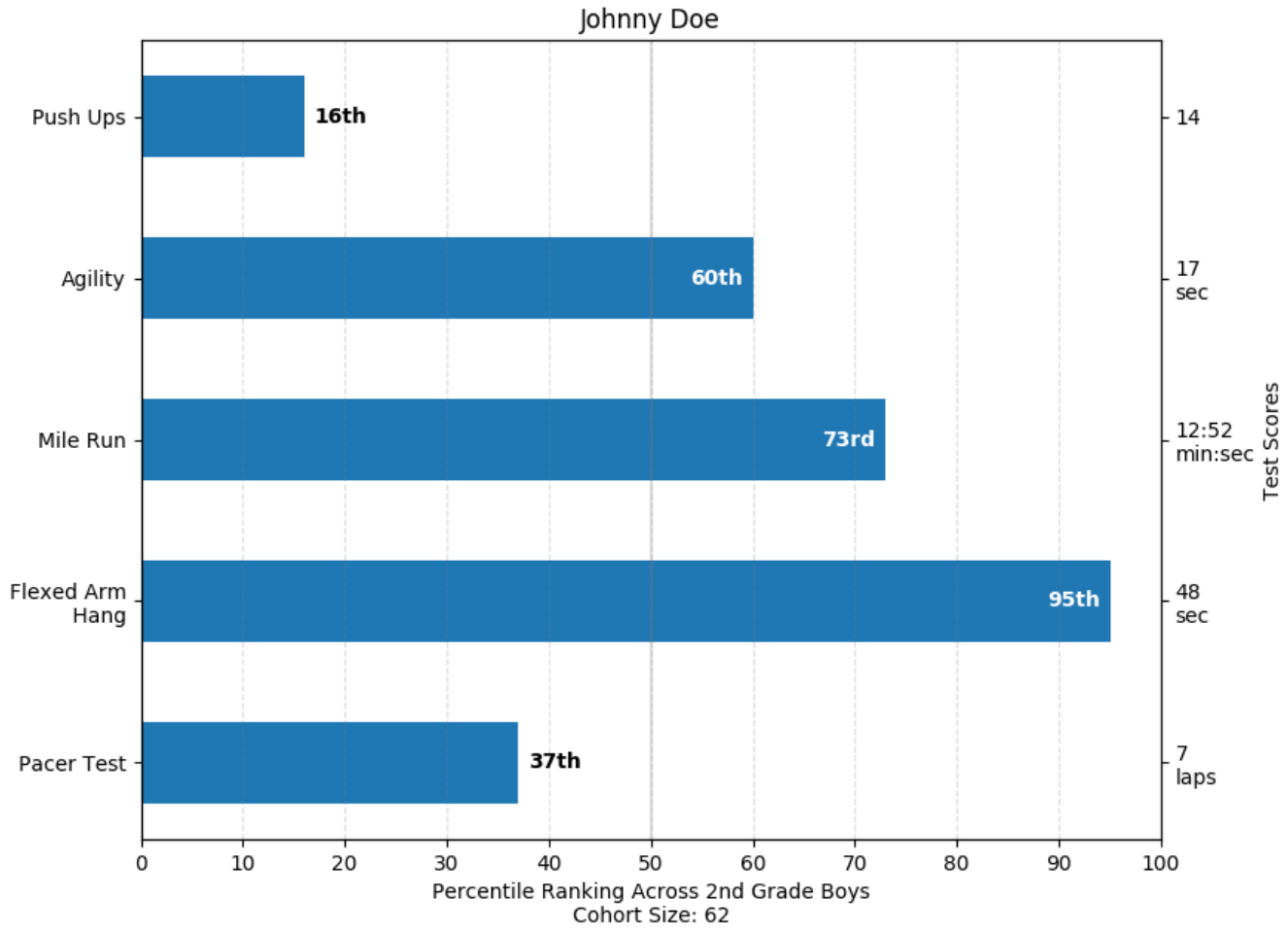
```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator
from collections import namedtuple

np.random.seed(42)

Student = namedtuple('Student', ['name', 'grade', 'gender'])
```

```
Score = namedtuple('Score', ['score', 'percentile'])

# GLOBAL CONSTANTS
testNames = ['Pacer Test', 'Flexed Arm\n Hang', 'Mile Run', 'Agility',
             'Push Ups']
testMeta = dict(zip(testNames, ['laps', 'sec', 'min:sec', 'sec', '']))


def attach_ordinal(num):
    """helper function to add ordinal string to integers

    1 -> 1st
    56 -> 56th
    """
    suffixes = {str(i): v
                for i, v in enumerate(['th', 'st', 'nd', 'rd', 'th',
                                       'th', 'th', 'th', 'th', 'th'])}

    v = str(num)
    # special case early teens
    if v in {'11', '12', '13'}:
        return v + 'th'
    return v + suffixes[v[-1]]


def format_score(scr, test):
    """
    Build up the score labels for the right Y-axis by first
    appending a carriage return to each string and then tacking on
    the appropriate meta information (i.e., 'laps' vs 'seconds'). We
    want the labels centered on the ticks, so if there is no meta
    info (like for pushups) then don't add the carriage return to
    the string
    """
    md = testMeta[test]
    if md:
        return '{0}\n{1}'.format(scr, md)
    else:
        return scr


def format_ycursor(y):
    y = int(y)
    if y < 0 or y >= len(testNames):
        return ''
    else:
```

```
        return testNames[y]


def plot_student_results(student, scores, cohort_size):
    #  create the figure
    fig, ax1 = plt.subplots(figsize=(9, 7))
    fig.subplots_adjust(left=0.115, right=0.88)
    fig.canvas.set_window_title('Eldorado K-8 Fitness Chart')

    pos = np.arange(len(testNames))

    rects = ax1.barh(pos, [scores[k].percentile for k in testNames],
                     align='center',
                     height=0.5,
                     tick_label=testNames)

    ax1.set_title(student.name)

    ax1.set_xlim([0, 100])
    ax1.xaxis.set_major_locator(MaxNLocator(11))
    ax1.xaxis.grid(True, linestyle='--', which='major',
                   color='grey', alpha=.25)

    # Plot a solid vertical gridline to highlight the median position
    ax1.axvline(50, color='grey', alpha=0.25)

    # Set the right-hand Y-axis ticks and labels
    ax2 = ax1.twinx()

    scoreLabels = [format_score(scores[k].score, k) for k in testNames]

    # set the tick locations
    ax2.set_yticks(pos)
    # make sure that the limits are set equally on both yaxis so the
    # ticks line up
    ax2.set_ylim(ax1.get_ylim())

    # set the tick labels
    ax2.set_yticklabels(scoreLabels)

    ax2.set_ylabel('Test Scores')

    xlabel = ('Percentile Ranking Across {grade} Grade {gender}s\n'
              'Cohort Size: {cohort_size}')
    ax1.set_xlabel(xlabel.format(grade=attach_ordinal(student.grade),
                                 gender=student.gender.title(),
```

```
                        cohort_size=cohort_size))

    rect_labels = []
    # Lastly, write in the ranking inside each bar to aid in interpretation
    for rect in rects:
        # Rectangle widths are already integer-valued but are floating
        # type, so it helps to remove the trailing decimal point and 0 by
        # converting width to int type
        width = int(rect.get_width())

        rankStr = attach_ordinal(width)
        # The bars aren't wide enough to print the ranking inside
        if width < 40:
            # Shift the text to the right side of the right edge
            xloc = 5
            # Black against white background
            clr = 'black'
            align = 'left'
        else:
            # Shift the text to the left side of the right edge
            xloc = -5
            # White on magenta
            clr = 'white'
            align = 'right'

        # Center the text vertically in the bar
        yloc = rect.get_y() + rect.get_height() / 2
        label = ax1.annotate(rankStr, xy=(width, yloc), xytext=(xloc, 0),
                             textcoords="offset points",
                             ha=align, va='center',
                             color=clr, weight='bold', clip_on=True)
        rect_labels.append(label)

    # make the interactive mouse over give the bar title
    ax2.fmt_ydata = format_ycursor
    # return all of the artists created
    return {'fig': fig,
            'ax': ax1,
            'ax_right': ax2,
            'bars': rects,
            'perc_labels': rect_labels}


student = Student('Johnny Doe', 2, 'boy')
scores = dict(zip(testNames,
                  (Score(v, p) for v, p in
```

```
                zip(['7', '48', '12:52', '17', '14'],
                    np.round(np.random.uniform(0, 1,
                                               len(testNames)) * 100, 0)))))
cohort_size = 62  # The number of other 2nd grade boys

arts = plot_student_results(student, scores, cohort_size)
plt.show()
```



BÀI TIẾP THEO: BAR PLOT >>

Bài 15: Bar Plot - Matplotib Cơ Bản
Đăng bởi: Admin | Lượt xem: 4759 | Chuyên mục: AI

Biểu đồ hoặc đồ thị trình bày dữ liệu phân loại với các thanh hình chữ nhật có chiều cao hoặc chiều dài tỷ lệ với các giá trị đại diện. Các thanh có thể được vẽ theo chiều dọc hoặc chiều ngang.

Biểu đồ hiển thị so sánh giữa các categories rời rạc. Một trục của biểu đồ hiển thị các categories cụ thể được so sánh và trục còn lại thể hiện giá trị đo được.

Matplotlib API cung cấp hàm bar () có thể được sử dụng trong việc sử dụng kiểu MATLAB cũng như API. Signaturecủa hàm bar () được sử dụng với axis như sau:

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

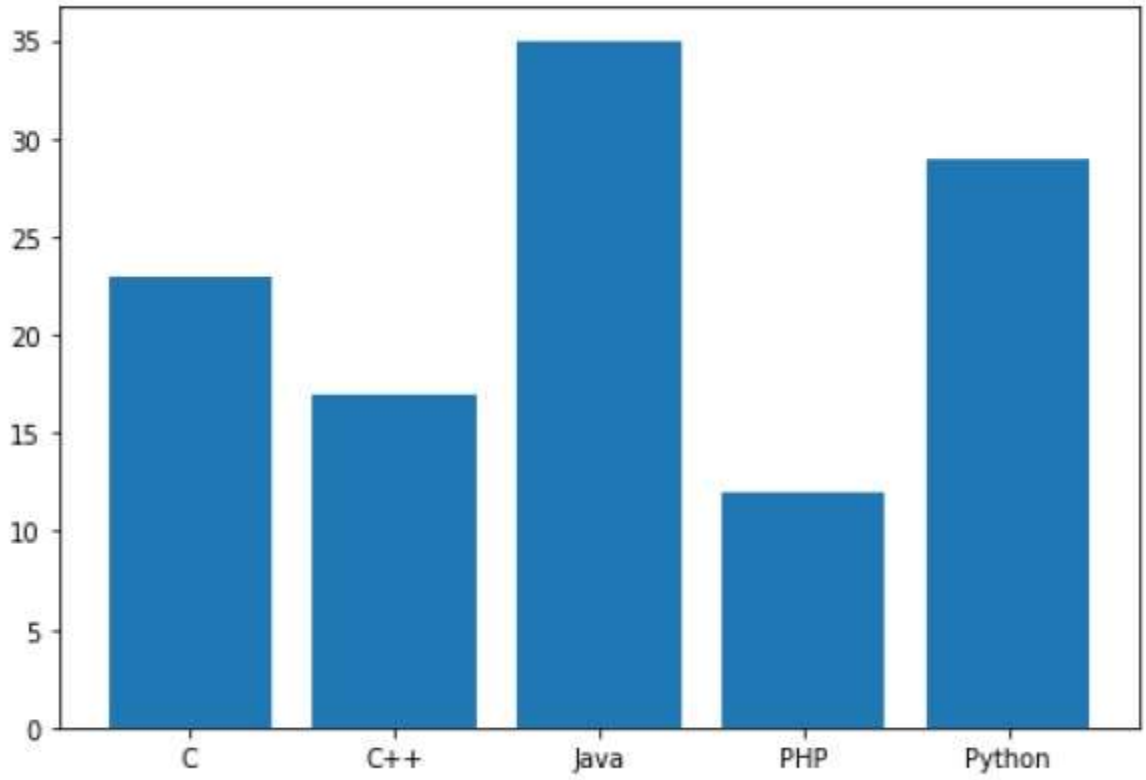
Hàm tạo một biểu đồ dạng thanh với hình chữ nhật giới hạn có kích thước (x − width = 2; x + width = 2; bottom; bottom + height).

Các tham số của hàm là -

x	**dãy vô hướng biểu diễn tọa độ x của các thanh. căn chỉnh các điều khiển nếu x là tâm thanh (mặc định) hoặc cạnh trái.**
height	vô hướng hoặc chuỗi các đại lượng vô hướng đại diện cho (các) chiều cao của các thanh.
width	vô hướng hoặc dạng mảng, tùy chọn. (các) chiều rộng của các thanh mặc định là 0,8
bottom	vô hướng hoặc dạng mảng, tùy chọn. (các) tọa độ y của các thanh mặc định Không có.
align	{‘Center’, ‘edge’}, tùy chọn, ‘center’ mặc định

Hàm trả về một object chứa Matplotlib với tất cả các thanh bar.
Sau đây là một ví dụ đơn giản về biểu đồ Matplotlib.Cho thấy số lượng sinh viên đăng ký cho các khóa học khác nhau được cung cấp tại một viện.

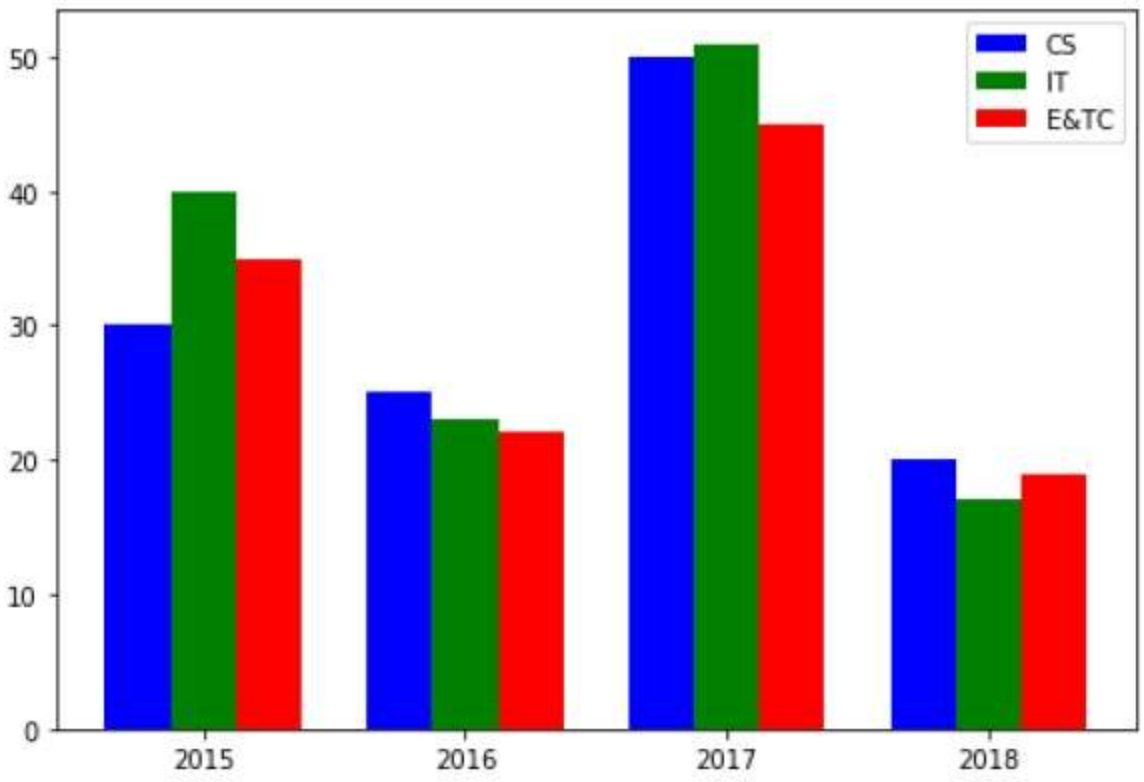
```
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
langs = ['C', 'C++', 'Java', 'Python', 'PHP']
students = [23,17,35,29,12]
ax.bar(langs,students)
plt.show()
```



Khi so sánh một số đại lượng và khi thay đổi một biến, ta có thể muốn có một biểu đồ thanh trong đó ta có các thanh có một màu cho một giá trị đại lượng.

Ta có thể vẽ nhiều biểu đồ thanh bằng cách sử dụng độ dày và vị trí của các thanh. Biến dữ liệu chứa ba chuỗi bốn giá trị. Tập lệnh sau sẽ hiển thị ba biểu đồ thanh gồm bốn thanh. Các thanh sẽ có độ dày 0,25 đơn vị. Mỗi biểu đồ sẽ được dịch chuyển 0,25 đơn vị so với biểu đồ trước đó. dữ liệu là một đa sắc số chứa số lượng sinh viên đã đậu vào ba ngành của một trường cao đẳng kỹ thuật trong bốn năm qua.

```
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)
```

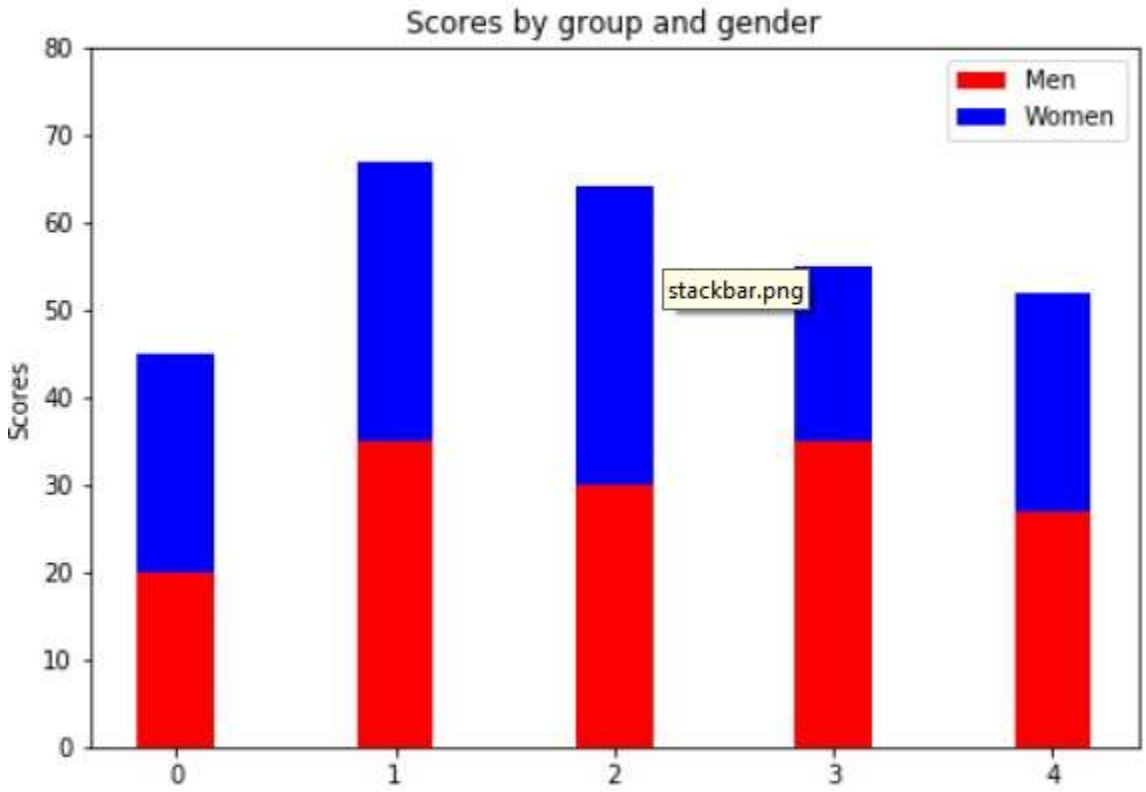


Biểu đồ xếp chồng các thanh đại diện cho các nhóm khác nhau và chồng lên nhau. Chiều cao của thanh kết quả hiển thị kết quả tổng hợp của các nhóm.

Tham số dưới cùng tùy chọn của hàm pyplot.bar () cho phép bạn chỉ định giá trị bắt đầu cho một thanh. Thay vì chạy từ 0 đến một giá trị, nó sẽ đi từ dưới cùng đến giá trị. Lệnh gọi đầu tiên tới pyplot.bar () vẽ các thanh màu xanh lam. Lệnh gọi thứ hai tới pyplot.bar () vẽ các thanh màu đỏ, với đáy của các thanh màu xanh nằm trên cùng của các thanh màu đỏ.

```
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()
```



BÀI TIẾP THEO: HISTOGRAM >>

1) Biểu đồ Histogram

Histogram là sự phân bố dữ liệu số. Nó ước lượng phân phối xác suất của một biến liên tục và là một dạng biểu đồ thanh.

Để tạo biểu đồ, hãy làm theo các bước sau:

- **Bin** phạm vi giá trị.
- Chia toàn bộ phạm vi giá trị thành một loạt các khoảng.
- Đếm xem có bao nhiêu giá trị rơi vào mỗi khoảng.

Các bins thường được chỉ định là các khoảng liên tiếp, không chồng chéo của một biến.

Hàm matplotlib.pyplot.hist () vẽ một biểu đồ, tính toán và vẽ biểu đồ của x.

Tham số :

Bảng sau liệt kê các tham số cho histogram:

x	**mảng hoặc chuỗi các mảng**

bins	số nguyên hoặc chuỗi hoặc 'auto', tùy chọn

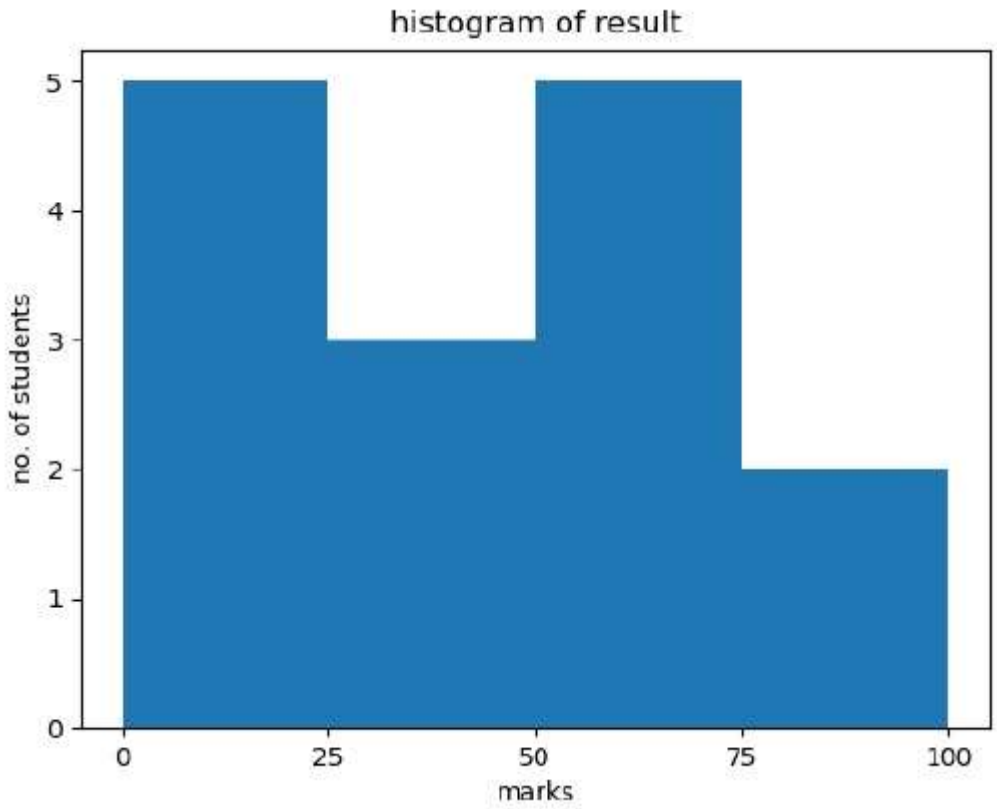
Thông số tùy chọn

range	**Phạm vi dưới và trên của bin**
density	Nếu true, phần tử đầu tiên của bộ giá trị trả về sẽ là số đếm được chuẩn hóa để tạo thành mật độ xác suất
cumulative	Nếu true, thì biểu đồ được tính toán trong đó mỗi thùng cung cấp số lượng trong thùng đó cộng với tất cả các thùng cho các giá trị nhỏ hơn
histtype	Loại histogram để vẽ. Mặc định là 'bar'

Ví dụ sau vẽ biểu đồ về điểm của các sinh viên trong một lớp học. Bốn thùng, 0-25, 26-50, 51-75 và 76-100 được xác định. Biểu đồ cho thấy số học sinh rơi vào phạm vi này.

```
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()
```

Cốt truyện xuất hiện như hình dưới đây -



2) Biểu đồ Pie

Biểu đồ tròn có thể hiển thị một chuỗi dữ liệu. Biểu đồ tròn hiển thị kích thước của các mục (được gọi là wedge) trong một chuỗi dữ liệu, tỷ lệ với tổng của các mục. Các điểm dữ liệu trong biểu đồ tròn được hiển thị dưới dạng phần trăm của toàn bộ hình tròn.

Matplotlib API có một hàm pie () tạo ra một biểu đồ tròn đại diện cho dữ liệu trong một mảng. Diện tích phần của mỗi hình wedge được cho bởi x / sum (x). Nếu sum (x) <1, thì các giá trị của x cung cấp trực tiếp diện tích phân số và mảng sẽ không được chuẩn hóa. Bánh kết quả sẽ có một hình wedge trống có kích thước 1 - sum (x).

Biểu đồ hình tròn trông đẹp nhất nếu hình và trục là hình vuông hoặc khía cạnh Axes bằng nhau.

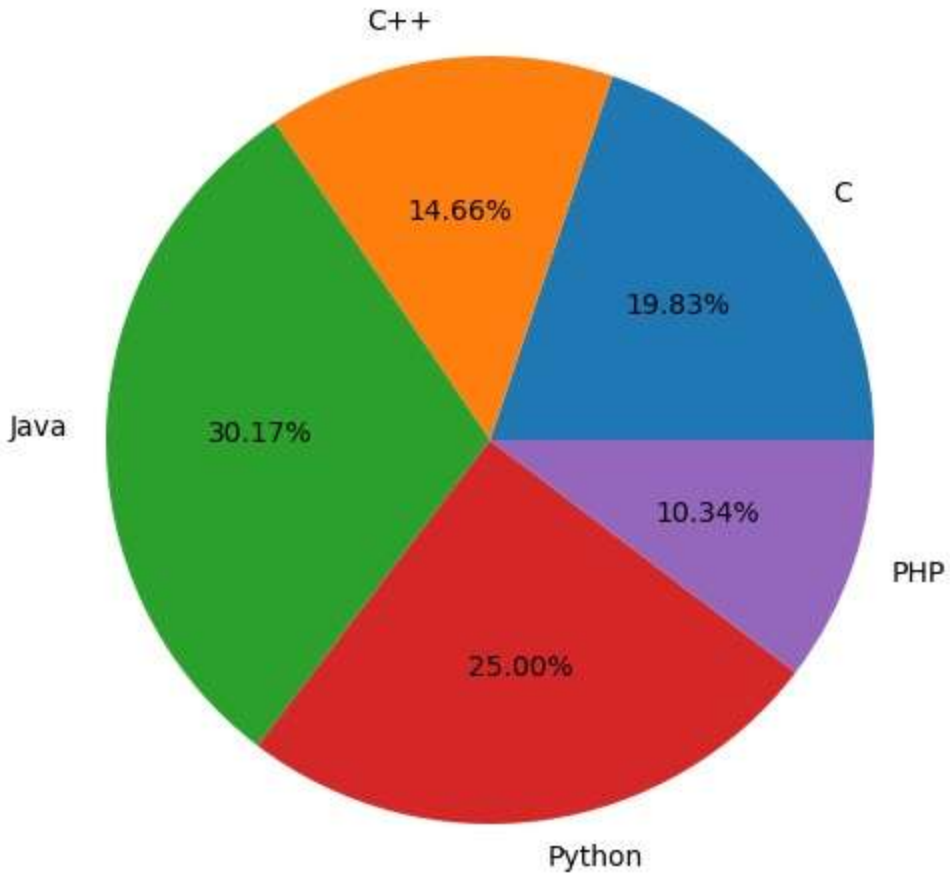
Các tham số cần biết :

x	**giống mảng. Các kích thước wedge.**
labels	danh sách. Một chuỗi các chuỗi cung cấp các label cho mỗi wedge.

Colors	Một chuỗi các vòng màu matplotlib mà qua đó biểu đồ tròn sẽ xoay vòng. Nếu Không, sẽ sử dụng các màu trong chu kỳ hiện đang hoạt động.
Autopct	chuỗi, được sử dụng để gắn label các wedge với giá trị số . Label sẽ được đặt bên trong wedge. Chuỗi định dạng sẽ là fmt% pct.

Đoạn code sau sử dụng hàm pie () để hiển thị biểu đồ tròn của danh sách sinh viên đăng ký các khóa học ngôn ngữ máy tính khác nhau. Tỷ lệ phần trăm tương ứng được hiển thị bên trong nêm tương ứng với sự trợ giúp của tham số autopct được đặt thành% 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()
```

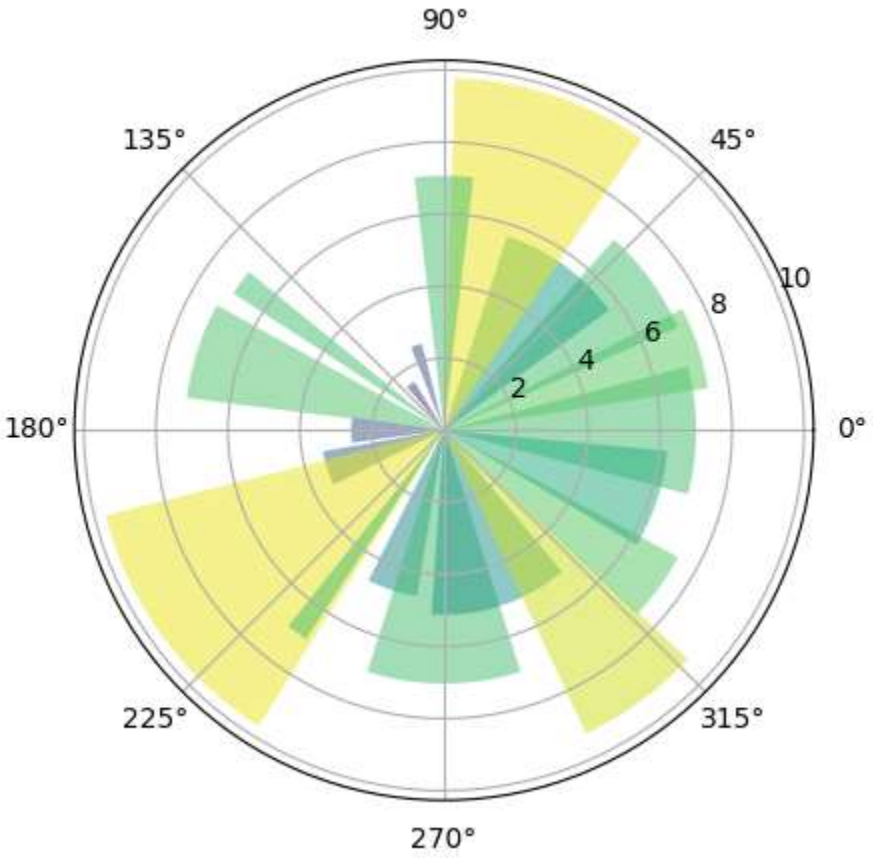


2. Ví dụ minh hoạ :

Ví dụ 1 : Demo của biểu đồ tròn trên một trục cực.

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


# Fixing random state for reproducibility
np.random.seed(19680801)

# Compute pie slices
N = 20
theta = np.linspace(0.0, 2 * np.pi, N, endpoint=False)
radii = 10 * np.random.rand(N)
width = np.pi / 4 * np.random.rand(N)
colors = plt.cm.viridis(radii / 10.)

ax = plt.subplot(111, projection='polar')
ax.bar(theta, radii, width=width, bottom=0.0, color=colors, alpha=0.5)
```

```
plt.show()
```
Kết quả :



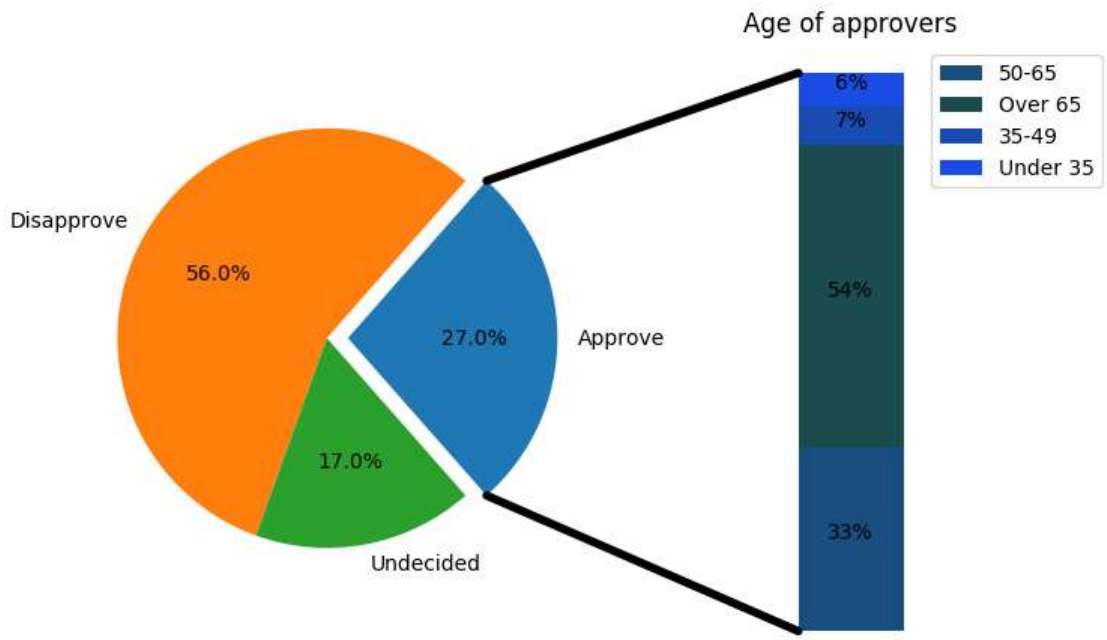
Ví dụ 2 : Bar of pie

Tạo biểu đồ "Bar of pie" trong đó lát đầu tiên của hình tròn được "exploded" thành biểu đồ thanh với sự phân tích sâu hơn về các đặc điểm của lát đó. Ví dụ minh họa bằng cách sử dụng một hình có nhiều bộ trục và sử dụng danh sách các bản vá trục để thêm hai Bản vá kết nối để liên kết các biểu đồ subplot.

```
import matplotlib.pyplot as plt
from matplotlib.patches import ConnectionPatch
import numpy as np

# make figure and assign axis objects
fig = plt.figure(figsize=(9, 5.0625))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)
fig.subplots_adjust(wspace=0)
```

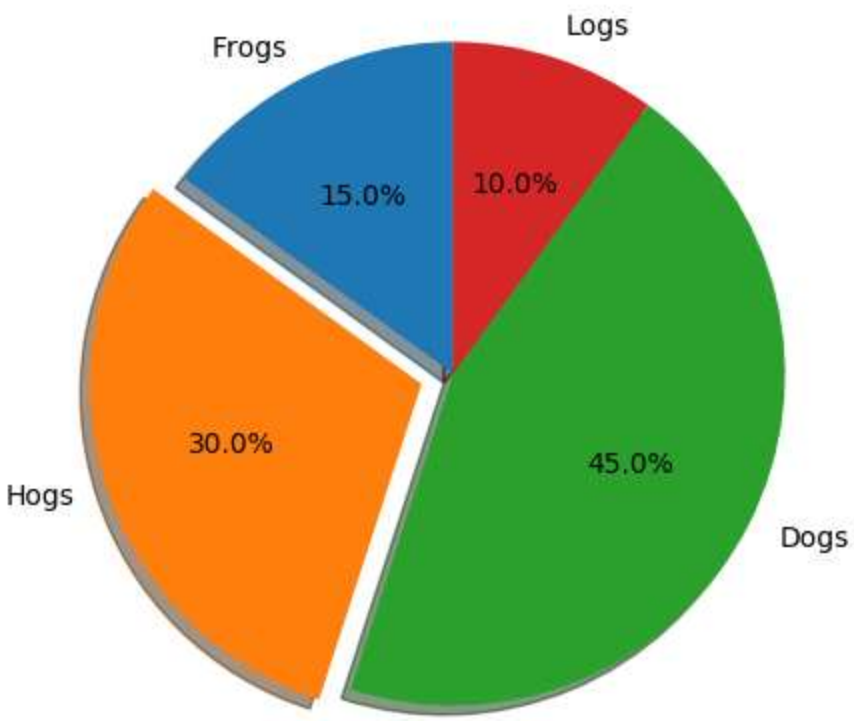
```
# pie chart parameters
ratios = [.27, .56, .17]
labels = ['Approve', 'Disapprove', 'Undecided']
explode = [0.1, 0, 0]
# rotate so that first wedge is split by the x-axis
angle = -180 * ratios[0]
ax1.pie(ratios, autopct='%1.1f%%', startangle=angle,
        labels=labels, explode=explode)

# bar chart parameters

xpos = 0
bottom = 0
ratios = [.33, .54, .07, .06]
width = .2
colors = [[.1, .3, .5], [.1, .3, .3], [.1, .3, .7], [.1, .3, .9]]

for j in range(len(ratios)):
    height = ratios[j]
    ax2.bar(xpos, height, width, bottom=bottom, color=colors[j])
    ypos = bottom + ax2.patches[j].get_height() / 2
    bottom += height
    ax2.text(xpos, ypos, "%d%%" % (ax2.patches[j].get_height() * 100),
             ha='center')

ax2.set_title('Age of approvers')
ax2.legend(('50-65', 'Over 65', '35-49', 'Under 35'))
ax2.axis('off')
ax2.set_xlim(- 2.5 * width, 2.5 * width)

# use ConnectionPatch to draw lines between the two plots
# get the wedge data
theta1, theta2 = ax1.patches[0].theta1, ax1.patches[0].theta2
center, r = ax1.patches[0].center, ax1.patches[0].r
bar_height = sum([item.get_height() for item in ax2.patches])

# draw top connecting line
x = r * np.cos(np.pi / 180 * theta2) + center[0]
y = np.sin(np.pi / 180 * theta2) + center[1]
con = ConnectionPatch(xyA=(- width / 2, bar_height), xyB=(x, y),
                      coordsA="data", coordsB="data", axesA=ax2, axesB=ax1)
con.set_color([0, 0, 0])
con.set_linewidth(4)
ax2.add_artist(con)
```

```
# draw bottom connecting line
x = r * np.cos(np.pi / 180 * theta1) + center[0]
y = np.sin(np.pi / 180 * theta1) + center[1]
con = ConnectionPatch(xyA=(- width / 2, 0), xyB=(x, y), coordsA="data",
                      coordsB="data", axesA=ax2, axesB=ax1)
con.set_color([0, 0, 0])
ax2.add_artist(con)
con.set_linewidth(4)

plt.show()
```



Ví dụ 3 :

```
import matplotlib.pyplot as plt

# Pie chart, where the slices will be ordered and plotted counter-clockwise:
labels = 'Frogs', 'Hogs', 'Dogs', 'Logs'
sizes = [15, 30, 45, 10]
explode = (0, 0.1, 0, 0)  # only "explode" the 2nd slice (i.e. 'Hogs')

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

plt.show()
```



3) **BÀI TIẾP THEO: SCATTER PLOT (BIỂU ĐỒ PHÂN TÁN) >>**

Bài 18: Scatter Plot (Biểu đồ phân tán) - Matplotib Cơ Bản
Đăng bởi: Admin | Lượt xem: 6588 | Chuyên mục: AI

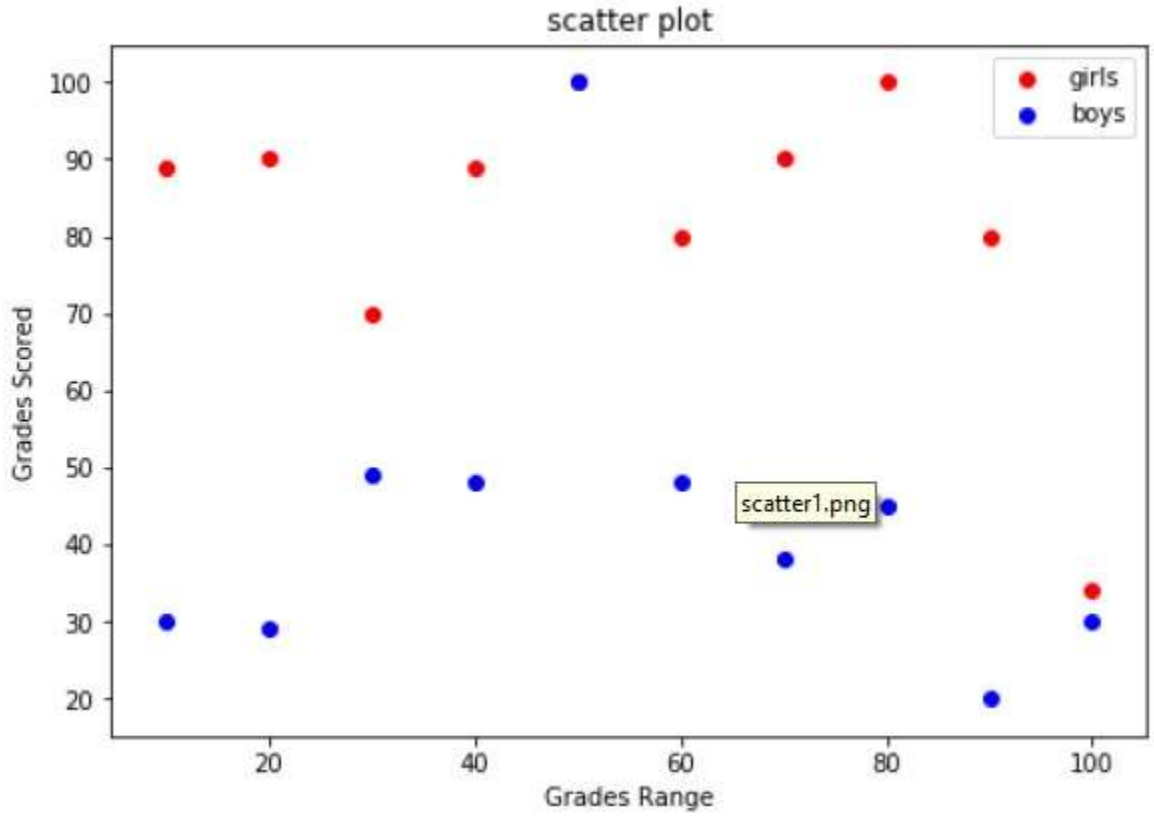
---

1. Khái niệm cơ bản :

Biểu đồ phân tán được sử dụng để vẽ các điểm dữ liệu trên trục hoành và trục tung để thể hiện mức độ ảnh hưởng của một biến này bởi biến khác. Mỗi hàng trong bảng dữ liệu được biểu thị bằng một điểm đánh dấu, vị trí phụ thuộc vào giá trị của nó trong các cột được đặt trên trục X và Y. Một biến thứ ba có thể được đặt để tương ứng với màu sắc hoặc kích thước của các điểm đánh dấu, do đó thêm một chiều khác vào biểu đồ.

Ví dụ dưới đây vẽ một biểu đồ phân tán của các cấp lớp so với cấp độ của nam và nữ bằng hai màu sắc khác nhau.

```
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()
```

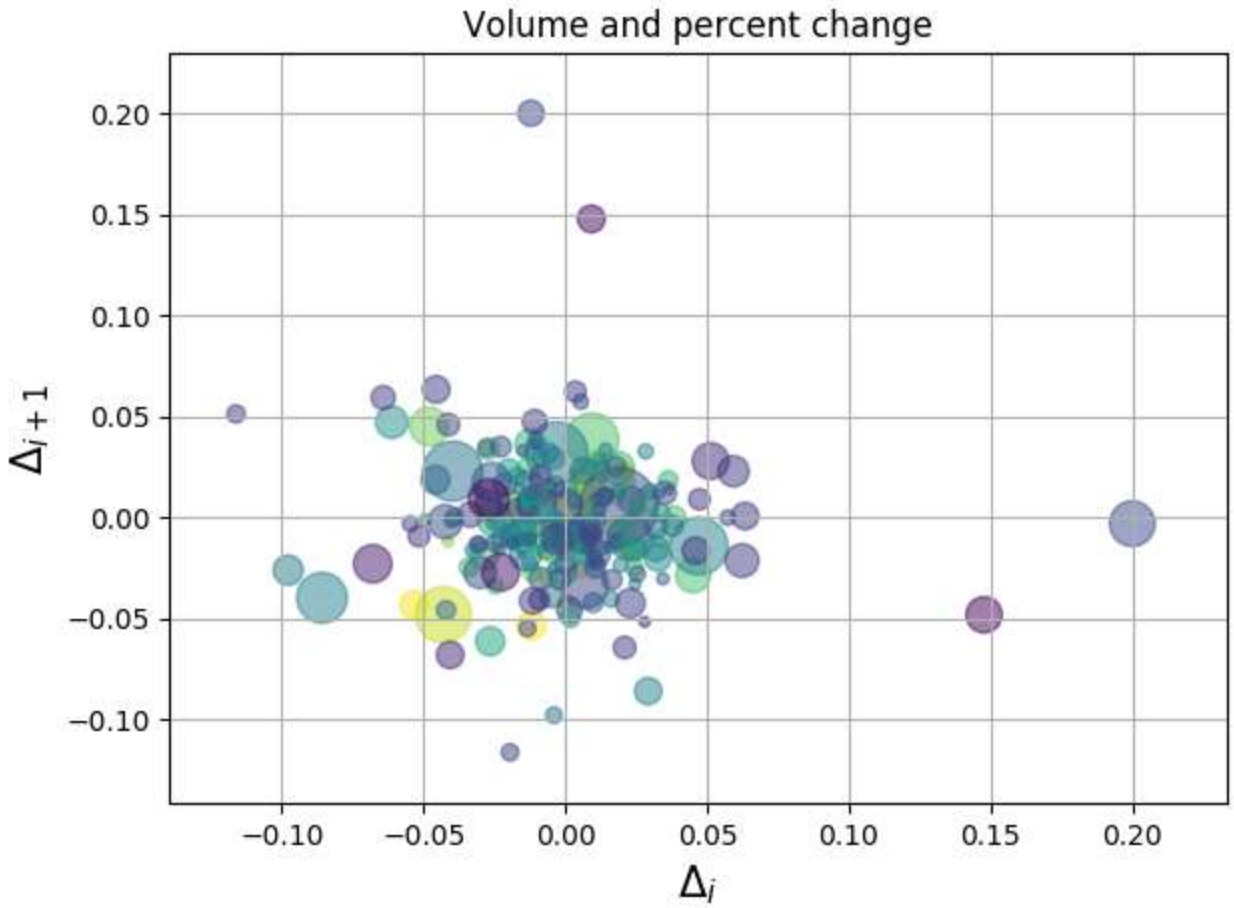


2. Ví dụ minh hoạ :

Ví dụ 1 : Biểu diễn biểu đồ phân tán với các màu và kích thước điểm đánh dấu khác nhau.

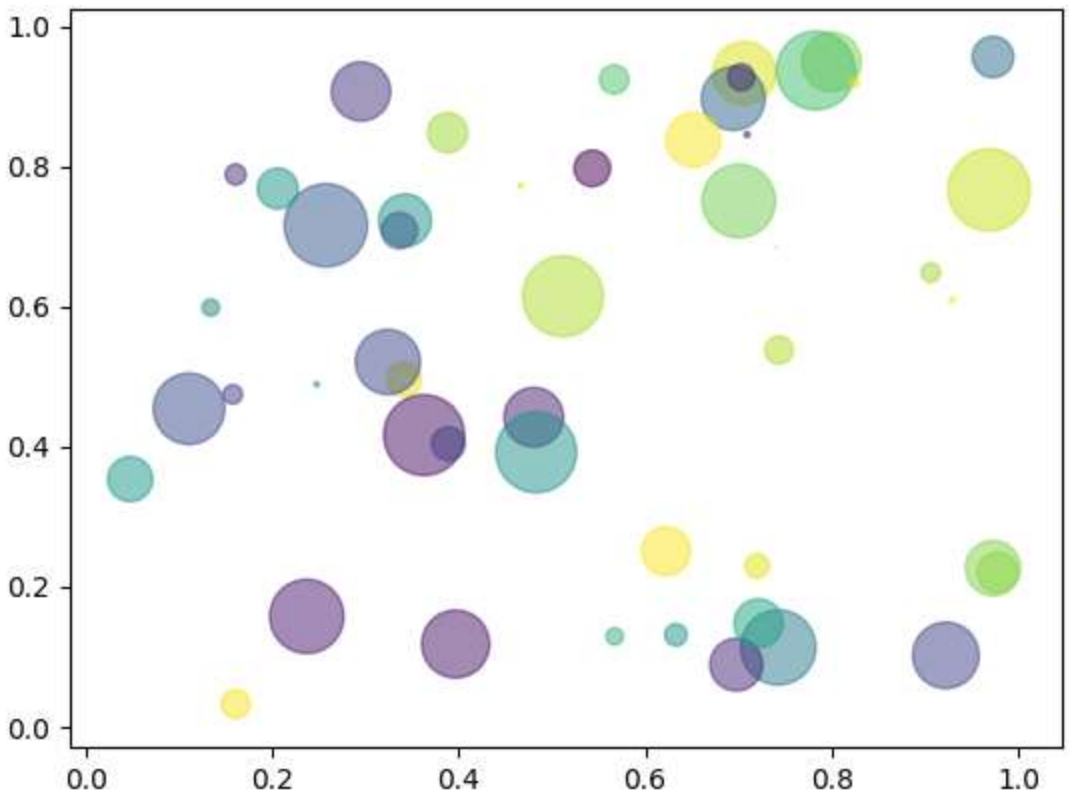
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cbook as cbook

# Load a numpy record array from yahoo csv data with fields date, open, close,
# volume, adj_close from the mpl-data/example directory. The record array
# stores the date as an np.datetime64 with a day unit ('D') in the date column.
```

```
with cbook.get_sample_data('goog.npz') as datafile:
    price_data = np.load(datafile)['price_data'].view(np.recarray)
price_data = price_data[-250:]  # get the most recent 250 trading days

delta1 = np.diff(price_data.adj_close) / price_data.adj_close[:-1]

# Marker size in units of points^2
volume = (15 * price_data.volume[:-2] / price_data.volume[0])**2
close = 0.003 * price_data.close[:-2] / 0.003 * price_data.open[:-2]

fig, ax = plt.subplots()
ax.scatter(delta1[:-1], delta1[1:], c=close, s=volume, alpha=0.5)

ax.set_xlabel(r'$\Delta_i$', fontsize=15)
ax.set_ylabel(r'$\Delta_{i+1}$', fontsize=15)
ax.set_title('Volume and percent change')

ax.grid(True)
fig.tight_layout()

plt.show()
```



Ví dụ 2 :

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

# Fixing random state for reproducibility
np.random.seed(19680801)


N = 50
x = np.random.rand(N)
y = np.random.rand(N)
colors = np.random.rand(N)
area = (30 * np.random.rand(N))**2  # 0 to 15 point radii

plt.scatter(x, y, s=area, c=colors, alpha=0.5)
plt.show()
```

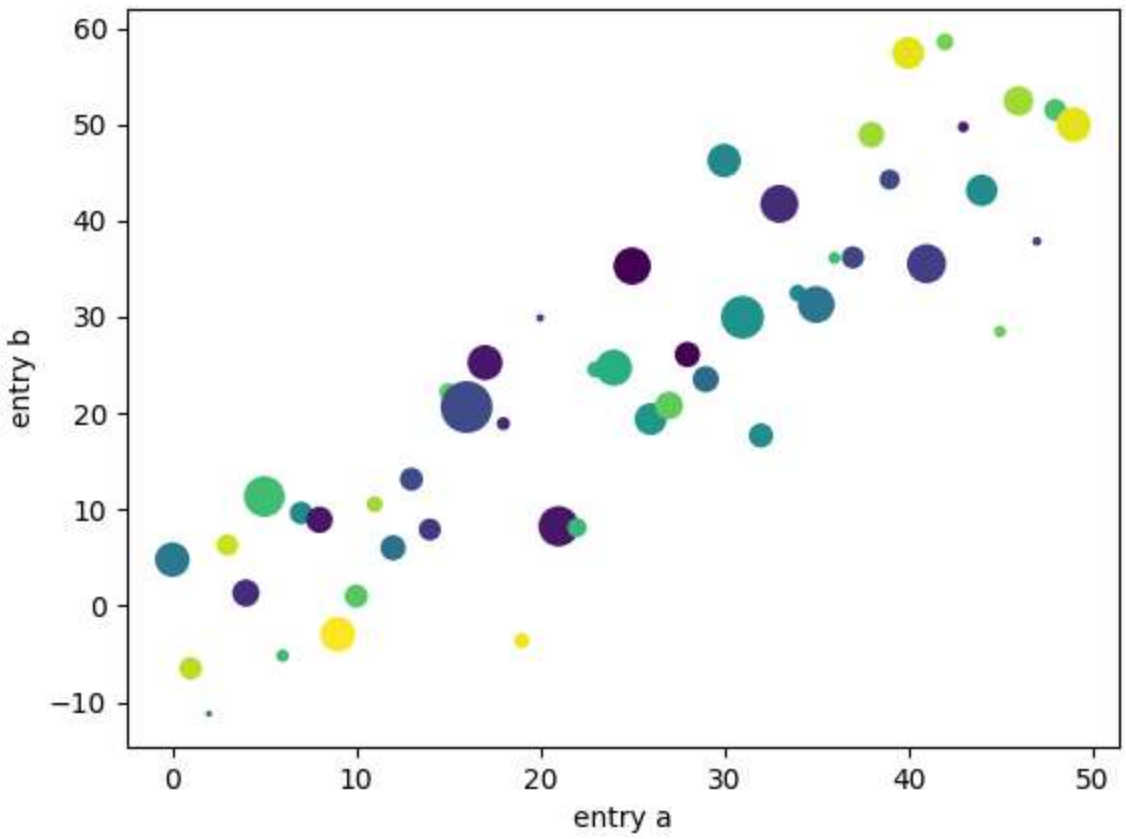


Ví dụ 3 : Sử dụng keywords để tạo plot

Có một số trường hợp bạn có dữ liệu ở dạng cho phép bạn truy cập các biến cụ thể bằng chuỗi. Ví dụ: numpy.recarray hoặc pandas.DataFrame.

Matplotlib cho phép ta cung cấp một đối tượng như vậy với đối số từ khóa dữ liệu. Nếu được cung cấp, có thể tạo các plot các chuỗi tương ứng với các biến này.

```
import matplotlib.pyplot as plt
np.random.seed(19680801)

data = {'a': np.arange(50),
        'c': np.random.randint(0, 50, 50),
        'd': np.random.randn(50)}
data['b'] = data['a'] + 10 * np.random.randn(50)
data['d'] = np.abs(data['d']) * 100

fig, ax = plt.subplots()
ax.scatter('a', 'b', c='c', s='d', data=data)
ax.set(xlabel='entry a', ylabel='entry b')
plt.show()
```



2. Ưu và nhược điểm của Scatter diagram

2.1 Ưu điểm

Scatter diagram có một số ưu điểm sau:

- Dễ dàng trong việc vẽ biểu đồ
- Thể hiện rõ mối tương quan giữa các biến số và xu hướng dữ liệu
- Biểu diễn được tất cả các dữ liệu từ nhỏ đến lớn và cả các giá trị ngoại lai
- Có thể sử dụng trong nhiều ngành nghề với các kiểu dữ liệu khác nhau

Biểu đồ này rất dễ vẽ và biểu diễn được toàn bộ loại dữ liệu

>>>> Đọc Thêm: 7 nguyên tắc quản lý chất lượng trong ISO 9001:2015

2.2 Nhược điểm

Bên cạnh những ưu điểm, Scatter diagram còn có một số nhược điểm sau:

- Phán đoán mang tính chủ quan khi chỉ dựa trên biểu đồ
- Hệ số tương quan giữa các biến số khó đưa ra kết quả chính xác
- Các biến số cần đáp ứng điều kiện là biến liên tục
- Chỉ biểu diễn được 2 biến số trên một biểu đồ

Một trong những nhược điểm của biểu đồ này là yêu cầu các biến phải là biến liên tục

3. Các bước vẽ biểu đồ phân tán
Để vẽ biểu đồ phân tán thì bạn chỉ cần thực hiện theo 4 bước sau đây:

- Bước 1: Thu thập dữ liệu các cặp biến số. Số lượng các cặp biến số phải lớn hơn 30
- Bước 2: Vẽ đồ thị, trục tung là một biến số, trục hoành là kết quả của biến số đó hoặc một biến số thứ hai
- Bước 3: Biểu diễn các điểm trên đồ thị bằng các điểm thể hiện mối tương quan giữa hai biến số. Nếu các điểm trùng nhau thì dùng các ký hiệu khác nhau để phân biệt.
- Bước 4: Đánh giá mối quan hệ giữa hai biến số theo hệ số tương quan

- Nếu hệ số tương quan dương thì là mối tương quan thuận, khi đó sự gia tăng của biến số này (biến độc lập) sẽ dẫn đến sự gia tăng của biến số kia (biến phụ thuộc).

- Nếu hệ số tương quan âm thì là mối tương quan nghịch, khi đó sự gia tăng của biến số này (biến độc lập) sẽ dẫn đến sự giảm kết quả của biến số kia (biến phụ thuộc).

- Nếu hệ số tương quan bằng 0 thì là không có mối tương quan giữa hai biến.

BÀI TIẾP THEO: CONTOUR PLOT (ĐỒ THỊ ĐƯỜNG BAO) >>

Bài 19: Contour Plot (Đồ thị đường bao) - Matplotib Cơ Bản

1. Khái niệm cơ bản

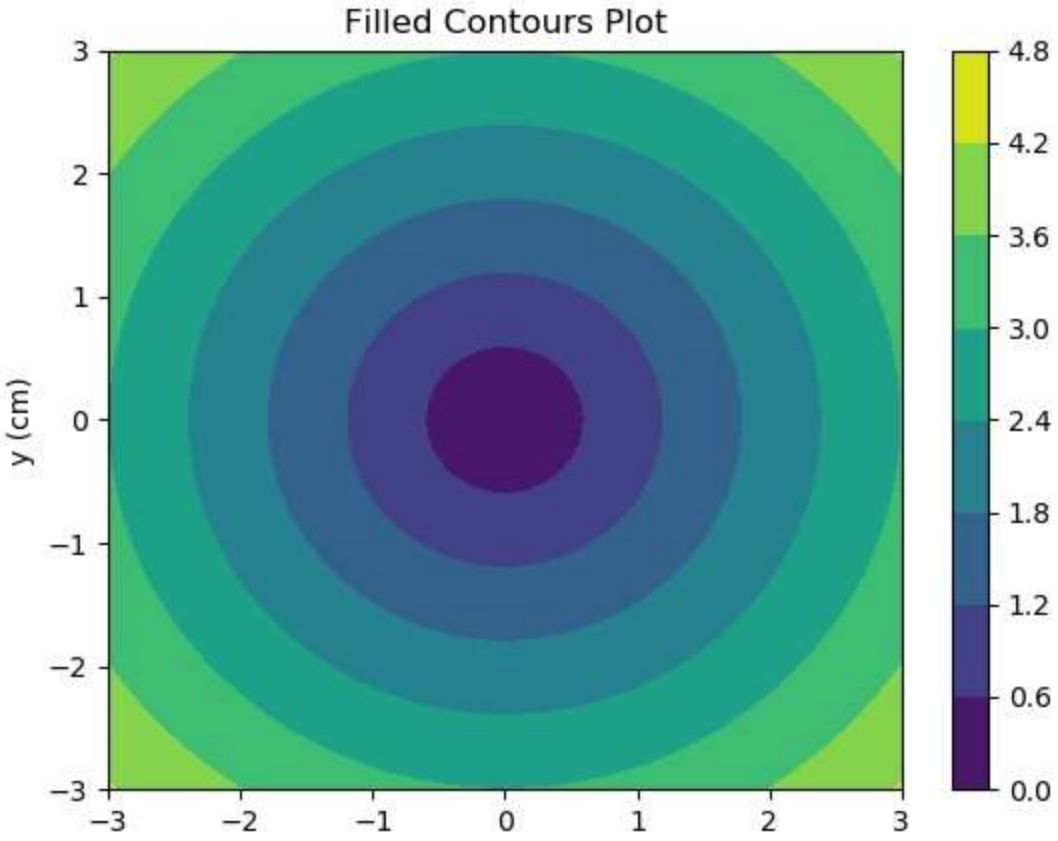
Đồ thị đường bao (đôi khi được gọi là Đồ thị mức) là một cách để thể hiện một bề mặt ba chiều trên một mặt phẳng hai chiều. Nó vẽ biểu đồ hai biến dự báo X Y trên trục y và một biến phản ứng Z dưới dạng các đường bao. Những đường bao này đôi khi được gọi là z-slice hoặc giá trị iso-response.

Một đồ thị đường bao là thích hợp nếu bạn muốn xem đường viền Z thay đổi như thế nào dưới dạng hàm của hai đầu vào X và Y, sao cho Z = f (X, Y). Đường đồng mức hoặc đường cô lập của hàm hai biến là một đường cong mà hàm có giá trị không đổi.

Các biến độc lập x và y thường bị giới hạn trong một lưới thông thường gọi là meshgrid. Numpy.meshgrid tạo ra một lưới hình chữ nhật từ một mảng các giá trị x và một mảng các giá trị y.

API Matplotlib chứa hàm contour () và contourf () để vẽ đường đồng mức và đường bao đã tô tương ứng. Cả hai hàm đều cần ba tham số x, y và 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()
```



2. Ví dụ minh hoạ :

Ví dụ 1 :

Import các thư viện cần thiết :

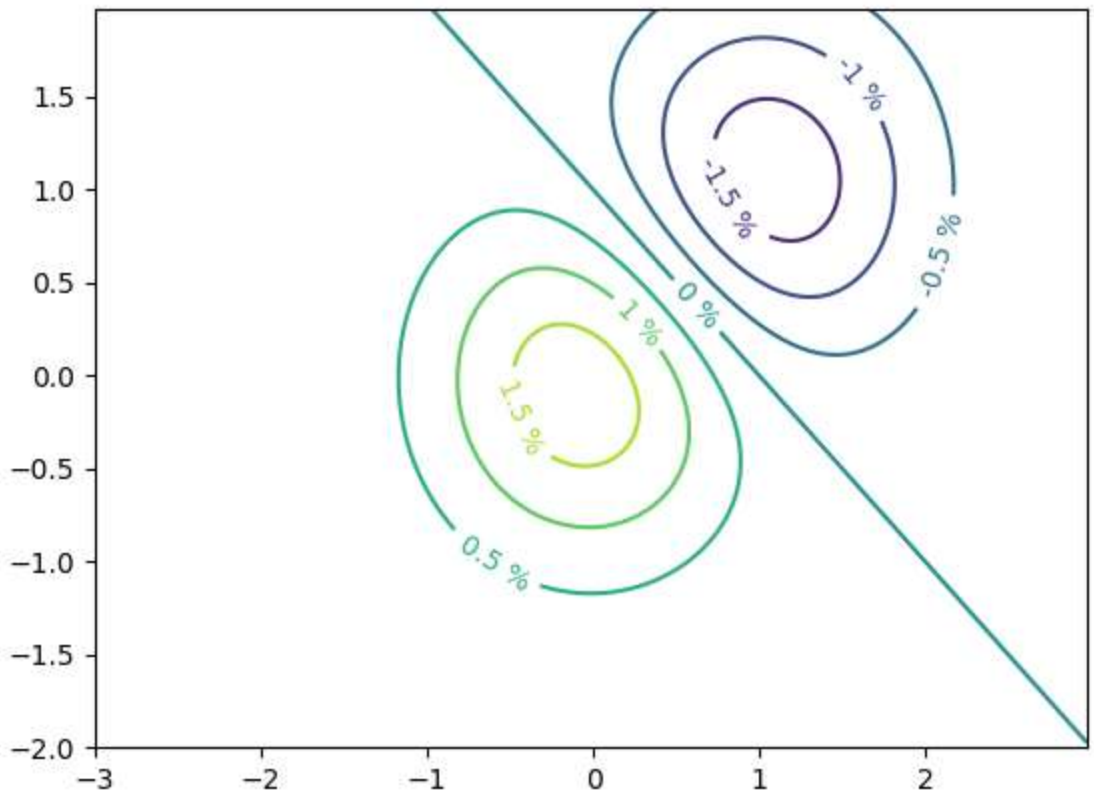
```
import matplotlib
import numpy as np
import matplotlib.ticker as ticker
import matplotlib.pyplot as plt
```

Định nghĩa các giá trị :

```
delta = 0.025
x = np.arange(-3.0, 3.0, delta)
y = np.arange(-2.0, 2.0, delta)
X, Y = np.meshgrid(x, y)
Z1 = np.exp(-X**2 - Y**2)
Z2 = np.exp(-(X - 1)**2 - (Y - 1)**2)
Z = (Z1 - Z2) * 2
```

Tạo nhãn đường viền bằng cách sử dụng các class float như sau :

```
# Define a class that forces representation of float to look a certain way
# This remove trailing zero so '1.0' becomes '1'


class nf(float):
    def __repr__(self):
        s = f'{self:.1f}'
        return f'{self:.0f}' if s[-1] == '0' else s


# Basic contour plot
fig, ax = plt.subplots()
CS = ax.contour(X, Y, Z)

# Recast levels to new class
CS.levels = [nf(val) for val in CS.levels]

# Label levels with specially formatted floats
if plt.rcParams["text.usetex"]:
    fmt = r'%r \%%'
else:
    fmt = '%r %%'

ax.clabel(CS, CS.levels, inline=True, fmt=fmt, fontsize=10)
```

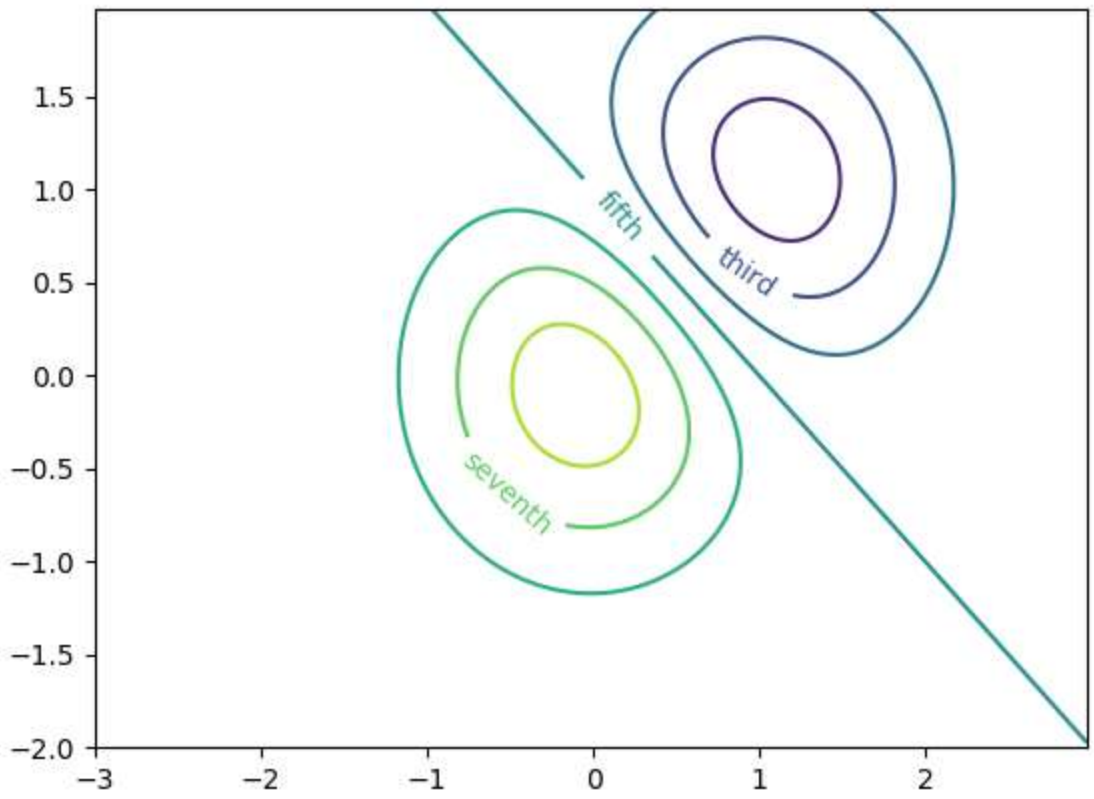


Kết quả :

```
<a list of 7 text.Text objects>
```

Gắn nhãn các đường viền bằng các chuỗi tùy ý bằng cách sử dụng dictionary

```
fig1, ax1 = plt.subplots()

# Basic contour plot
CS1 = ax1.contour(X, Y, Z)

fmt = {}
strs = ['first', 'second', 'third', 'fourth', 'fifth', 'sixth', 'seventh']
for l, s in zip(CS1.levels, strs):
    fmt[l] = s

# Label every other level using strings
ax1.clabel(CS1, CS1.levels[::2], inline=True, fmt=fmt, fontsize=10)
```

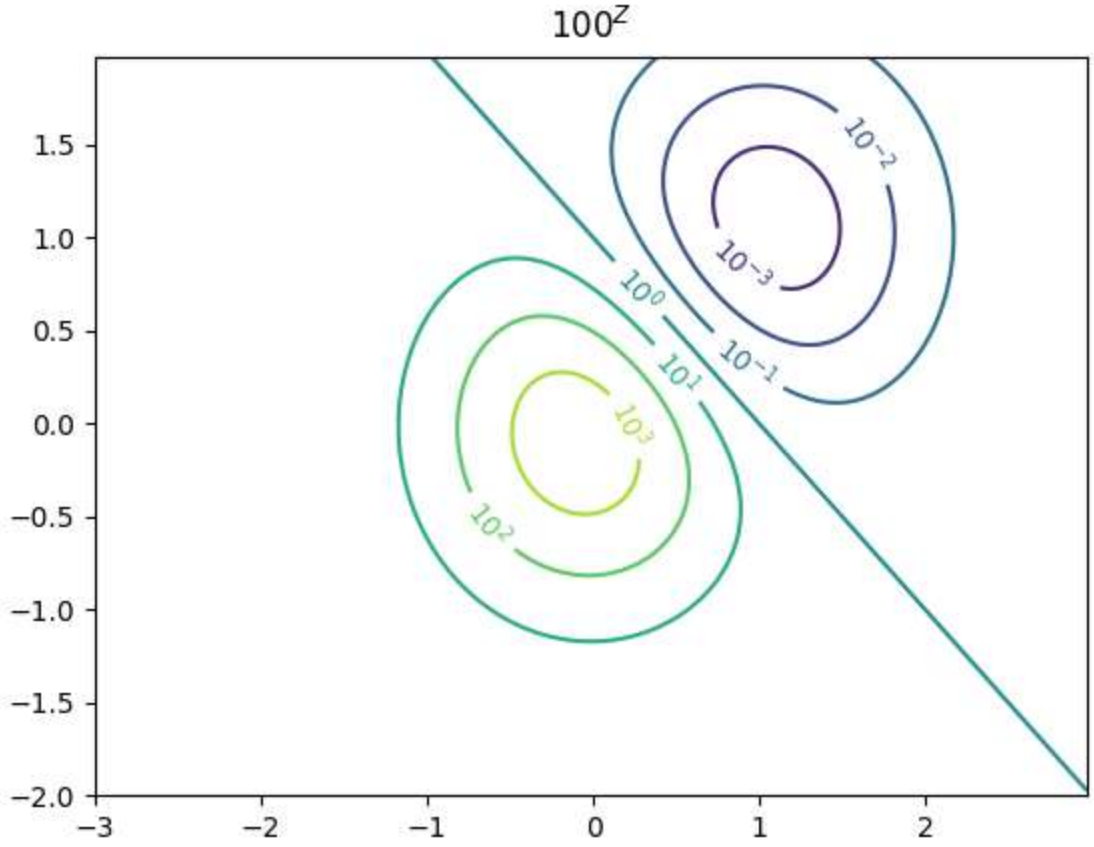


Kết quả :

```
<a list of 3 text.Text objects>
```

Sử dụng Formatter :

```
fig2, ax2 = plt.subplots()

CS2 = ax2.contour(X, Y, 100**Z, locator=plt.LogLocator())
fmt = ticker.LogFormatterMathtext()
fmt.create_dummy_axis()
ax2.clabel(CS2, CS2.levels, fmt=fmt)
ax2.set_title("$100^Z$")

plt.show()
```



Ví dụ 2 : Cách sử dụng phương thức axis.Axes.contourf () để tạo các đồ thị đường bao đã điền.

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

origin = 'lower'

delta = 0.025

x = y = np.arange(-3.0, 3.01, delta)
X, Y = np.meshgrid(x, y)
Z1 = np.exp(-X**2 - Y**2)
Z2 = np.exp(-(X - 1)**2 - (Y - 1)**2)
Z = (Z1 - Z2) * 2

nr, nc = Z.shape

# put NaNs in one corner:
Z[-nr // 6:, -nc // 6:] = np.nan
# contourf will convert these to masked
```

```
Z = np.ma.array(Z)
Z[:nr // 6, :nc // 6] = np.ma.masked

# mask a circle in the middle:
interior = np.sqrt(X**2 + Y**2) < 0.5
Z[interior] = np.ma.masked

# We are using automatic selection of contour levels;
# this is usually not such a good idea, because they don't
# occur on nice boundaries, but we do it here for purposes
# of illustration.

fig1, ax2 = plt.subplots(constrained_layout=True)
CS = ax2.contourf(X, Y, Z, 10, cmap=plt.cm.bone, origin=origin)

# Note that in the following, we explicitly pass in a subset of
# the contour levels used for the filled contours.  Alternatively,
# We could pass in additional levels to provide extra resolution,
# or leave out the levels kwarg to use all of the original levels.

CS2 = ax2.contour(CS, levels=CS.levels[::2], colors='r', origin=origin)

ax2.set_title('Nonsense (3 masked regions)')
ax2.set_xlabel('word length anomaly')
ax2.set_ylabel('sentence length anomaly')

# Make a colorbar for the ContourSet returned by the contourf call.
cbar = fig1.colorbar(CS)
cbar.ax.set_ylabel('verbosity coefficient')
# Add the contour line levels to the colorbar
cbar.add_lines(CS2)

fig2, ax2 = plt.subplots(constrained_layout=True)
# Now make a contour plot with the levels specified,
# and with the colormap generated automatically from a list
# of colors.
levels = [-1.5, -1, -0.5, 0, 0.5, 1]
CS3 = ax2.contourf(X, Y, Z, levels,
                   colors=('r', 'g', 'b'),
                   origin=origin,
                   extend='both')
# Our data range extends outside the range of levels; make
# data below the lowest contour level yellow, and above the
# highest level cyan:
```

```
CS3.cmap.set_under('yellow')
CS3.cmap.set_over('cyan')

CS4 = ax2.contour(X, Y, Z, levels,
                  colors=('k',),
                  linewidths=(3,),
                  origin=origin)
ax2.set_title('Listed colors (3 masked regions)')
ax2.clabel(CS4, fmt='%2.1f', colors='w', fontsize=14)

# Notice that the colorbar command gets all the information it
# needs from the ContourSet object, CS3.
fig2.colorbar(CS3)

# Illustrate all 4 possible "extend" settings:
extends = ["neither", "both", "min", "max"]
cmap = plt.cm.get_cmap("winter")
cmap.set_under("magenta")
cmap.set_over("yellow")
# Note: contouring simply excludes masked or nan regions, so
# instead of using the "bad" colormap value for them, it draws
# nothing at all in them.  Therefore the following would have
# no effect:
# cmap.set_bad("red")

fig, axs = plt.subplots(2, 2, constrained_layout=True)

for ax, extend in zip(axs.ravel(), extends):
    cs = ax.contourf(X, Y, Z, levels, cmap=cmap, extend=extend, origin=origin)
    fig.colorbar(cs, ax=ax, shrink=0.9)
    ax.set_title("extend = %s" % extend)
    ax.locator_params(nbins=4)

plt.show()
```

Bài 20: Quiver Plot - Matplotib Cơ Bản
Đăng bởi: Admin | Lượt xem: 1682 | Chuyên mục: AI

---

1. Khái niệm cơ bản :

Biểu đồ quiver hiển thị các vectơ vận tốc dưới dạng mũi tên với các thành phần (u, v) tại các điểm (x, y).

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

Đoạn code trên vẽ các vectơ dưới dạng arrow tại các tọa độ được chỉ định trong mỗi cặp phần tử tương ứng trong x và y.

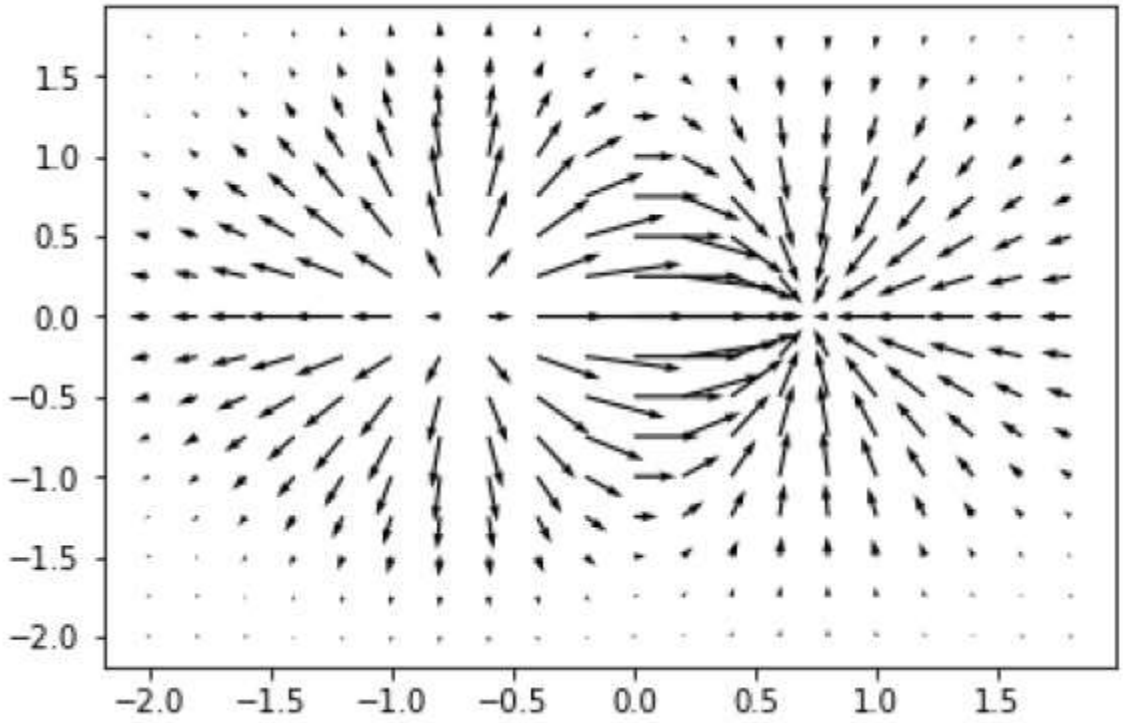
Các tham số :

Bảng sau liệt kê các thông số khác nhau cho plot Quiver:

x	**Mảng 1D hoặc 2D. Tọa độ x tại các vị trí arrow**
y	Mảng 1D hoặc 2D. Tọa độ y tại các vị trí arrow
u	Mảng 1D hoặc 2D. Các thành phần x của vectơ arrow
v	
c	

Ví dụ :

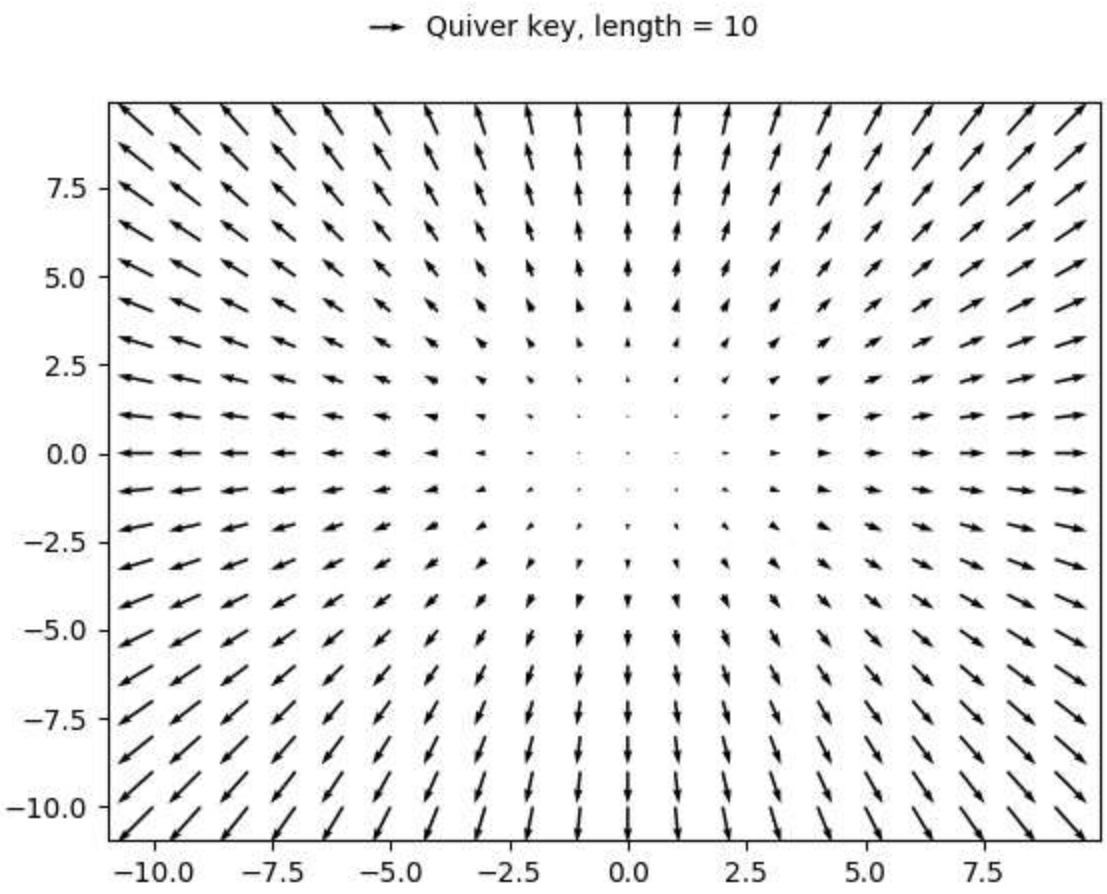
```
import matplotlib.pyplot as plt
import numpy as np
x,y = np.meshgrid(np.arange(-2, 2, .2), np.arange(-2, 2, .25))
z = x*np.exp(-x**2 - y**2)
v, u = np.gradient(z, .2, .2)
fig, ax = plt.subplots()
q = ax.quiver(x,y,u,v)
plt.show()
```



2. Các ví dụ minh hoạ :

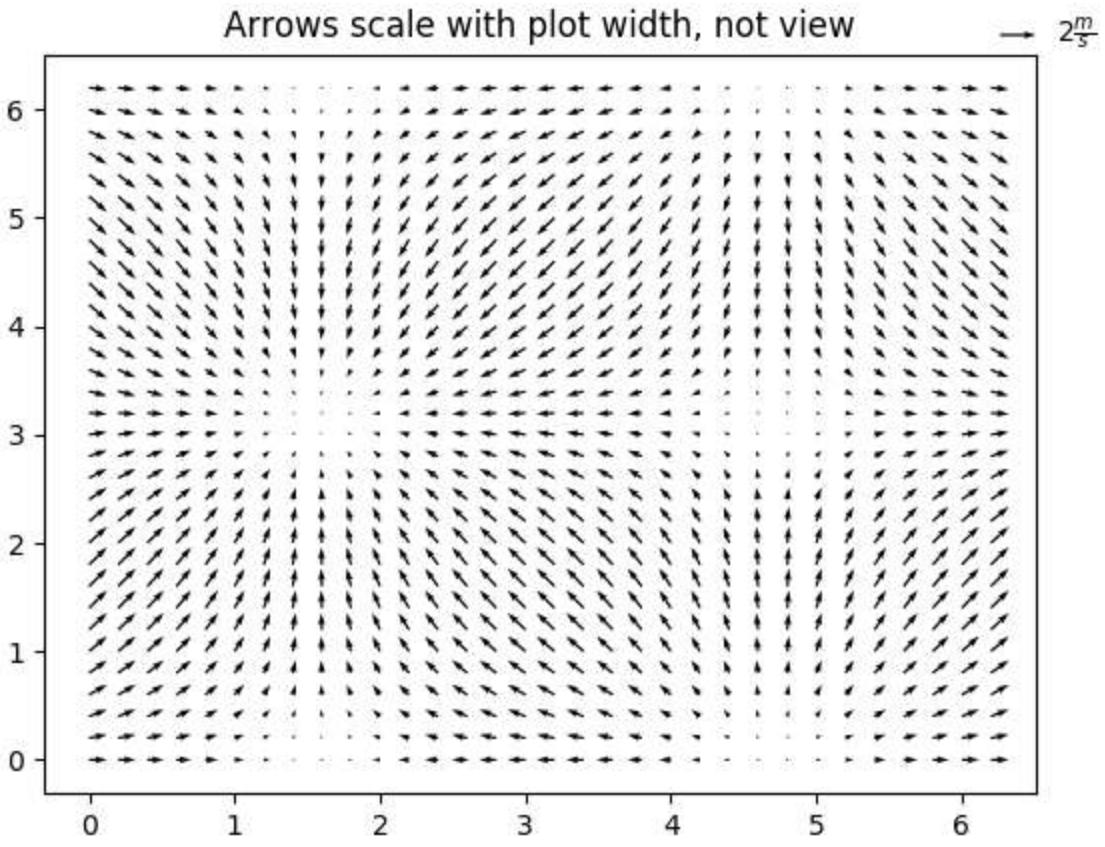
Ví dụ 1 :

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

X = np.arange(-10, 10, 1)
Y = np.arange(-10, 10, 1)
U, V = np.meshgrid(X, Y)

fig, ax = plt.subplots()
q = ax.quiver(X, Y, U, V)
ax.quiverkey(q, X=0.3, Y=1.1, U=10,
             label='Quiver key, length = 10', labelpos='E')

plt.show()
```

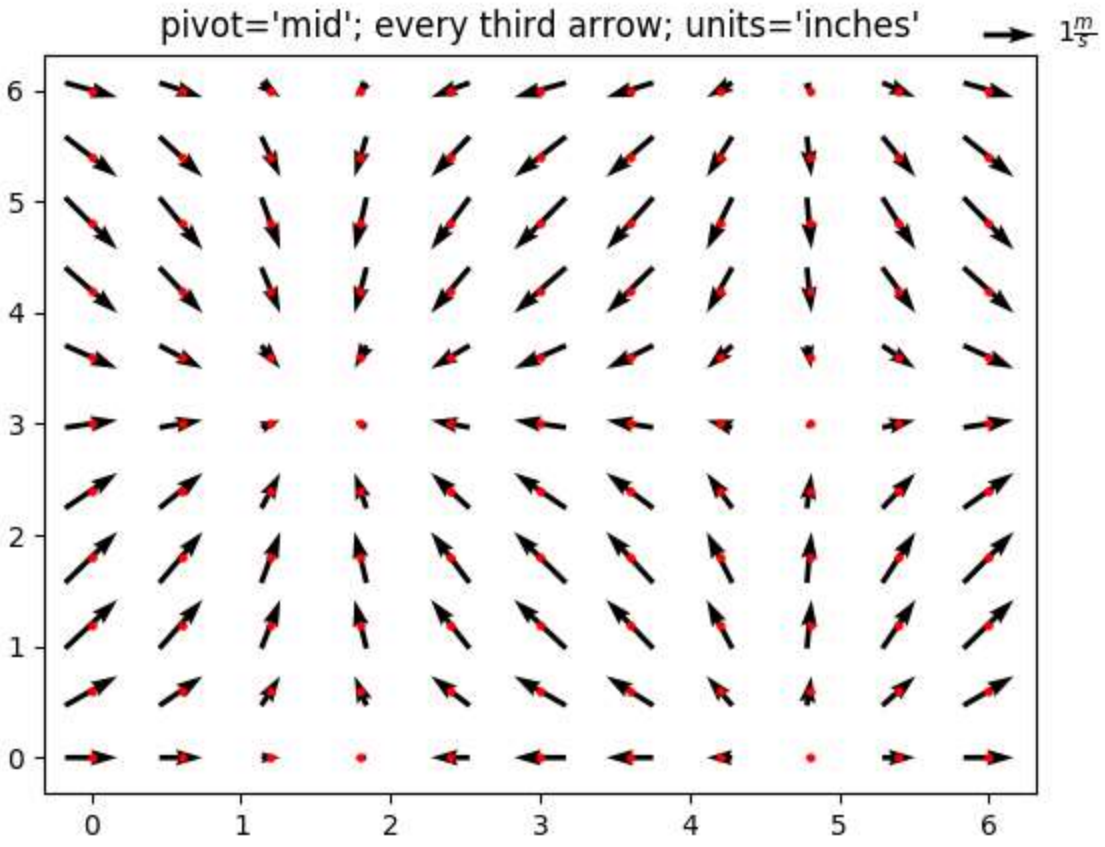


Ví dụ 2 :

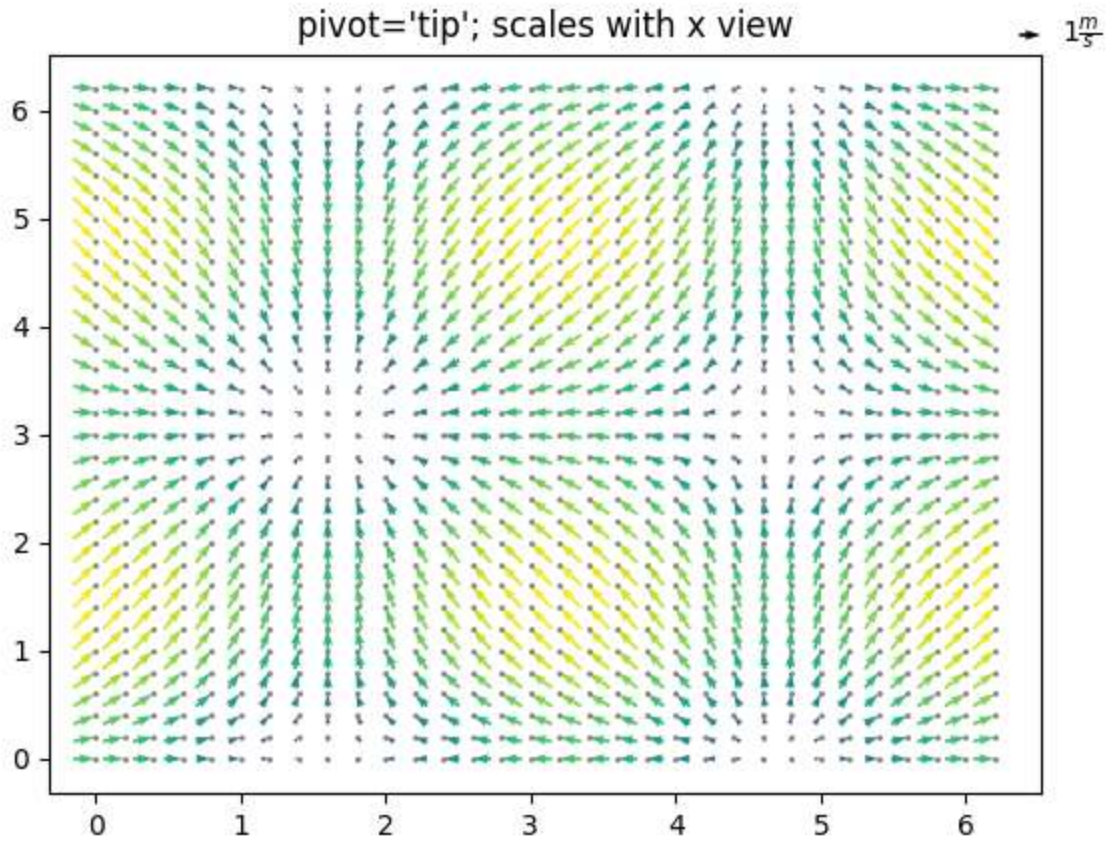
```
import matplotlib.pyplot as plt
import numpy as np

X, Y = np.meshgrid(np.arange(0, 2 * np.pi, .2), np.arange(0, 2 * np.pi, .2))
U = np.cos(X)
V = np.sin(Y)
fig1, ax1 = plt.subplots()
ax1.set_title('Arrows scale with plot width, not view')
Q = ax1.quiver(X, Y, U, V, units='width')
qk = ax1.quiverkey(Q, 0.9, 0.9, 2, r'$2 \frac{m}{s}$', labelpos='E',
                   coordinates='figure')
```




```
fig2, ax2 = plt.subplots()
ax2.set_title("pivot='mid'; every third arrow; units='inches'")
Q = ax2.quiver(X[::3, ::3], Y[::3, ::3], U[::3, ::3], V[::3, ::3],
               pivot='mid', units='inches')
qk = ax2.quiverkey(Q, 0.9, 0.9, 1, r'$1 \frac{m}{s}$', labelpos='E',
                   coordinates='figure')
ax2.scatter(X[::3, ::3], Y[::3, ::3], color='r', s=5)
```



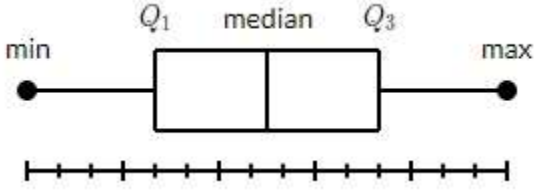

```
# sphinx_gallery_thumbnail_number = 3

fig3, ax3 = plt.subplots()
ax3.set_title("pivot='tip'; scales with x view")
M = np.hypot(U, V)
Q = ax3.quiver(X, Y, U, V, M, units='x', pivot='tip', width=0.022,
               scale=1 / 0.15)
qk = ax3.quiverkey(Q, 0.9, 0.9, 1, r'$1 \frac{m}{s}$', labelpos='E',
                   coordinates='figure')
ax3.scatter(X, Y, color='0.5', s=1)

plt.show()
```



Bài 21: Box Plot (Biểu đồ nén) - Matplotib Cơ Bản
Đăng bởi: Admin | Lượt xem: 3747 | Chuyên mục: AI

1. khái niệm cơ bản :

Biểu đồ nén hiển thị tóm tắt tập hợp dữ liệu có chứa minimum, phần tư thứ nhất, trung vị, phần tư thứ ba và maximum. Một đường thẳng đứng đi qua hộp ở trung tuyến. Râu đi từ mỗi phần tư đến nhỏ nhất hoặc tối đa.



Box Plot (Biểu đồ nén) - Matplotib Cơ Bản
Đăng bởi: Admin | Lượt xem: 3747 | Chuyên mục: AI

1. khái niệm cơ bản :

Biểu đồ nén hiển thị tóm tắt tập hợp dữ liệu có chứa minimum, phần tư thứ nhất, trung vị, phần tư thứ ba và maximum. Một đường thẳng đứng đi qua hộp ở trung tuyến. Râu đi từ mỗi phần tư đến nhỏ nhất hoặc tối đa.

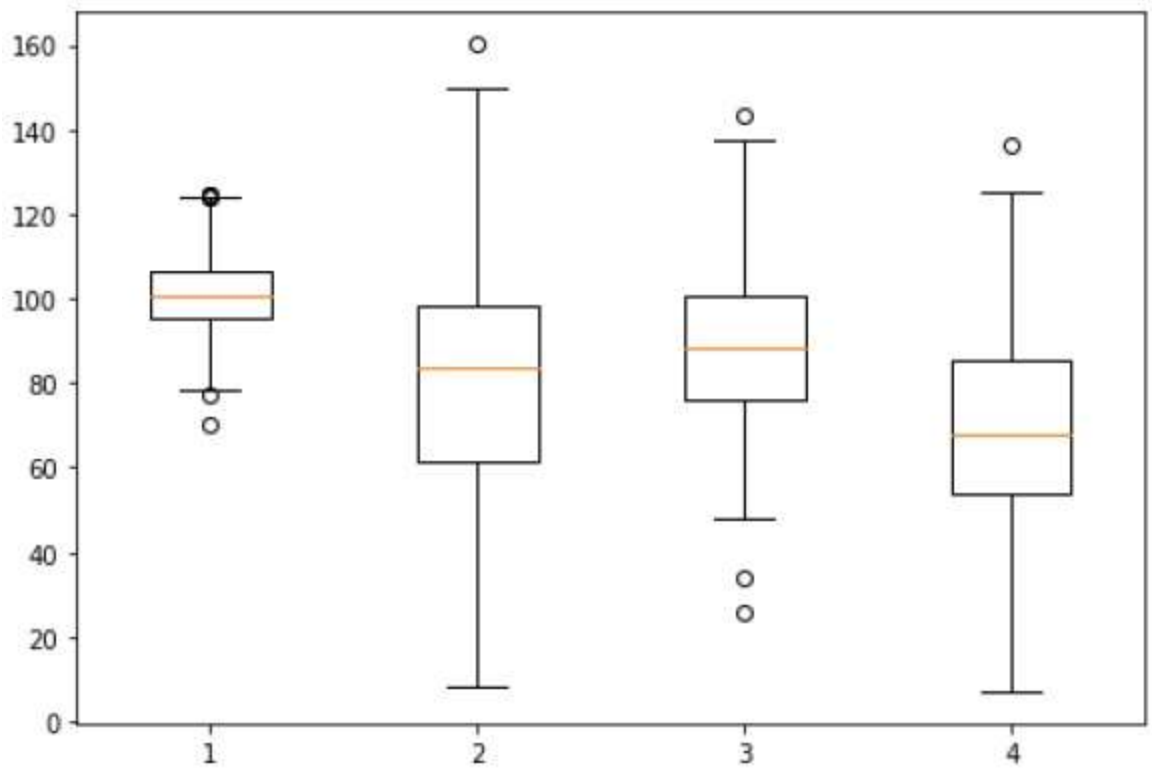
Ta tiến hành tạo biểu đồ box. Ta sử dụng hàm numpy.random.normal () để tạo dữ liệu giả. Nó cần ba đối số, giá trị trung bình , độ lệch chuẩn của phân phối chuẩn và số lượng giá trị mong muốn.

```
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)
```

Danh sách các mảng mà ta đã tạo ở trên là đầu vào bắt buộc duy nhất để tạo boxplot. Sử dụng đoạn code data_to_plot :

```
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()
```

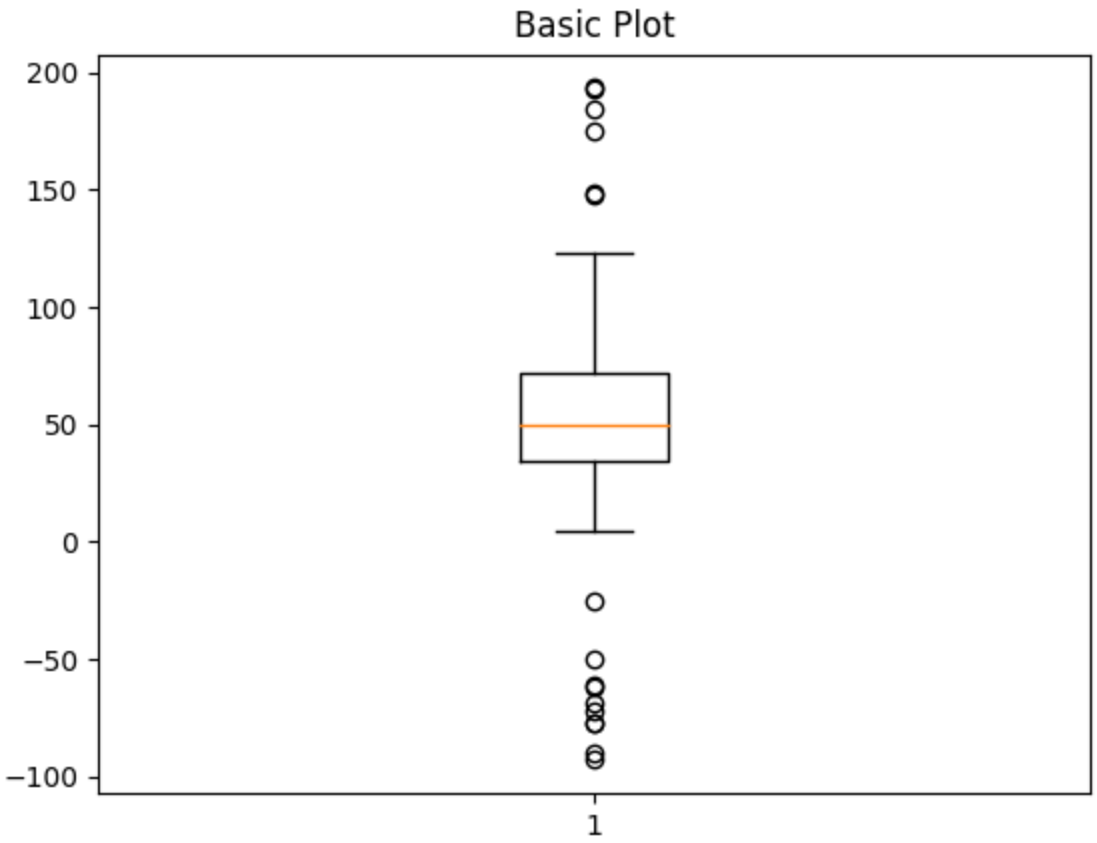
Kết quả như sau :



2. Ví dụ :

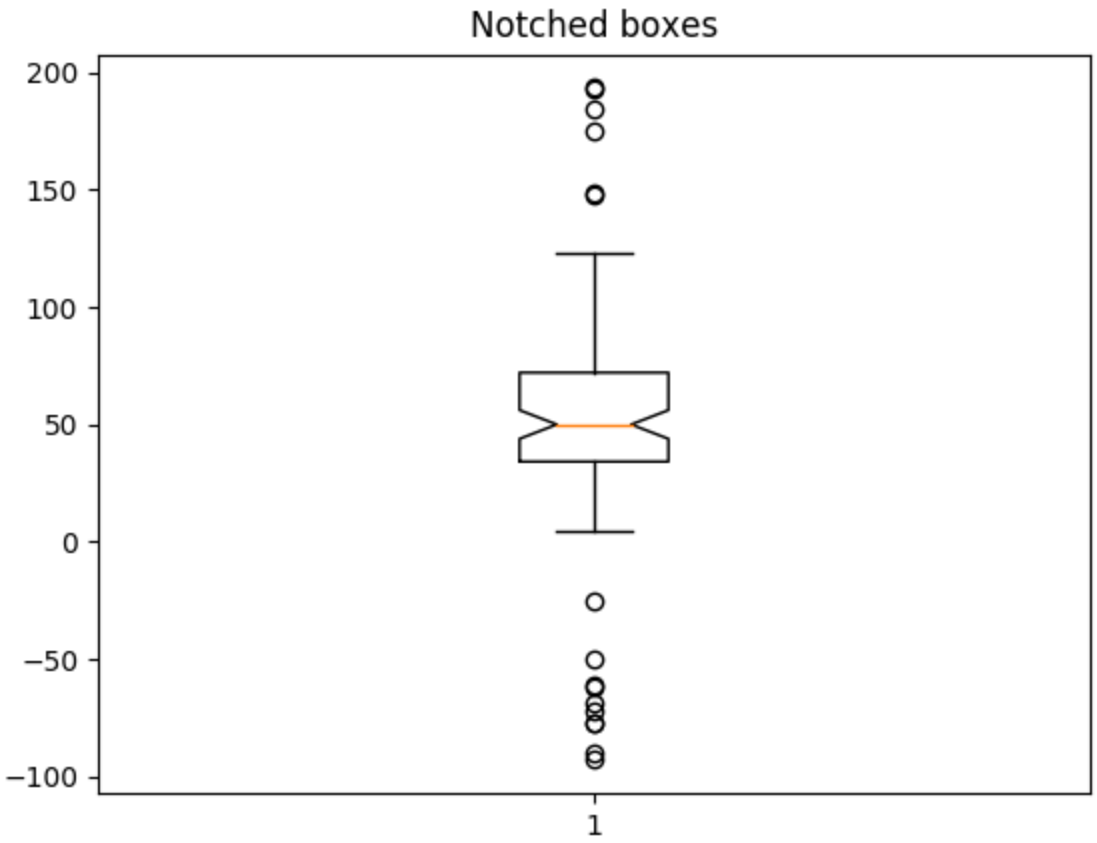
Các ví dụ cơ bản về biểu đồ nén :

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

# Fixing random state for reproducibility
np.random.seed(19680801)

# fake up some data
spread = np.random.rand(50) * 100
center = np.ones(25) * 50
flier_high = np.random.rand(10) * 100 + 100
flier_low = np.random.rand(10) * -100
data = np.concatenate((spread, center, flier_high, flier_low))
fig1, ax1 = plt.subplots()
ax1.set_title('Basic Plot')
ax1.boxplot(data)
```



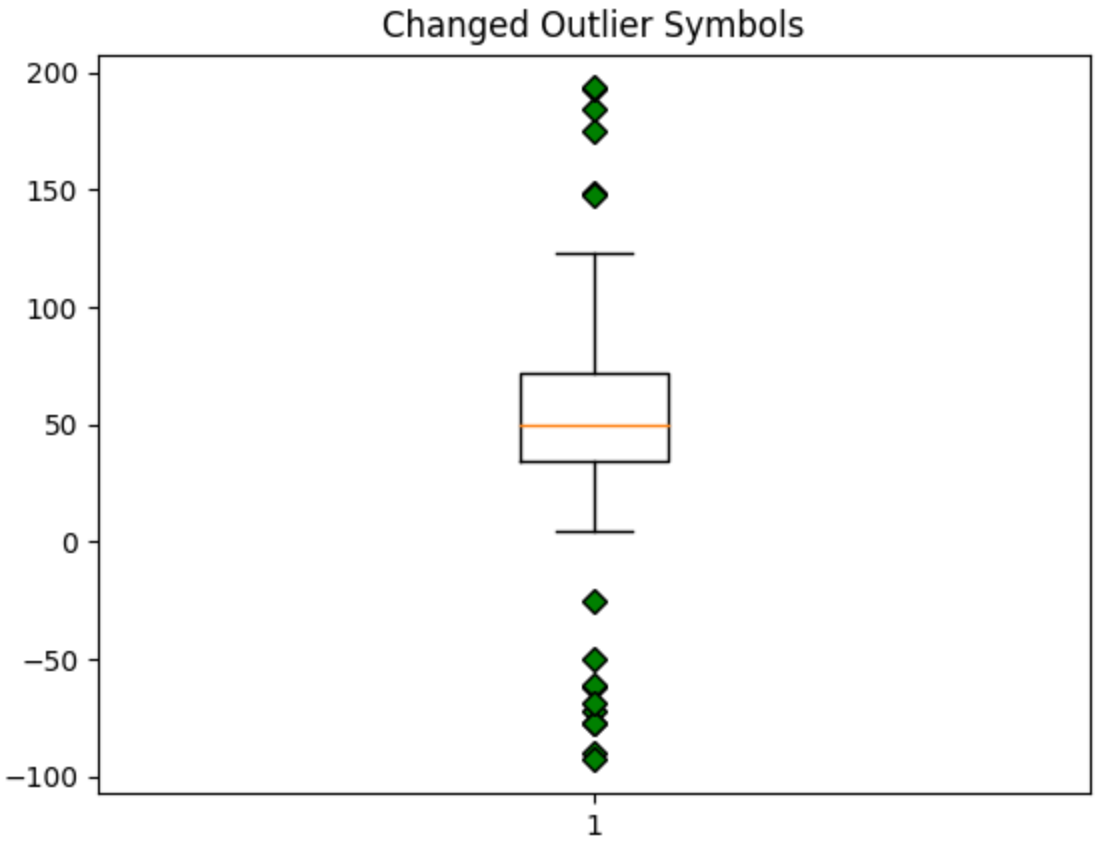
Kết quả :

```
{'whiskers': [<matplotlib.lines.Line2D object at 0x7fb114c3df60>, <matplotlib.lines.Line2D object at 0x7fb114c3d048>], 'caps': [<matplotlib.lines.Line2D object at 0x7fb114c3d8d0>, <matplotlib.lines.Line2D object at 0x7fb114c13390>], 'boxes': [<matplotlib.lines.Line2D object at 0x7fb114c3dcc0>], 'medians': [<matplotlib.lines.Line2D object at 0x7fb1146f4518>], 'fliers': [<matplotlib.lines.Line2D object at 0x7fb1146f42e8>], 'means': []}
fig2, ax2 = plt.subplots()
ax2.set_title('Notched boxes')
ax2.boxplot(data, notch=True)
```



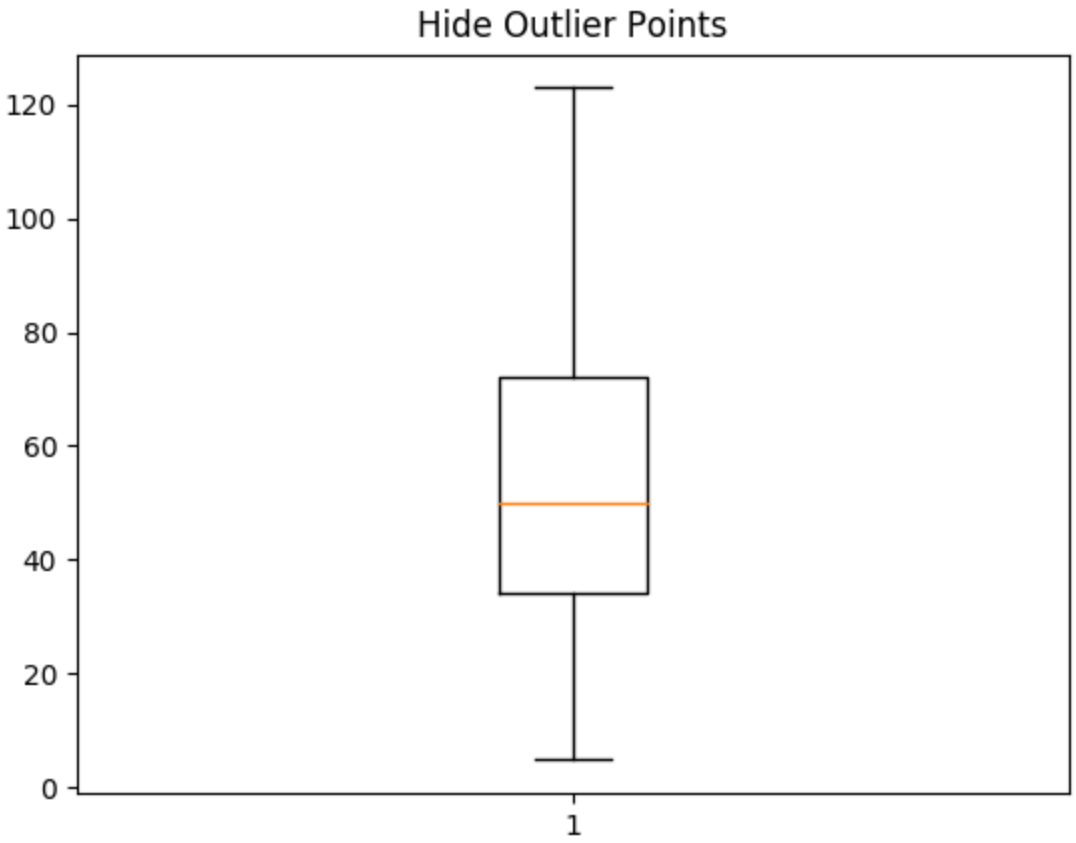
Kết quả :

```
{'whiskers': [<matplotlib.lines.Line2D object at 0x7fb113e887f0>, <matplotlib.lines.Line2D object at 0x7fb113e88ba8>], 'caps': [<matplotlib.lines.Line2D object at 0x7fb113e88ef0>, <matplotlib.lines.Line2D object at 0x7fb113e7e278>], 'boxes': [<matplotlib.lines.Line2D object at 0x7fb113e886a0>], 'medians': [<matplotlib.lines.Line2D object at 0x7fb113e7e5c0>], 'fliers': [<matplotlib.lines.Line2D object at 0x7fb113e7e908>], 'means': []}
green_diamond = dict(markerfacecolor='g', marker='D')
fig3, ax3 = plt.subplots()
ax3.set_title('Changed Outlier Symbols')
ax3.boxplot(data, flierprops=green_diamond)
```



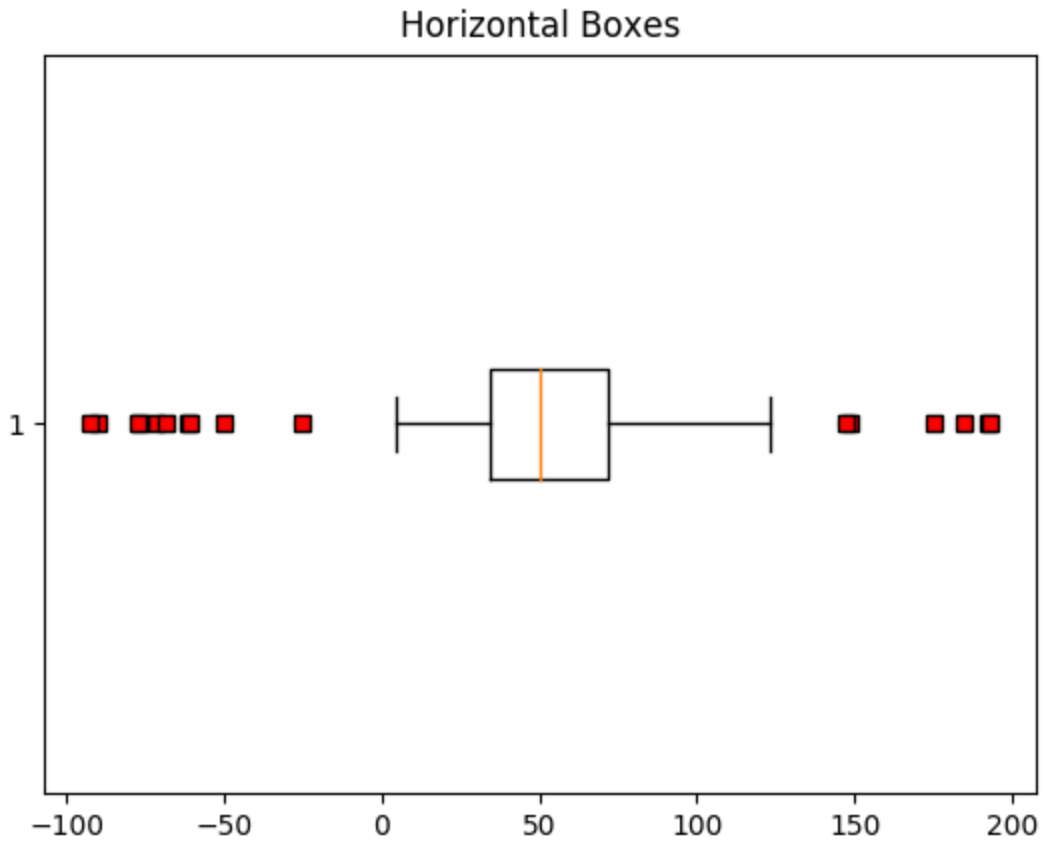
Kết quả :

```
{'whiskers': [<matplotlib.lines.Line2D object at 0x7fb113c908d0>, <matplotlib.lines.Line2D object at 0x7fb113c90c88>], 'caps': [<matplotlib.lines.Line2D object at 0x7fb113c90fd0>, <matplotlib.lines.Line2D object at 0x7fb113db1358>], 'boxes': [<matplotlib.lines.Line2D object at 0x7fb113c90780>], 'medians': [<matplotlib.lines.Line2D object at 0x7fb113db16a0>], 'fliers': [<matplotlib.lines.Line2D object at 0x7fb113db19e8>], 'means': []}
fig4, ax4 = plt.subplots()
ax4.set_title('Hide Outlier Points')
ax4.boxplot(data, showfliers=False)
```



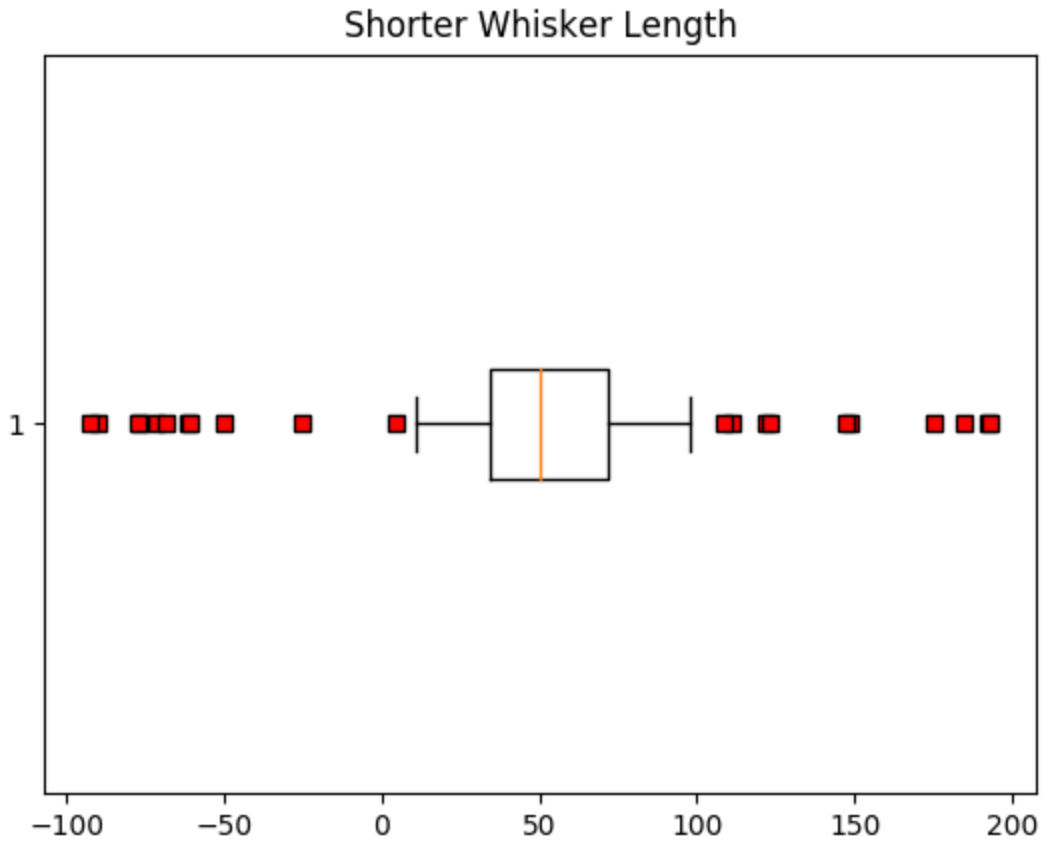
Kết quả :

```
{'whiskers': [<matplotlib.lines.Line2D object at 0x7fb113b87518>, <matplotlib.lines.Line2D object at 0x7fb113b878d0>], 'caps': [<matplotlib.lines.Line2D object at 0x7fb113b87c18>, <matplotlib.lines.Line2D object at 0x7fb113b87f60>], 'boxes': [<matplotlib.lines.Line2D object at 0x7fb113b873c8>], 'medians': [<matplotlib.lines.Line2D object at 0x7fb113b61160>], 'fliers': [], 'means': []}
red_square = dict(markerfacecolor='r', marker='s')
fig5, ax5 = plt.subplots()
ax5.set_title('Horizontal Boxes')
ax5.boxplot(data, vert=False, flierprops=red_square)
```



Kết quả :

```
{'whiskers': [<matplotlib.lines.Line2D object at 0x7fb113d842e8>, <matplotlib.lines.Line2D object at 0x7fb113d846a0>], 'caps': [<matplotlib.lines.Line2D object at 0x7fb113d849e8>, <matplotlib.lines.Line2D object at 0x7fb113d84d30>], 'boxes': [<matplotlib.lines.Line2D object at 0x7fb113d84198>], 'medians': [<matplotlib.lines.Line2D object at 0x7fb113d680b8>], 'fliers': [<matplotlib.lines.Line2D object at 0x7fb113d68400>], 'means': []}
fig6, ax6 = plt.subplots()
ax6.set_title('Shorter Whisker Length')
ax6.boxplot(data, flierprops=red_square, vert=False, whis=0.75)
```



Kết quả :

```
{'whiskers': [<matplotlib.lines.Line2D object at 0x7fb113e304a8>, <matplotlib.lines.Line2D object at 0x7fb113e31358>], 'caps': [<matplotlib.lines.Line2D object at 0x7fb113e316a0>, <matplotlib.lines.Line2D object at 0x7fb113e319e8>], 'boxes': [<matplotlib.lines.Line2D object at 0x7fb113e305c0>], 'medians': [<matplotlib.lines.Line2D object at 0x7fb113e31d30>], 'fliers': [<matplotlib.lines.Line2D object at 0x7fb113e4b0b8>], 'means': []}
```

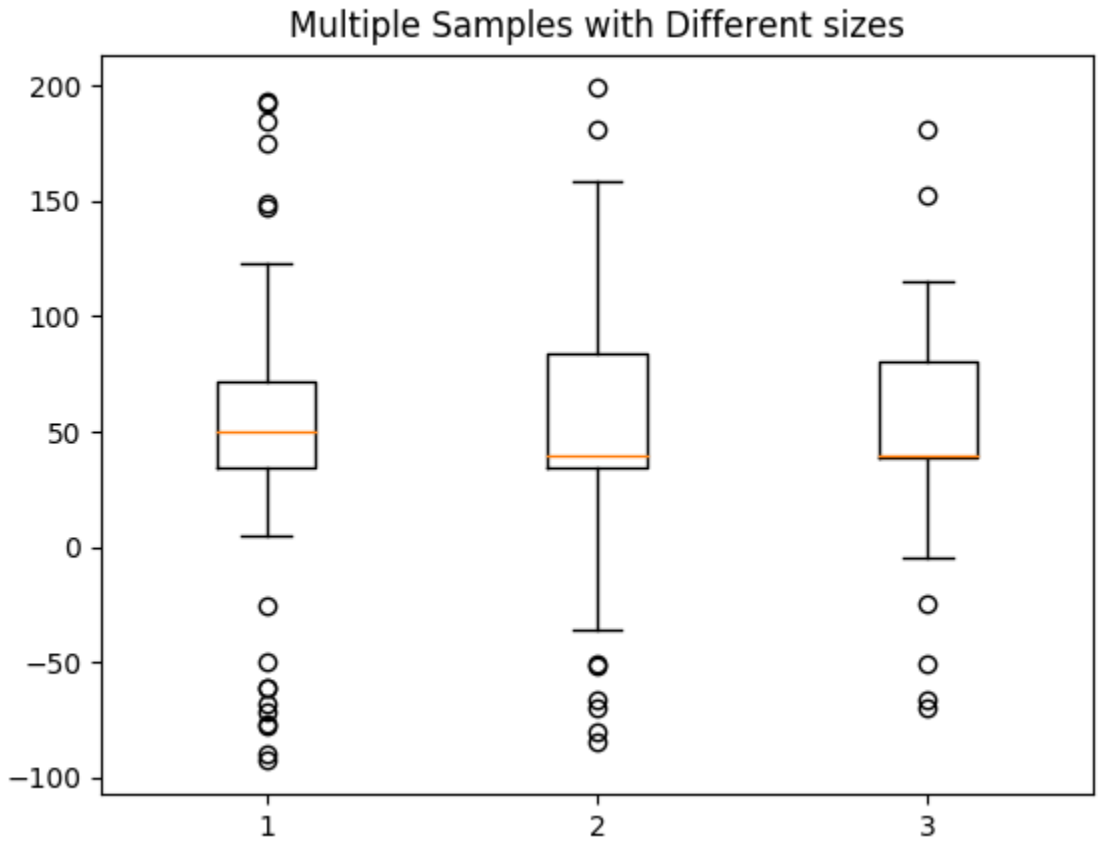
Ta tiến hành tạo thêm dữ liệu giả :

```
spread = np.random.rand(50) * 100
center = np.ones(25) * 40
flier_high = np.random.rand(10) * 100 + 100
flier_low = np.random.rand(10) * -100
d2 = np.concatenate((spread, center, flier_high, flier_low))
data.shape = (-1, 1)
d2.shape = (-1, 1)
```

Tạo mảng 2-D chỉ hoạt động nếu tất cả các cột có cùng độ dài. Nếu không, hãy sử dụng một danh sách để thay thế. Điều này thực sự hiệu quả hơn vì dù sao thì boxplot cũng chuyển đổi mảng 2-D thành danh sách các vectơ bên trong.

```
data = [data, d2, d2[::2,0]]
```

```
fig7, ax7 = plt.subplots()
ax7.set_title('Multiple Samples with Different sizes')
ax7.boxplot(data)

plt.show()
```



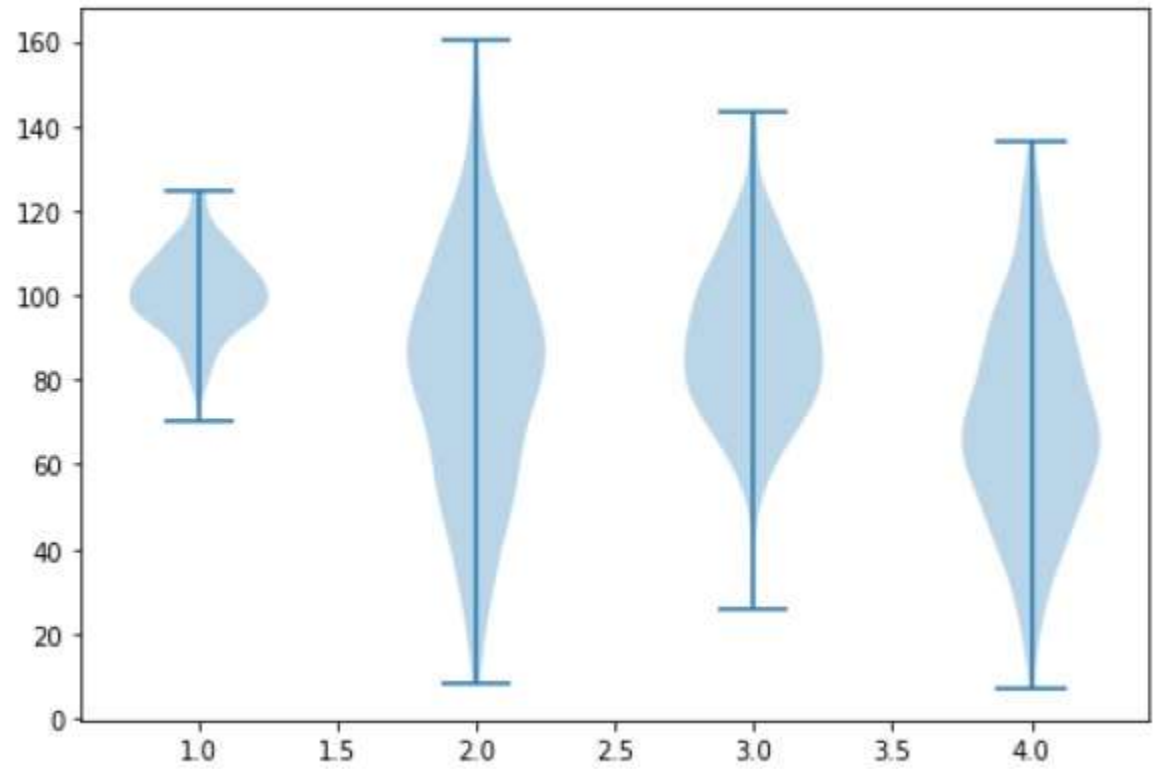
Bài 22: Violin Plot - Matplotib Cơ Bản
Đăng bởi: Admin | Lượt xem: 2724 | Chuyên mục: AI

1. Khái niệm

Biểu đồ violin tương tự như biểu đồ nến, ngoại trừ việc chúng hiển thị mật độ xác suất của dữ liệu ở các giá trị khác nhau. Các biểu đồ này bao gồm một điểm đánh dấu cho trung vị của dữ liệu và một nến hiển thị phạm vi liên phần tư, như trong các plot box tiêu chuẩn. Phủ trên plot box này là ước tính mật độ nhân. Giống như box plot, plot violin được sử dụng để thể hiện sự so sánh của một phân phối thay đổi (hoặc phân phối mẫu) trên các "danh mục" khác nhau.

Một plot violin có nhiều thông tin hơn một plot box đơn thuần. Trên thực tế, trong khi biểu đồ nến chỉ hiển thị thống kê tóm tắt như phạm vi trung bình / trung vị và giữa các phần, thì biểu đồ violin hiển thị toàn bộ phân phối dữ liệu.

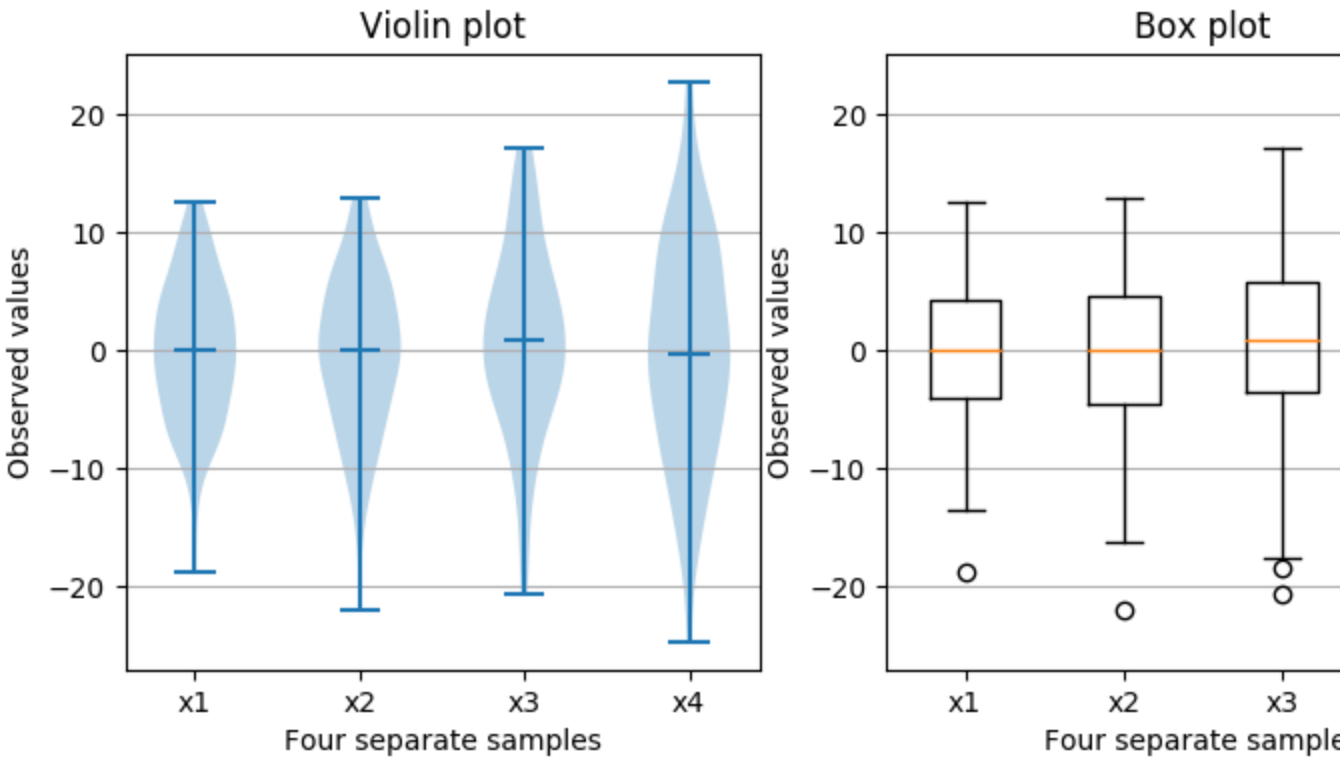
```
import matplotlib.pyplot as plt

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)

## combine these different collections into a list
data_to_plot = [collectn_1, collectn_2, collectn_3, collectn_4]

# Create a figure instance
fig = plt.figure()

# Create an axes instance
ax = fig.add_axes([0,0,1,1])

# Create the boxplot
bp = ax.violinplot(data_to_plot)
plt.show()
```



2. Ví dụ :

Ví dụ 1 : so sánh giữa biểu đồ violin và biểu đồ nến :

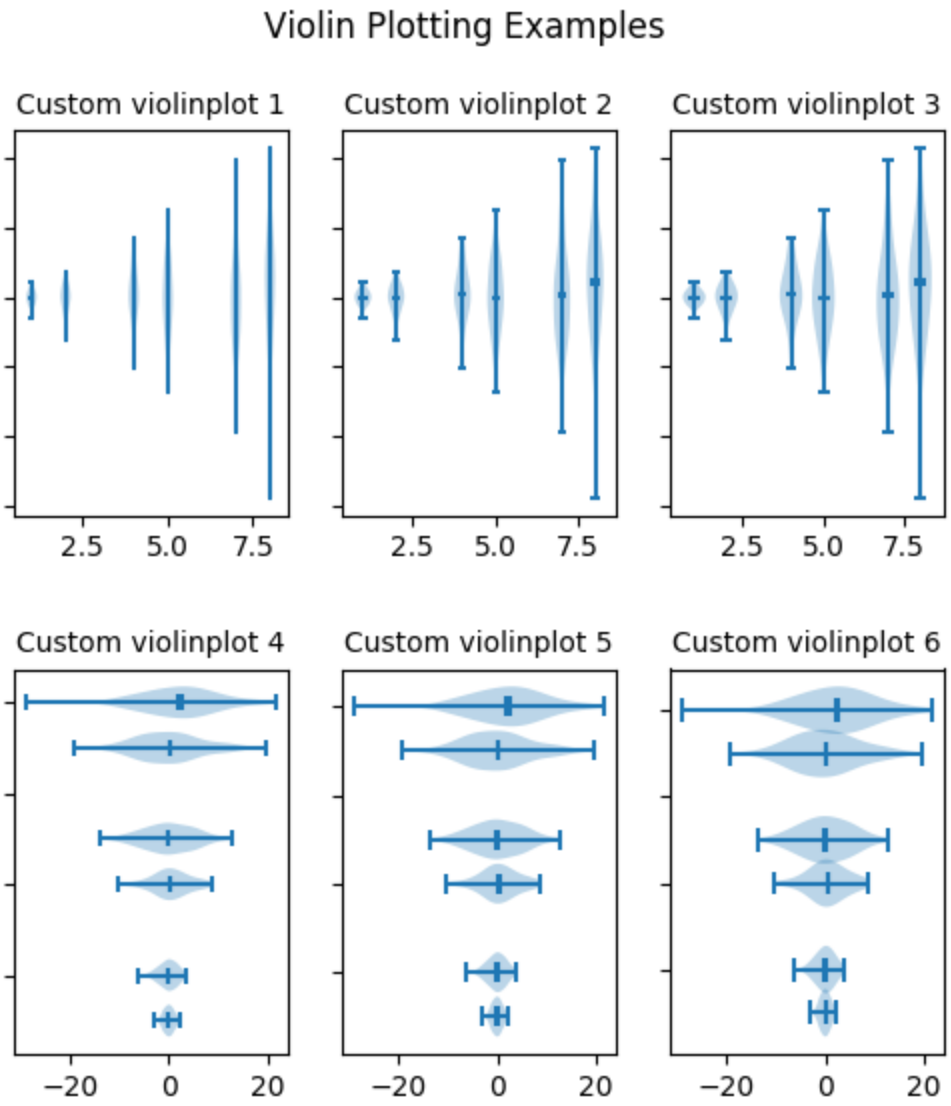
```
import matplotlib.pyplot as plt
import numpy as np

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(9, 4))

# Fixing random state for reproducibility
np.random.seed(19680801)


# generate some random test data
all_data = [np.random.normal(0, std, 100) for std in range(6, 10)]

# plot violin plot
axes[0].violinplot(all_data,
                   showmeans=False,
                   showmedians=True)
axes[0].set_title('Violin plot')

# plot box plot
axes[1].boxplot(all_data)
axes[1].set_title('Box plot')

# adding horizontal grid lines
for ax in axes:
    ax.yaxis.grid(True)
    ax.set_xticks([y + 1 for y in range(len(all_data))])
    ax.set_xlabel('Four separate samples')
    ax.set_ylabel('Observed values')

# add x-tick labels
plt.setp(axes, xticks=[y + 1 for y in range(len(all_data))],
         xticklabels=['x1', 'x2', 'x3', 'x4'])
plt.show()
```



Ví dụ 2 :

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

# Fixing random state for reproducibility
np.random.seed(19680801)


# fake data
fs = 10  # fontsize
pos = [1, 2, 4, 5, 7, 8]
data = [np.random.normal(0, std, size=100) for std in pos]

fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(6, 6))

axes[0, 0].violinplot(data, pos, points=20, widths=0.3,
                      showmeans=True, showextrema=True, showmedians=True)
axes[0, 0].set_title('Custom violinplot 1', fontsize=fs)

axes[0, 1].violinplot(data, pos, points=40, widths=0.5,
                      showmeans=True, showextrema=True, showmedians=True,
                      bw_method='silverman')
axes[0, 1].set_title('Custom violinplot 2', fontsize=fs)
```

```
axes[0, 2].violinplot(data, pos, points=60, widths=0.7, showmeans=True,
                      showextrema=True, showmedians=True, bw_method=0.5)
axes[0, 2].set_title('Custom violinplot 3', fontsize=fs)

axes[1, 0].violinplot(data, pos, points=80, vert=False, widths=0.7,
                      showmeans=True, showextrema=True, showmedians=True)
axes[1, 0].set_title('Custom violinplot 4', fontsize=fs)

axes[1, 1].violinplot(data, pos, points=100, vert=False, widths=0.9,
                      showmeans=True, showextrema=True, showmedians=True,
                      bw_method='silverman')
axes[1, 1].set_title('Custom violinplot 5', fontsize=fs)

axes[1, 2].violinplot(data, pos, points=200, vert=False, widths=1.1,
                      showmeans=True, showextrema=True, showmedians=True,
                      bw_method=0.5)
axes[1, 2].set_title('Custom violinplot 6', fontsize=fs)

for ax in axes.flat:
    ax.set_yticklabels([])

fig.suptitle("Violin Plotting Examples")
fig.subplots_adjust(hspace=0.4)
plt.show()
```



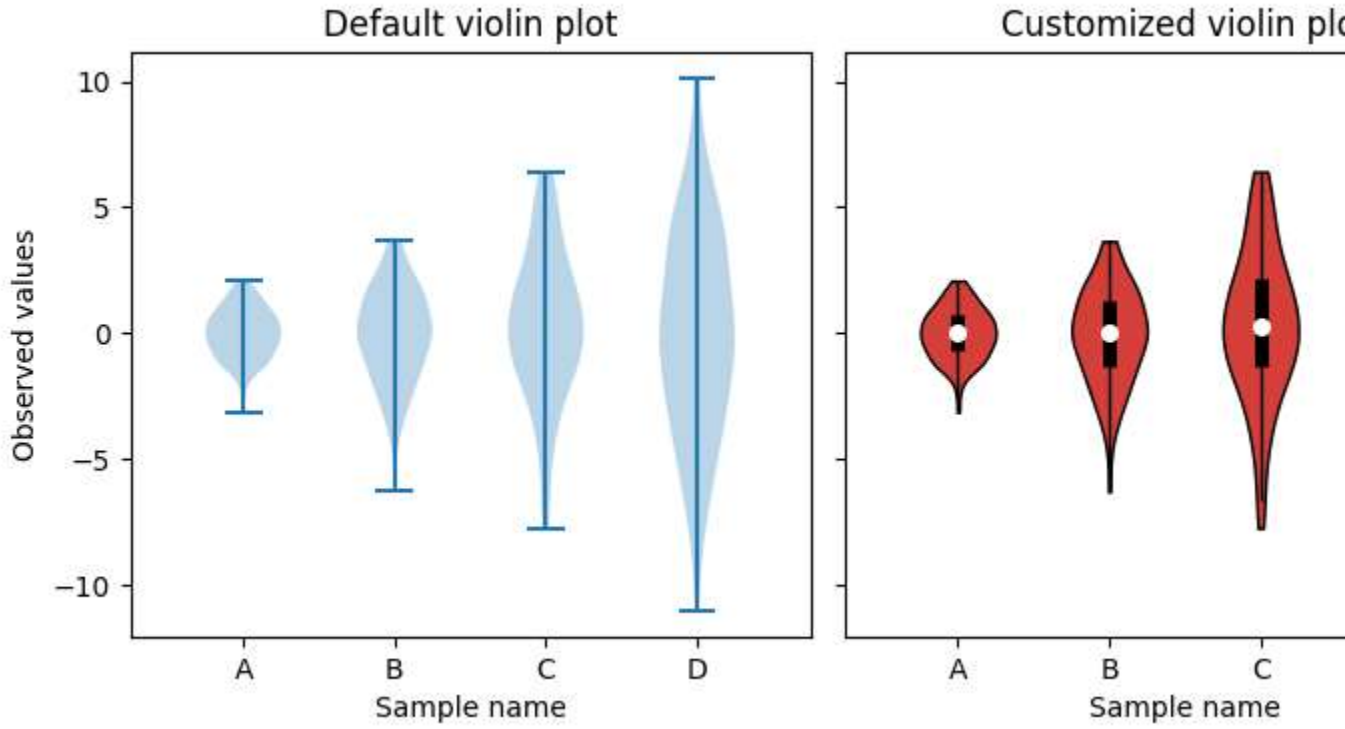
Ví dụ 3 : Điều chỉnh biểu đồ violin

Ví dụ này trình bày cách tùy chỉnh hoàn toàn các plot violin. Biểu đồ đầu tiên hiển thị kiểu mặc định bằng cách chỉ cung cấp dữ liệu. Biểu đồ thứ hai đầu tiên giới hạn những gì matplotlib vẽ với các kwargs bổ sung. Sau đó, một biểu diễn đơn giản của một biểu đồ nến được vẽ trên đầu trang

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


def adjacent_values(vals, q1, q3):
    upper_adjacent_value = q3 + (q3 - q1) * 1.5
    upper_adjacent_value = np.clip(upper_adjacent_value, q3, vals[-1])
```

```
    lower_adjacent_value = q1 - (q3 - q1) * 1.5
    lower_adjacent_value = np.clip(lower_adjacent_value, vals[0], q1)
    return lower_adjacent_value, upper_adjacent_value


def set_axis_style(ax, labels):
    ax.get_xaxis().set_tick_params(direction='out')
    ax.xaxis.set_ticks_position('bottom')
    ax.set_xticks(np.arange(1, len(labels) + 1))
    ax.set_xticklabels(labels)
    ax.set_xlim(0.25, len(labels) + 0.75)
    ax.set_xlabel('Sample name')


# create test data
np.random.seed(19680801)
data = [sorted(np.random.normal(0, std, 100)) for std in range(1, 5)]

fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(9, 4), sharey=True)

ax1.set_title('Default violin plot')
ax1.set_ylabel('Observed values')
ax1.violinplot(data)

ax2.set_title('Customized violin plot')
parts = ax2.violinplot(
        data, showmeans=False, showmedians=False,
        showextrema=False)

for pc in parts['bodies']:
    pc.set_facecolor('#D43F3A')
    pc.set_edgecolor('black')
    pc.set_alpha(1)

quartile1, medians, quartile3 = np.percentile(data, [25, 50, 75], axis=1)
whiskers = np.array([
    adjacent_values(sorted_array, q1, q3)
    for sorted_array, q1, q3 in zip(data, quartile1, quartile3)])
whiskersMin, whiskersMax = whiskers[:, 0], whiskers[:, 1]

inds = np.arange(1, len(medians) + 1)
ax2.scatter(inds, medians, marker='o', color='white', s=30, zorder=3)
ax2.vlines(inds, quartile1, quartile3, color='k', linestyle='-', lw=5)
ax2.vlines(inds, whiskersMin, whiskersMax, color='k', linestyle='-', lw=1)

# set style for the axes
```

```
labels = ['A', 'B', 'C', 'D']
for ax in [ax1, ax2]:
    set_axis_style(ax, labels)

plt.subplots_adjust(bottom=0.15, wspace=0.05)
plt.show()
```



BÀI TIẾP THEO: THREE-DIMENSIONAL PLOTTING (BIỂU ĐỒ 3 CHIỀU) >>

Bài 23: Three-dimensional Plotting (Biểu đồ 3 chiều) - Matplotib Cơ Bản
Đăng bởi: Admin | Lượt xem: 3514 | Chuyên mục: AI

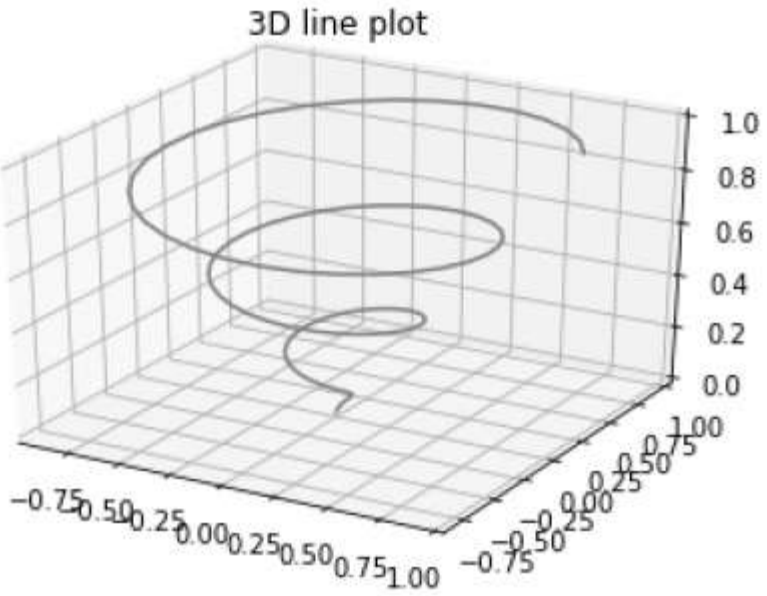
Matplotlib ban đầu chỉ được thiết kế với mục đích vẽ biểu đồ hai chiều, một số tiện ích vẽ biểu đồ ba chiều đã được xây dựng trên màn hình hai chiều của Matplotlib trong các phiên bản sau, để cung cấp một bộ công cụ để trực quan hóa dữ liệu ba chiều. Các plot ba chiều được kích hoạt bằng cách nhập bộ công cụ mplot3d, đi kèm với package Matplotlib.

Các trục ba chiều có thể được tạo bằng cách chuyển từ khóa chiếu = '3d' vào bất kỳ quy trình tạo trục thông thường nào.

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

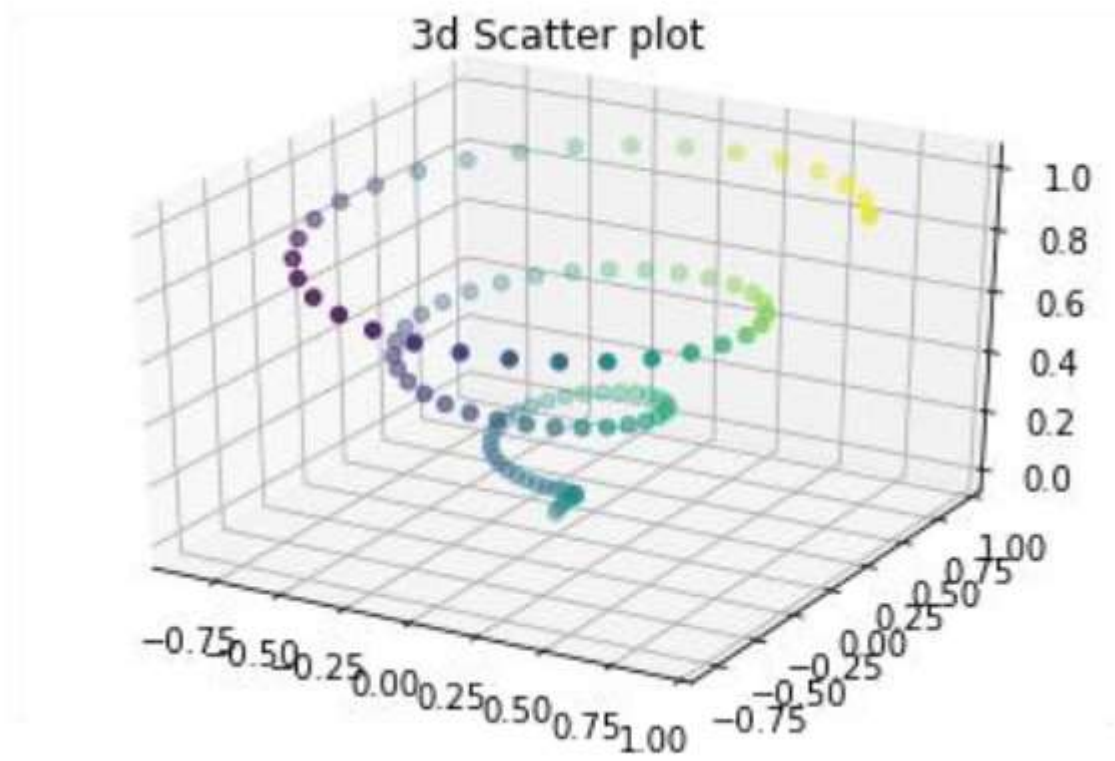
```
import matplotlib.pyplot as plt
fig = plt.figure()
ax = plt.axes(projection='3d')
z = np.linspace(0, 1, 100)
x = z * np.sin(20 * z)
y = z * np.cos(20 * z)
ax.plot3D(x, y, z, 'gray')
ax.set_title('3D line plot')
plt.show()
```

Bây giờ ta có thể vẽ nhiều kiểu biểu đồ ba chiều khác nhau. Biểu đồ ba chiều cơ bản nhất là biểu đồ đường 3D được tạo từ bộ (x, y, z) . Tạo bằng cách sử dụng hàm ax.plot3D.



Biểu đồ phân tán 3D được tạo bằng cách sử dụng hàm 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()
```



BÀI TIẾP THEO: 3D CONTOUR PLOT (BIỂU ĐỒ VIỀN 3D) >>

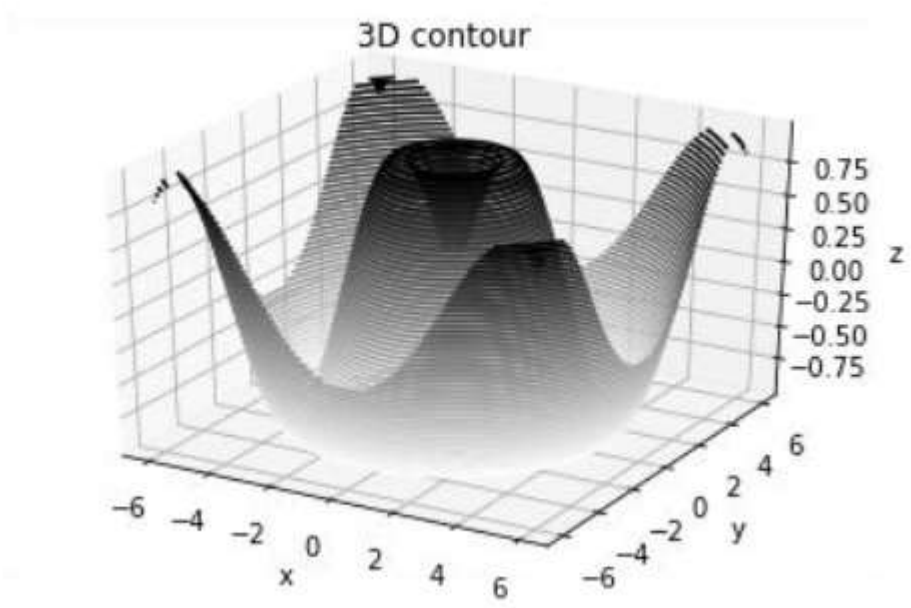
Bài 24: 3D Contour Plot (Biểu đồ viền 3D) - Matplotib Cơ Bản
Đăng bởi: Admin | Lượt xem: 1418 | Chuyên mục: AI

1. Khái niệm :

Hàm ax.contour3D () tạo biểu đồ đường bao ba chiều. Nó yêu cầu tất cả dữ liệu đầu vào phải ở dạng lưới thông thường hai chiều, với dữ liệu Z được đánh giá tại mỗi điểm. Ở đây, ta sẽ trình bày một biểu đồ đường bao ba chiều của một hàm hình sin ba chiều.

```
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()
```

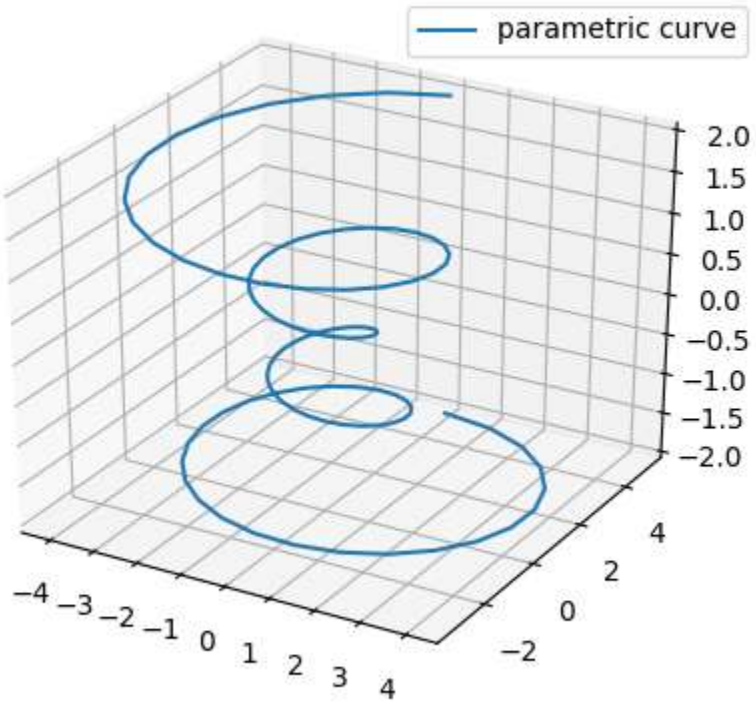


2. Ví dụ :

Ví dụ 1 : Lines plot

```
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import matplotlib.pyplot as plt

mpl.rcParams['legend.fontsize'] = 10

fig = plt.figure()
ax = fig.gca(projection='3d')
theta = np.linspace(-4 * np.pi, 4 * np.pi, 100)
z = np.linspace(-2, 2, 100)
r = z**2 + 1
```

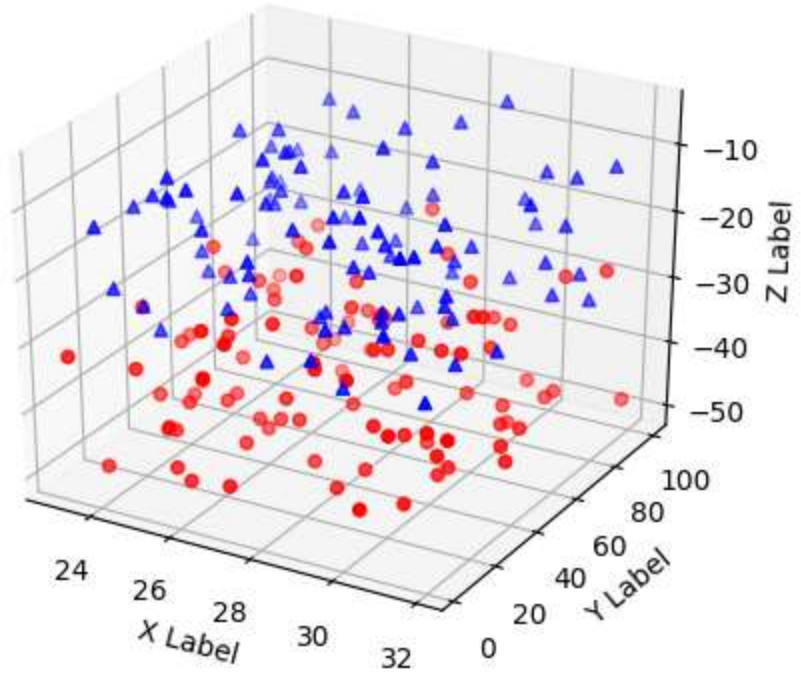
```
x = r * np.sin(theta)
y = r * np.cos(theta)
ax.plot(x, y, z, label='parametric curve')
ax.legend()

plt.show()
```



Ví dụ 2 : Scatter Plot

```
'''
==============
3D scatterplot
==============

Demonstration of a basic scatterplot in 3D.
'''

from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np


def randrange(n, vmin, vmax):
```

```
    '''
    Helper function to make an array of random numbers having shape (n, )
    with each number distributed Uniform(vmin, vmax).
    '''
    return (vmax - vmin)*np.random.rand(n) + vmin

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

n = 100

# For each set of style and range settings, plot n random points in the box
# defined by x in [23, 32], y in [0, 100], z in [zlow, zhigh].
for c, m, zlow, zhigh in [('r', 'o', -50, -25), ('b', '^', -30, -5)]:
    xs = randrange(n, 23, 32)
    ys = randrange(n, 0, 100)
    zs = randrange(n, zlow, zhigh)
    ax.scatter(xs, ys, zs, c=c, marker=m)

ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')

plt.show()
```



BÀI TIẾP THEO: 3D WIREFRAME PLOT >>

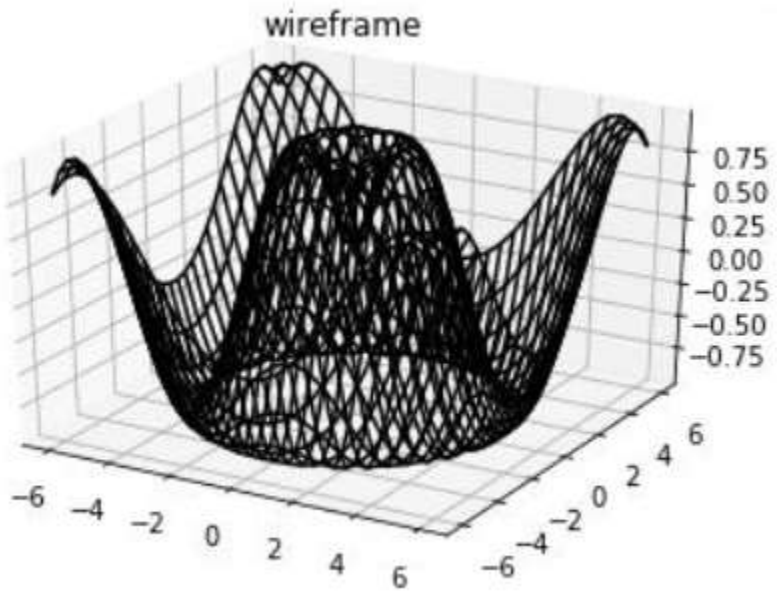
Bài 25: 3D Wireframe plot - Matplotib Cơ Bản
1. Khái niệm :

Biểu đồ Wireframe lấy một mạng lưới các giá trị và chiếu nó lên bề mặt ba chiều được chỉ định và có thể tạo ra các dạng ba chiều thu được khá dễ hình dung. Hàm plot_wireframe () được sử dụng cho mục đích -

```
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()
```

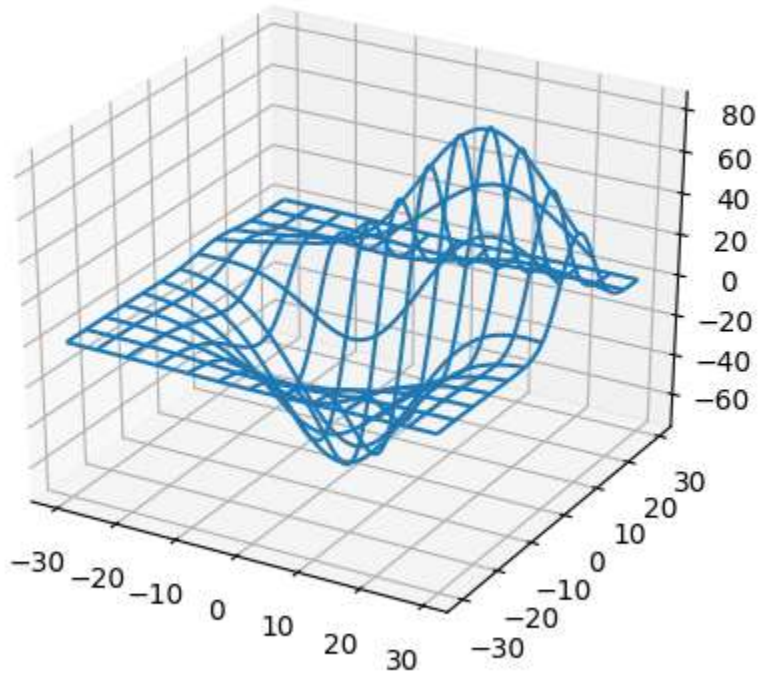
Kết quả như sau :



2. Ví dụ

```
'''
================
3D wireframe plot
================

A very basic demonstration of a wireframe plot.
'''

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


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

# Grab some test data.
X, Y, Z = axes3d.get_test_data(0.05)

# Plot a basic wireframe.
ax.plot_wireframe(X, Y, Z, rstride=10, cstride=10)
```

```
plt.show()
```



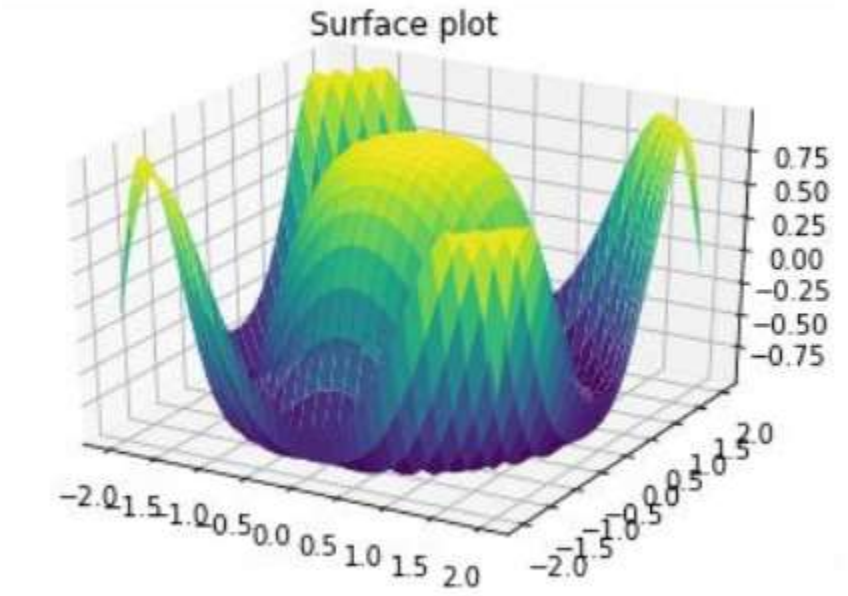
BÀI TIẾP THEO: 3D SURFACE PLOT >>

Bài 26: 3D Surface plot - Matplotib Cơ Bản
1. Khái niệm :

Biểu đồ mặt cho thấy mối quan hệ chức năng giữa một biến phụ thuộc được chỉ định (Y) và hai biến độc lập (X và Z). Biểu đồ là kết hợp với biểu đồ đường viền. Biểu đồ mặt giống như biểu đồ wireframe, nhưng mỗi mặt của wireframey là một đa giác được lấp đầy. Điều này có thể hỗ trợ nhận thức về cấu trúc liên kết của bề mặt đang được hình dung. Hàm plot_surface () x, y và z làm đối số.

```
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')
```

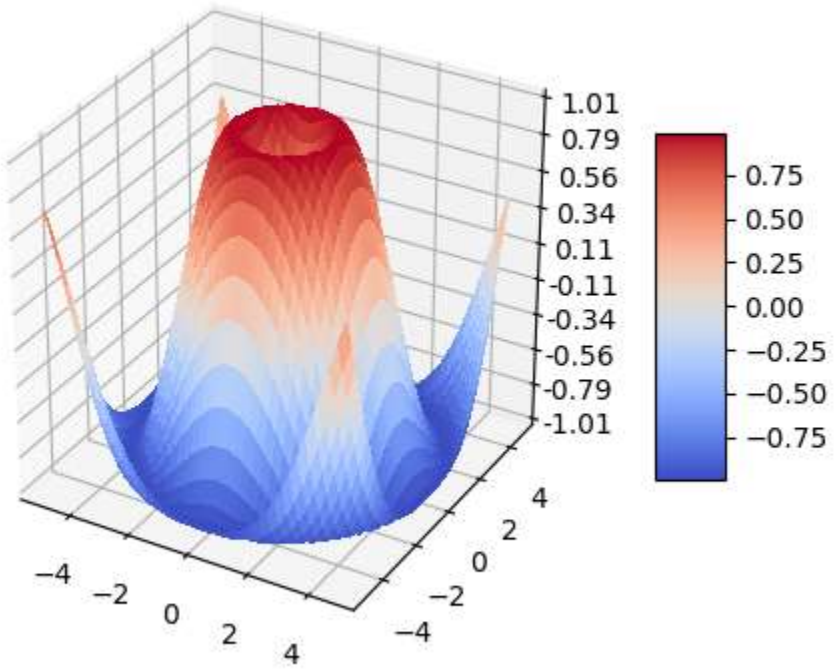
```
ax.plot_surface(x, y, z,cmap='viridis', edgecolor='none')
ax.set_title('Surface plot')
plt.show()
```
Kết quả :



2. Ví dụ :

```
'''
======================
3D surface (color map)
======================

Demonstrates plotting a 3D surface colored with the coolwarm color map.
The surface is made opaque by using antialiased=False.

Also demonstrates using the LinearLocator and custom formatting for the
z axis tick labels.
'''

from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
import numpy as np


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

```
# Make data.
X = np.arange(-5, 5, 0.25)
Y = np.arange(-5, 5, 0.25)
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X**2 + Y**2)
Z = np.sin(R)

# Plot the surface.
surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm,
                       linewidth=0, antialiased=False)

# Customize the z axis.
ax.set_zlim(-1.01, 1.01)
ax.zaxis.set_major_locator(LinearLocator(10))
ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))

# Add a color bar which maps values to colors.
fig.colorbar(surf, shrink=0.5, aspect=5)

plt.show()
```



Bài 27: Làm việc với văn bản - Matplotib Cơ Bản
Đăng bởi: Admin | Lượt xem: 1167 | Chuyên mục: AI

Matplotlib có hỗ trợ văn bản, bao gồm hỗ trợ các biểu thức toán học, hỗ trợ TrueType cho đầu ra raster và vector, văn bản được phân tách theo dòng mới với các phép xoay tùy ý và hỗ trợ unicode. Matplotlib bao gồm matplotlib.font_manager của riêng nó, thực hiện một nền tảng chéo, thuật toán tìm phông chữ tuân thủ W3C.

Người dùng có rất nhiều quyền kiểm soát đối với các thuộc tính văn bản (cỡ chữ, độ đậm của phông chữ, vị trí và màu văn bản, v.v.). Matplotlib thực hiện một số lượng lớn các ký hiệu và lệnh toán học TeX.

Danh sách các lệnh sau được sử dụng để tạo văn bản trong giao diện Pyplot:

figtext	**Thêm văn bản vào một vị trí tùy ý của Figure.**
suptitle	Thêm tiêu đề vào Figure
text	Thêm văn bản tại một vị trí tùy ý của Axes.
annotate	Thêm chú thích, với một mũi tên tùy chọn, tại vị trí tùy ý của Axes.
xlabel	Thêm nhãn vào trục x của Axes.
ylabel	Thêm nhãn vào trục y của Axes.
title	Thêm title vào Axes

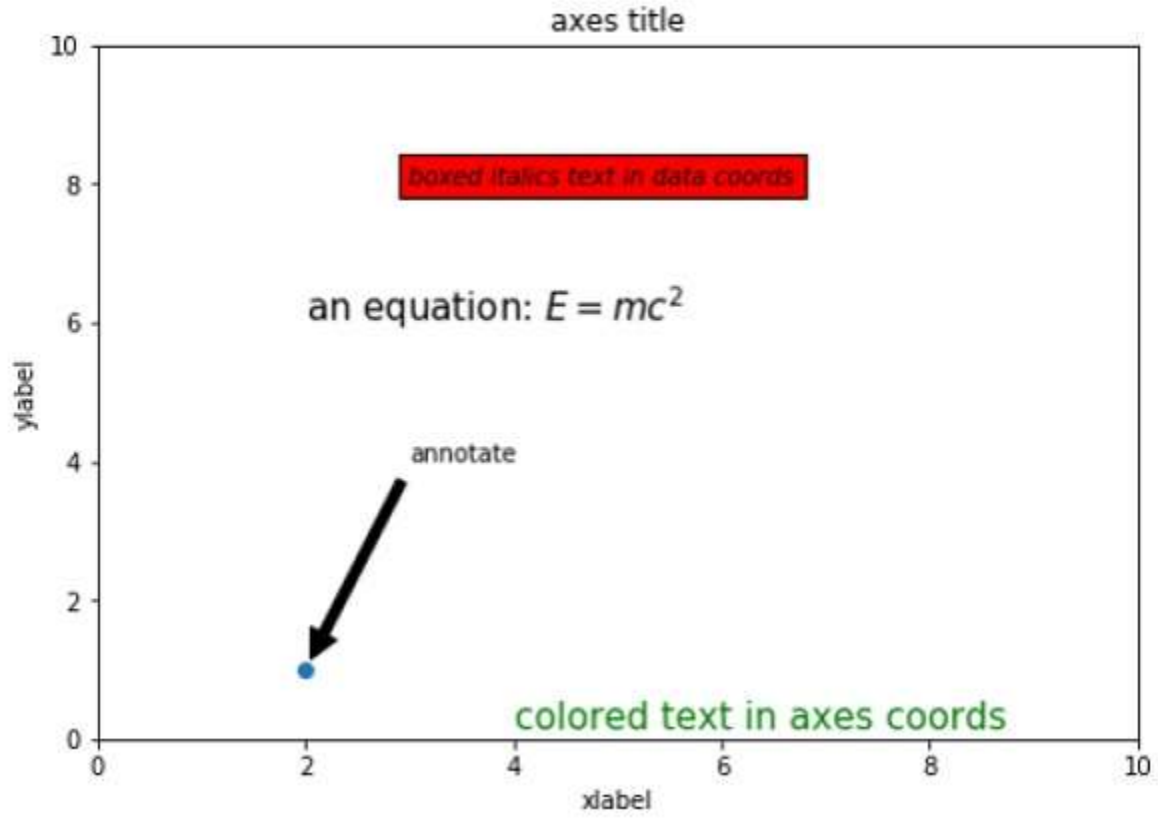
Tất cả các hàm này tạo và trả về một cá thể matplotlib.text.Text ().

Ví dụ :

```
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()
```

Kết quả :



Bài 28: Biểu thức toán học - Matplotib Cơ Bản
Đăng bởi: Admin | Lượt xem: 1396 | Chuyên mục: AI

1. Biểu thức toán học :

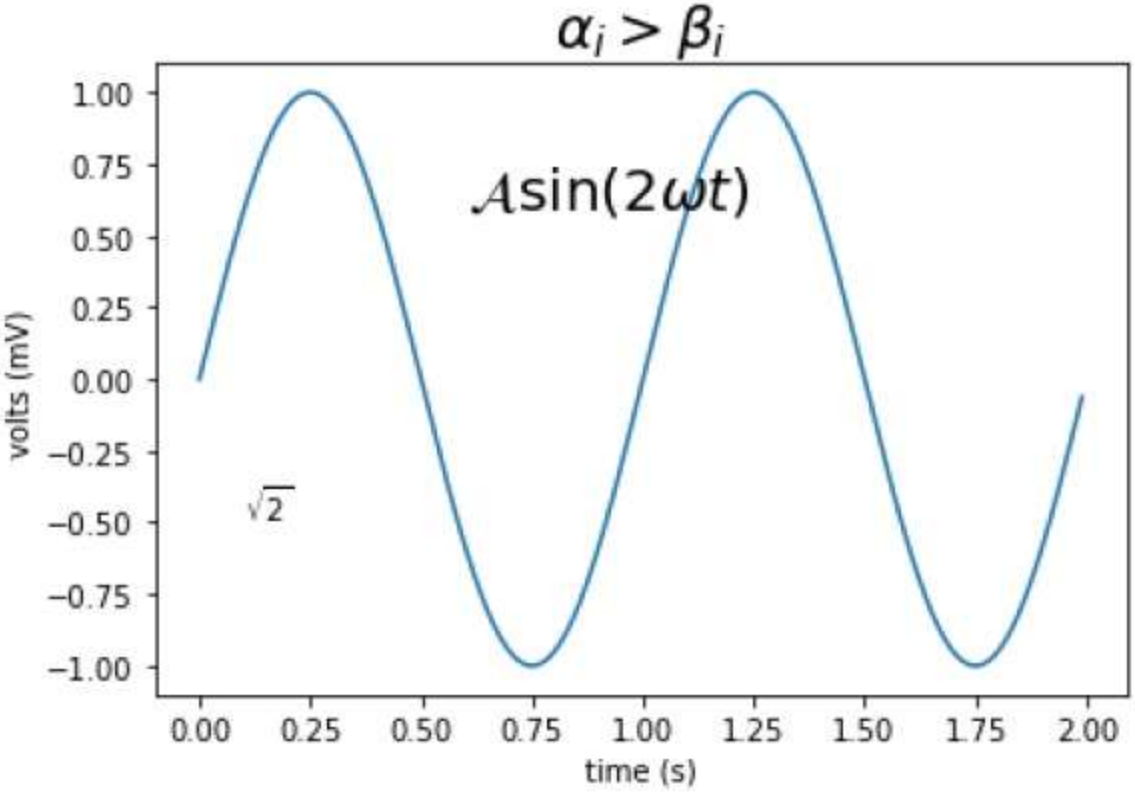
Sử dụng một tập con TeXmarkup trong bất kỳ chuỗi văn bản Matplotlib nào bằng cách đặt nó bên trong một cặp dấu đô la ($).

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

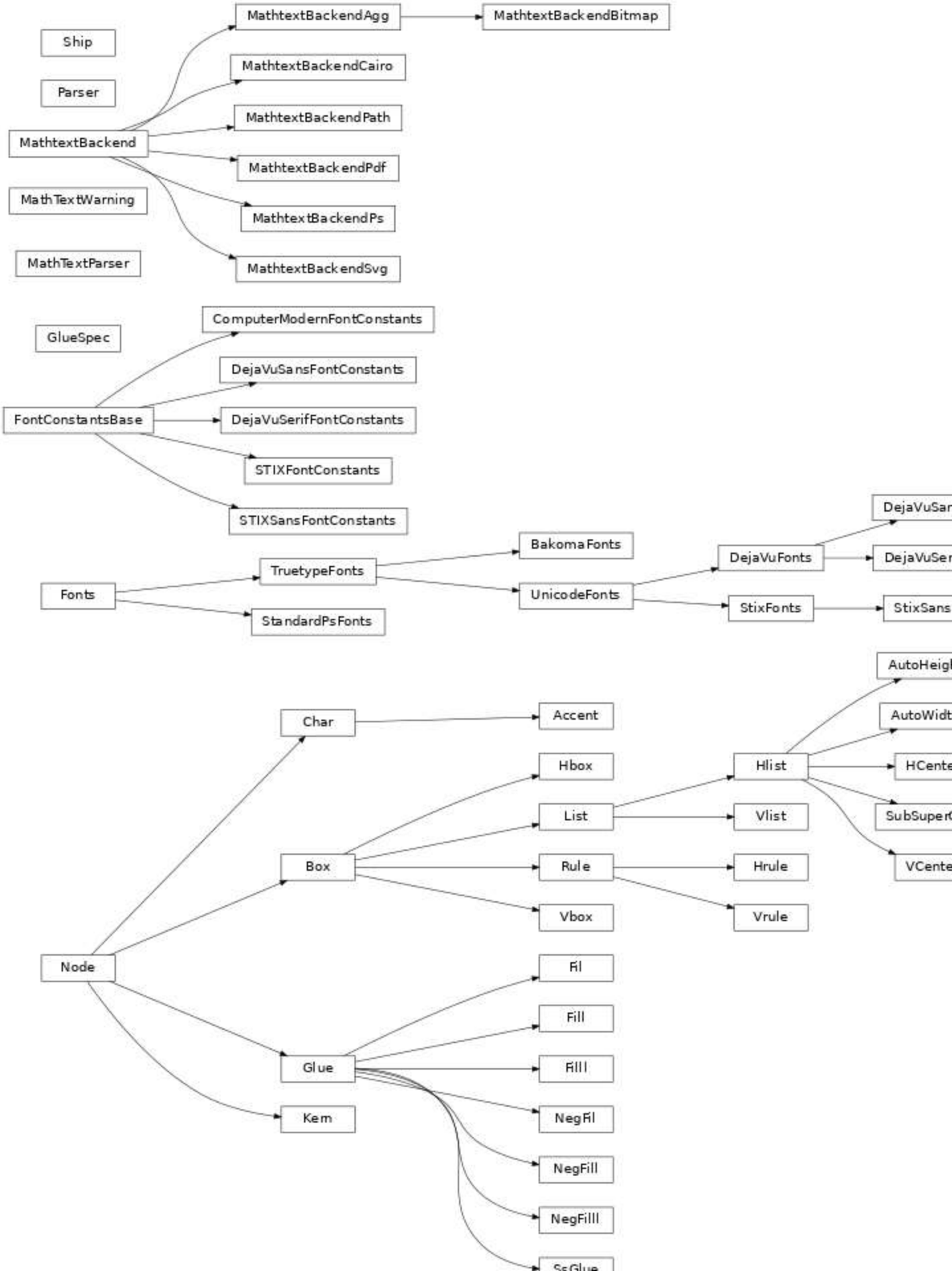
Để tạo chỉ số dưới và chỉ số trên, hãy sử dụng các ký hiệu '_' và '^' -

```
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()
```

Kết quả :


$\alpha_i > \beta_i$
$\mathcal{A}\sin(2\omega t)$
$\sqrt{2}$
volts (mV)
time (s)
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

2. matplotlib.mathtext


MathtextBackendAgg
MathtextBackendBitmap
Ship
MathtextBackendCairo
Parser
MathtextBackendPath
MathtextBackend
MathtextBackendPdf
MathTextWarning
MathtextBackendPs
MathTextParser
MathtextBackendSvg
ComputerModernFontConstants
GlueSpec
DejaVuSansFontConstants
FontConstantsBase
DejaVuSerifFontConstants
STIXFontConstants
STIXSansFontConstants
BakomaFonts
TruetypeFonts
Fonts
UnicodeFonts
DejaVuFonts
StandardPsFonts
StixFonts
Char
Accent
Hbox
List
Vlist
Box
Rule
Hrule
Vbox
Vrule
Node
Fil
Fill
Glue
Filll
Kern
NegFil
NegFill
NegFilll

mathtext là một mô-đun để phân tích cú pháp một tập hợp con trong toán học TeX và vẽ chúng vào phần mềm phụ trợ matplotlib.

Để biết cách sử dụng nó, hãy xem Viết biểu thức toán học. Tài liệu này chủ yếu liên quan đến chi tiết triển khai.

Mô-đun sử dụng pyparsing để phân tích cú pháp biểu thức TeX.

Bản phân phối Bakoma của phông chữ TeX Computer Modern và phông chữ STIX được hỗ trợ. Có hỗ trợ thử nghiệm cho việc sử dụng các phông chữ tùy ý, nhưng kết quả có thể khác nhau nếu không có sự tinh chỉnh và chỉ số phù hợp cho các phông chữ đó.

3. Ví dụ :

```
import matplotlib.pyplot as plt
import subprocess
import sys
import re

# Selection of features following "Writing mathematical expressions" tutorial
mathtext_titles = {
    0: "Header demo",
    1: "Subscripts and superscripts",
    2: "Fractions, binomials and stacked numbers",
    3: "Radicals",
    4: "Fonts",
    5: "Accents",
    6: "Greek, Hebrew",
    7: "Delimiters, functions and Symbols"}
n_lines = len(mathtext_titles)

# Randomly picked examples
mathext_demos = {
    0: r"$W^{3\beta}_{\delta_1 \rho_1 \sigma_2} = "
    r"U^{3\beta}_{\delta_1 \rho_1} + \frac{1}{8 \pi 2} "
    r"\int^{\alpha_2}_{\alpha_2} d \alpha^\prime_2 \left[\frac{ "
    r"U^{2\beta}_{\delta_1 \rho_1} - \alpha^\prime_2U^{1\beta}_"
    r"{\rho_1 \sigma_2} }{U^{0\beta}_{\rho_1 \sigma_2}}\right]$",

    1: r"$\alpha_i > \beta_i,\ "
    r"\alpha_{i+1}^j = {\rm sin}(2\pi f_j t_i) e^{-5 t_i/\tau},\ "
    r"\ldots$",

    2: r"$\frac{3}{4},\ \binom{3}{4},\ \genfrac{}{}{0}{}{3}{4},\ "
    r"\left(\frac{5 - \frac{1}{x}}{4}\right),\ \ldots$",

    3: r"$\sqrt{2},\ \sqrt[3]{x},\ \ldots$",
```

```
    4: r"$\mathrm{Roman}\ , \ \mathit{Italic}\ , \ \mathtt{Typewriter} \ "
    r"\mathrm{or}\ \mathcal{CALLIGRAPHY}$",

    5: r"$\acute a,\ \bar a,\ \breve a,\ \dot a,\ \ddot a, \ \grave a, \ "
    r"\hat a,\ \tilde a,\ \vec a,\ \widehat{xyz},\ \widetilde{xyz},\ "
    r"\ldots$",

    6: r"$\alpha,\ \beta,\ \chi,\ \delta,\ \lambda,\ \mu,\ "
    r"\Delta,\ \Gamma,\ \Omega,\ \Phi,\ \Pi,\ \Upsilon,\ \nabla,\ "
    r"\aleph,\ \beth,\ \daleth,\ \gimel,\ \ldots$",

    7: r"$\coprod,\ \int,\ \oint,\ \prod,\ \sum,\ "
    r"\log,\ \sin,\ \approx,\ \oplus,\ \star,\ \varpropto,\ "
    r"\infty,\ \partial,\ \Re,\ \leftrightsquigarrow, \ \ldots$"}


def doall():
    # Colors used in mpl online documentation.
    mpl_blue_rvb = (191. / 255., 209. / 256., 212. / 255.)
    mpl_orange_rvb = (202. / 255., 121. / 256., 0. / 255.)
    mpl_grey_rvb = (51. / 255., 51. / 255., 51. / 255.)

    # Creating figure and axis.
    plt.figure(figsize=(6, 7))
    plt.axes([0.01, 0.01, 0.98, 0.90], facecolor="white", frameon=True)
    plt.gca().set_xlim(0., 1.)
    plt.gca().set_ylim(0., 1.)
    plt.gca().set_title("Matplotlib's math rendering engine",
                        color=mpl_grey_rvb, fontsize=14, weight='bold')
    plt.gca().set_xticklabels("", visible=False)
    plt.gca().set_yticklabels("", visible=False)

    # Gap between lines in axes coords
    line_axesfrac = (1. / (n_lines))

    # Plotting header demonstration formula
    full_demo = mathext_demos[0]
    plt.annotate(full_demo,
                 xy=(0.5, 1. - 0.59 * line_axesfrac),
                 color=mpl_orange_rvb, ha='center', fontsize=20)

    # Plotting features demonstration formulae
    for i_line in range(1, n_lines):
        baseline = 1 - (i_line) * line_axesfrac
        baseline_next = baseline - line_axesfrac
        title = mathtext_titles[i_line] + ":"
```

```
    fill_color = ['white', mpl_blue_rvb][i_line % 2]
    plt.fill_between([0., 1.], [baseline, baseline],
                     [baseline_next, baseline_next],
                     color=fill_color, alpha=0.5)
    plt.annotate(title,
                 xy=(0.07, baseline - 0.3 * line_axesfrac),
                 color=mpl_grey_rvb, weight='bold')
    demo = mathext_demos[i_line]
```

Bài 29: Làm việc với ảnh - Matplotib Cơ Bản
1. Tổng quan

Mô-đun hình ảnh trong package Matplotlib cung cấp các chức năng cần thiết để tải, thay đổi tỷ lệ và hiển thị hình ảnh.

Thư viện Pillow hỗ trợ tải dữ liệu hình ảnh. Về cơ bản, Matplotlib chỉ hỗ trợ hình ảnh PNG. Các lệnh hiển thị bên dưới sẽ trở lại trên Pillow nếu quá trình đọc gốc không thành công.

Hình ảnh được sử dụng trong ví dụ này là tệp PNG, nhưng hãy ghi nhớ yêu cầu về Pillow đó cho dữ liệu của riêng bạn. Hàm imread () được sử dụng để đọc dữ liệu hình ảnh trong một đối tượng ndarray của float32 dtype.

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
img = mpimg.imread('mtplogo.png')
```
Giả sử rằng hình ảnh sau có tên là mtplogo.png có trong thư mục làm việc hiện tại

Bất kỳ mảng nào chứa dữ liệu hình ảnh đều có thể được lưu vào tệp đĩa bằng cách thực thi hàm imsave (). Tại đây, một phiên bản được lật theo chiều dọc của tệp png gốc được lưu bằng cách đặt tham số gốc là 'lower'.

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

Để vẽ hình ảnh trên trình xem Matplotlib, sử dụng hàm imshow ().

2. Ví dụ :

Nếu sử dụng IPython Notebook, bạn cần thêm dòng sau :

```
In [1]: %matplotlib inline
```

Điều này sẽ bật tính năng vẽ nội tuyến, nơi đồ họa plot sẽ xuất hiện trong notebook. Điều này có ý nghĩa quan trọng đối với khả năng tương tác. Đối với biểu đồ nội tuyến, các lệnh trong ô bên dưới ô xuất ra một biểu đồ sẽ không ảnh hưởng đến biểu đồ. Ví dụ: không thể thay đổi bản đồ màu từ các ô bên dưới ô tạo ra một biểu đồ. Tuy nhiên, đối với các phần mềm phụ trợ khác, chẳng hạn như Qt5, mở một cửa sổ riêng biệt, các ô bên dưới các plot sẽ thay đổi - nó là một đối tượng trực tiếp trong bộ nhớ.

Hướng dẫn này sẽ sử dụng giao diện vẽ đồ thị kiểu mệnh lệnh của matplotlib, pyplot. Giao diện này duy trì trạng thái chung và rất hữu ích để nhanh chóng và dễ dàng thử nghiệm với các cài đặt plot khác nhau. Lựa chọn thay thế là giao diện hướng đối tượng, giao diện này cũng rất mạnh và thường phù hợp hơn cho việc phát triển ứng dụng lớn. Nếu bạn muốn tìm hiểu về giao diện hướng đối tượng(IDE). Bây giờ, ta cùng xem qua các ví dụ :

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

Thư viện Pillow hỗ trợ tải dữ liệu hình ảnh. Về cơ bản, Matplotlib chỉ hỗ trợ hình ảnh PNG. Các lệnh hiển thị bên dưới sẽ trở lại trên Pillow nếu quá trình đọc gốc không thành công.

Đó là hình ảnh PNG RGB 24 bit (8 bit cho mỗi R, G, B). Tùy thuộc vào nơi bạn lấy dữ liệu của mình, các loại hình ảnh khác mà bạn rất có thể gặp phải là hình ảnh RGBA, cho phép tạo độ trong suốt hoặc hình ảnh thang độ xám (độ sáng) đơn kênh. Bạn có thể nhấp chuột phải vào nó và chọn "Lưu hình ảnh thành" để tải xuống máy tính

```
img = mpimg.imread('../../doc/_static/stinkbug.png')
print(img)
```

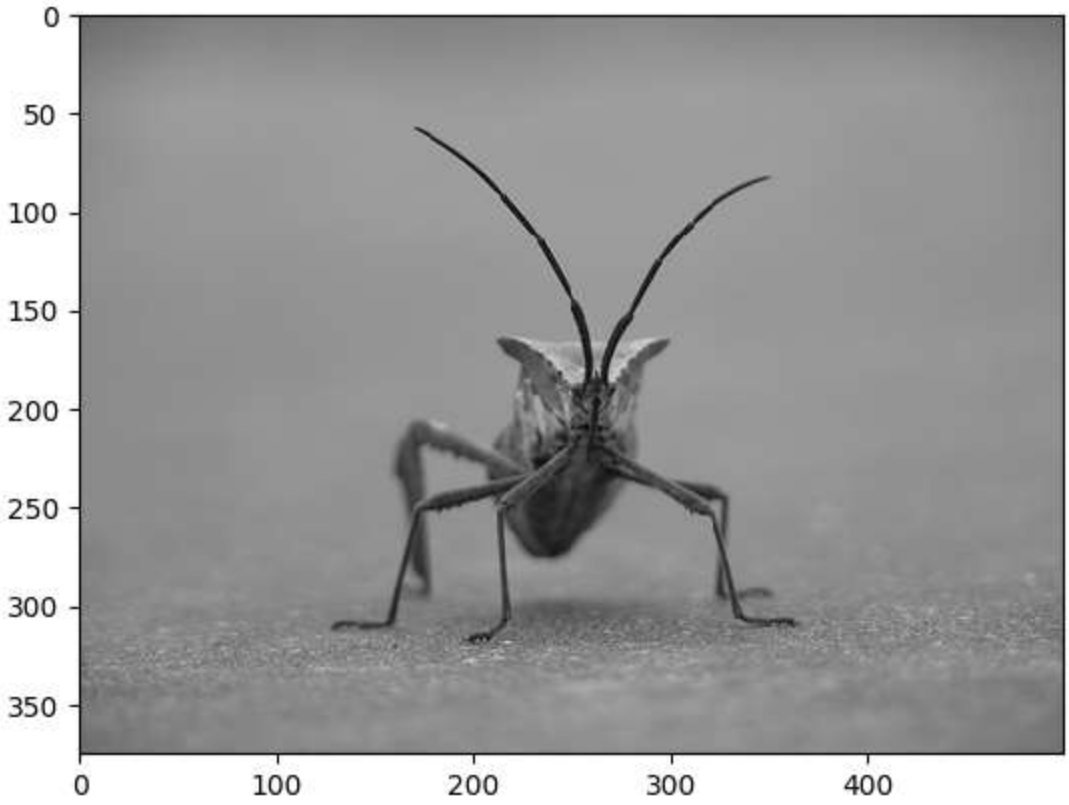
Kết quả như sau :

```
[[[0.40784314 0.40784314 0.40784314]
  [0.40784314 0.40784314 0.40784314]
  [0.40784314 0.40784314 0.40784314]
  ...
  [0.42745098 0.42745098 0.42745098]
  [0.42745098 0.42745098 0.42745098]
  [0.42745098 0.42745098 0.42745098]]

 [[0.4117647  0.4117647  0.4117647 ]
  [0.4117647  0.4117647  0.4117647 ]
  [0.4117647  0.4117647  0.4117647 ]
  ...
  [0.42745098 0.42745098 0.42745098]
  [0.42745098 0.42745098 0.42745098]
```

```
 [0.42745098 0.42745098 0.42745098]]

[[0.41960785 0.41960785 0.41960785]
 [0.41568628 0.41568628 0.41568628]
 [0.41568628 0.41568628 0.41568628]
 ...
 [0.43137255 0.43137255 0.43137255]
 [0.43137255 0.43137255 0.43137255]
 [0.43137255 0.43137255 0.43137255]]

...

[[0.4392157  0.4392157  0.4392157 ]
 [0.43529412 0.43529412 0.43529412]
 [0.43137255 0.43137255 0.43137255]
 ...
 [0.45490196 0.45490196 0.45490196]
 [0.4509804  0.4509804  0.4509804 ]
 [0.4509804  0.4509804  0.4509804 ]]

[[0.44313726 0.44313726 0.44313726]
 [0.44313726 0.44313726 0.44313726]
 [0.4392157  0.4392157  0.4392157 ]
 ...
 [0.4509804  0.4509804  0.4509804 ]
 [0.44705883 0.44705883 0.44705883]
 [0.44705883 0.44705883 0.44705883]]

[[0.44313726 0.44313726 0.44313726]
 [0.4509804  0.4509804  0.4509804 ]
 [0.4509804  0.4509804  0.4509804 ]
 ...
 [0.44705883 0.44705883 0.44705883]
 [0.44705883 0.44705883 0.44705883]
 [0.44313726 0.44313726 0.44313726]]]
```

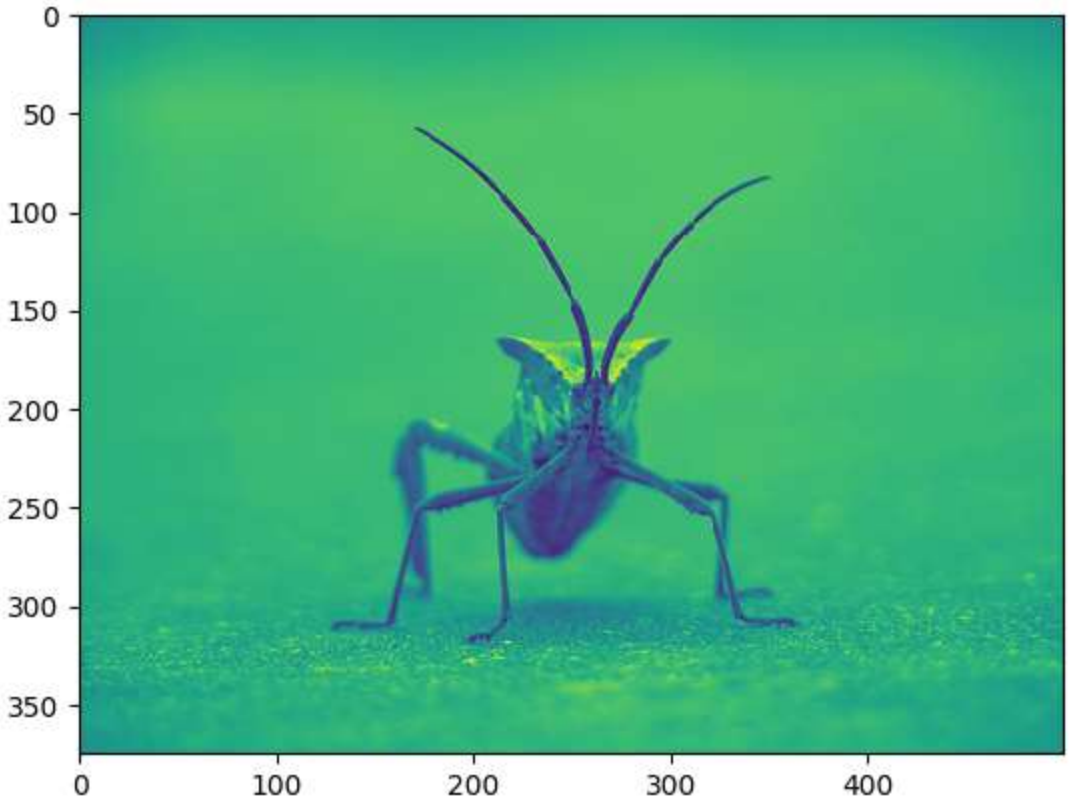
Show ảnh :

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



Áp dụng lược đồ giả màu cho các biểu đồ hình ảnh

```
lum_img = img[:, :, 0]

# This is array slicing.  You can read more in the `Numpy tutorial
# <https://docs.scipy.org/doc/numpy/user/quickstart.html>`_.

plt.imshow(lum_img)
```

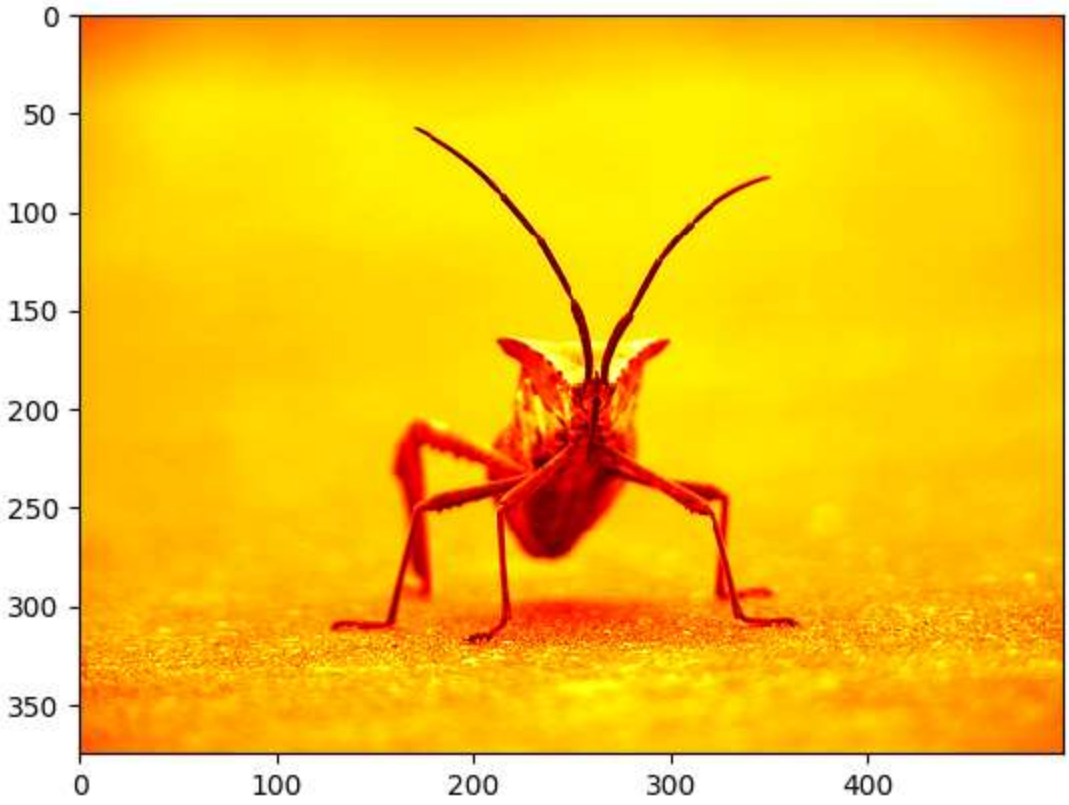


Kết quả :

```
<matplotlib.image.AxesImage object at 0x7fb110cd0ba8>
```

Bây giờ, với hình ảnh có độ sáng (2D, không màu), bản đồ màu mặc định (hay còn gọi là bảng tra cứu, LUT), được áp dụng. Mặc định được gọi là viridis. Có rất nhiều loại khác để lựa chọn.

```
plt.imshow(lum_img, cmap="hot")
```

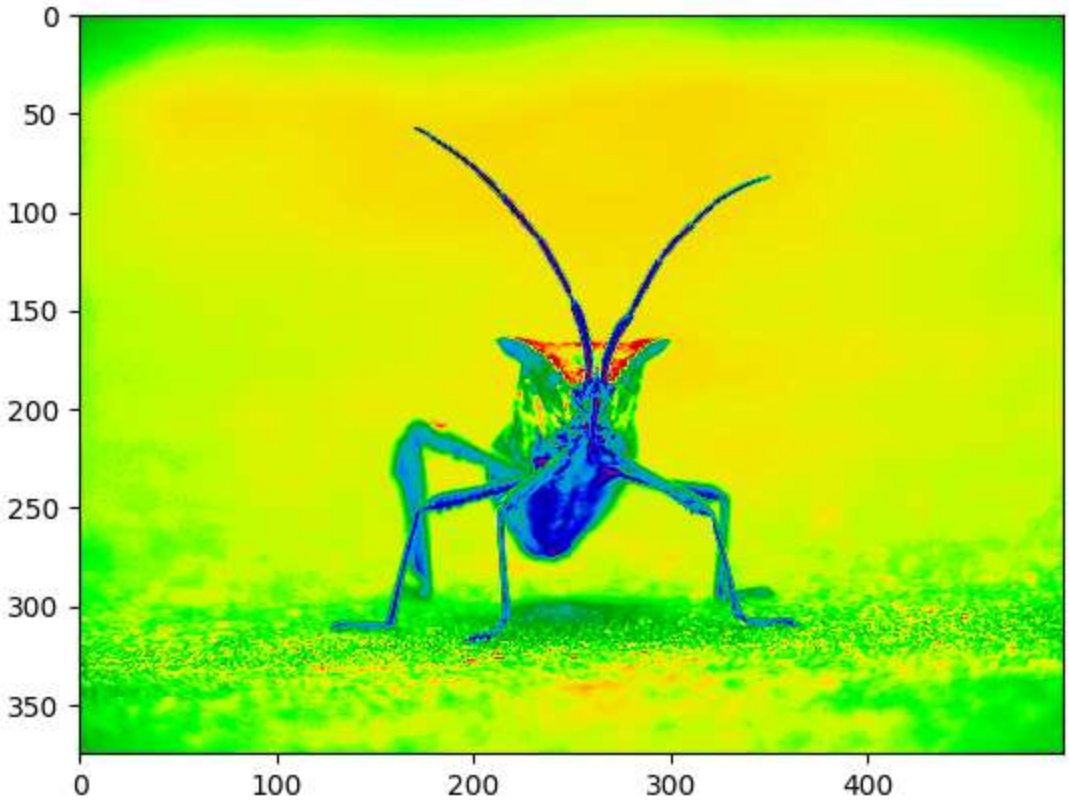


Kết quả :

```
<matplotlib.image.AxesImage object at 0x7fb111323390>
```

Lưu ý rằng bạn cũng có thể thay đổi bản đồ màu trên các plot hiện có bằng cách sử dụng phương thức set_cmap ():

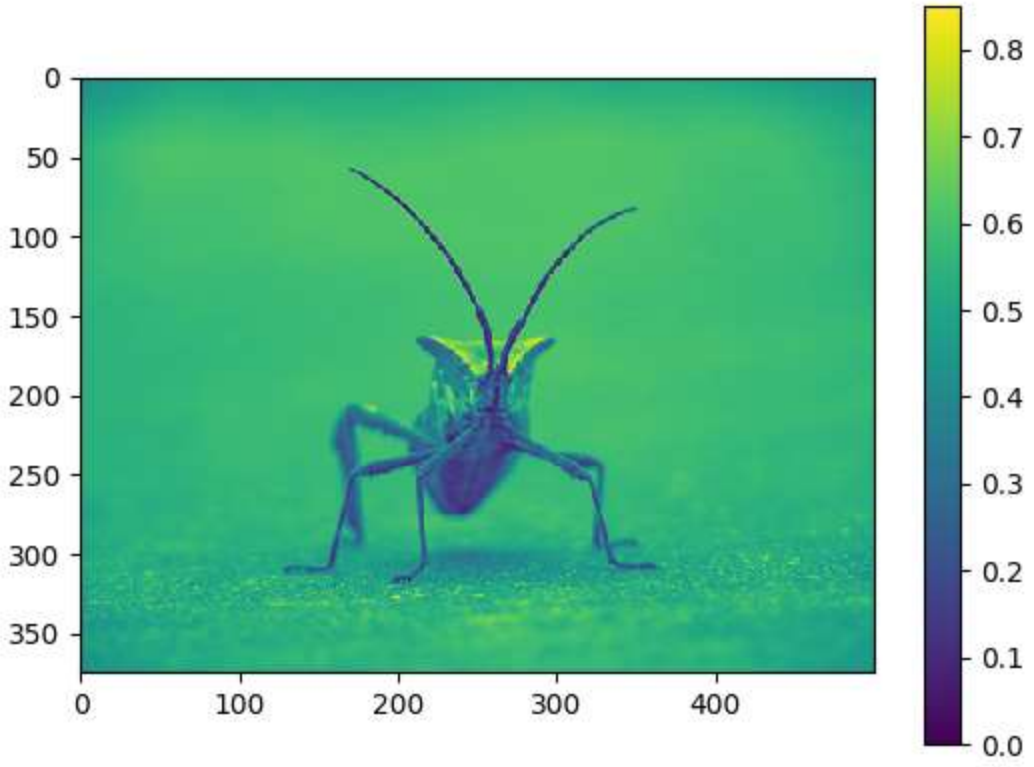
```
imgplot = plt.imshow(lum_img)
imgplot.set_cmap('nipy_spectral')
```



Tham chiếu thang màu :

Sẽ rất hữu ích nếu bạn có ý tưởng về giá trị của màu sắc. Ta có thể làm điều đó bằng cách thêm các thanh màu.

```
imgplot = plt.imshow(lum_img)
plt.colorbar()
```



BÀI TIẾP THEO: TRANSFORMS (BIẾN ĐỔI TRỤC) >>

Bài 30: Transforms (Biến đổi trục) - Matplotib Cơ Bản
Đăng bởi: Admin | Lượt xem: 1178 | Chuyên mục: AI

Package matplotlib được xây dựng dựa trên khung chuyển đổi để dễ dàng di chuyển giữa các hệ tọa độ. Bốn hệ tọa độ có thể được sử dụng. Các hệ thống được mô tả ngắn gọn trong bảng dưới đây:

Coordinate	Transformation Object	Miêu tả

Data	ax.transData	Hệ tọa độ dữ liệu land của người sử dụng được điều khiển bởi xlim và yli
Axes	ax.transAxes	Hệ trục tọa độ. (0,0) ở dưới cùng bên trái và (1,1) ở trên cùng bên phải củ
Figure	fig.transFigure	Hệ tọa độ của Hình. (0,0) ở dưới cùng bên trái và (1,1) ở trên cùng bên ph
display	None	Đây là hệ tọa độ pixel của màn hình. (0,0) là góc dưới bên trái và (chiều r cao) là góc trên bên phải của màn hình tính bằng pixel.Ngoài ra, (matplotlib.transforms.IdentityTransform ()) có thể được sử dụng thay vì

Xem xét ví dụ sau:

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

Text được đặt ở vị trí của một điểm dữ liệu (x, y). Vì vậy, ta sẽ nói về "data coords".

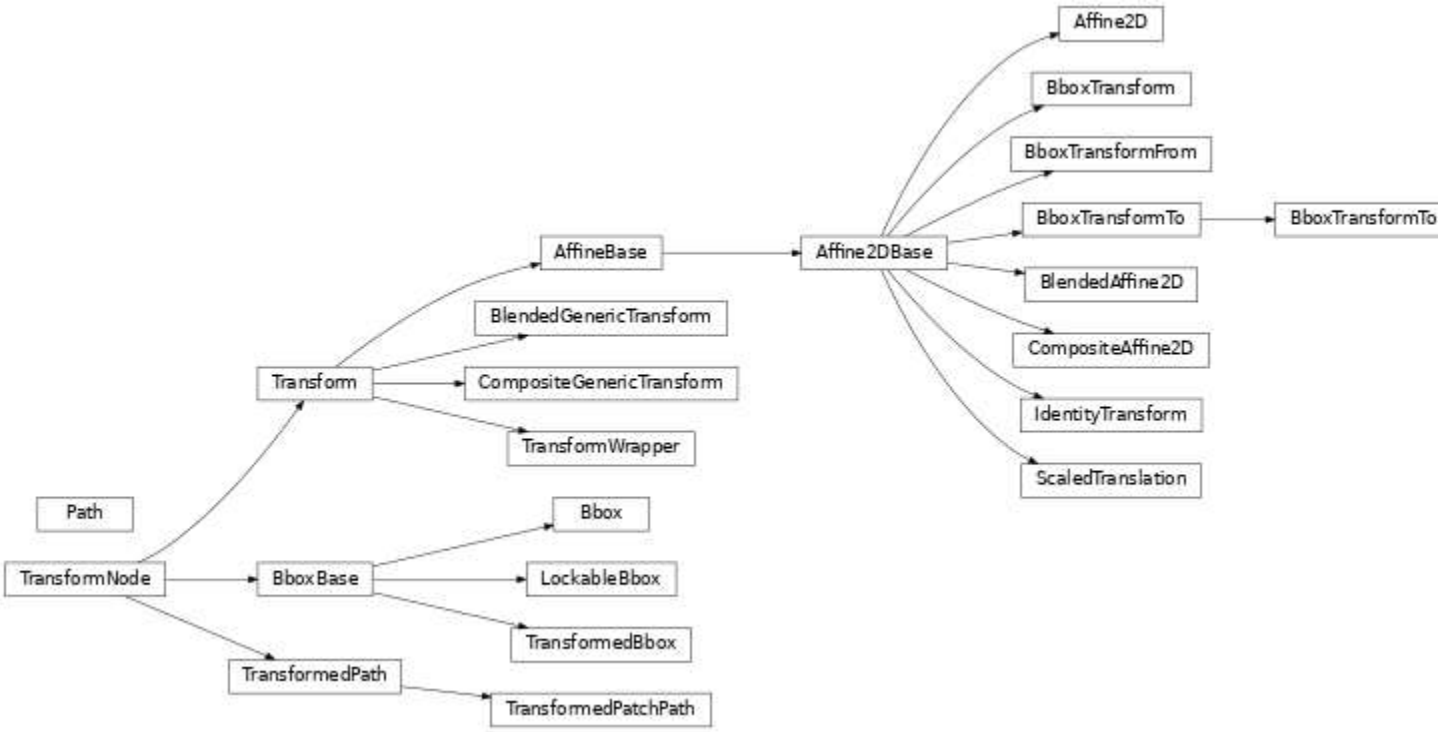
Sử dụng các đối tượng biến đổi khác, vị trí có thể được kiểm soát. Ví dụ, nếu bài kiểm tra trên được đặt ở tâm của hệ tọa độ trục:

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

Các phép biến đổi này có thể được sử dụng cho bất kỳ loại đối tượng Matplotlib nào. Biến đổi mặc định cho ax.text là ax.transData và biến đổi mặc định cho fig.text là fig.transFigure.

Hệ tọa độ trục cực kỳ hữu ích khi đặt văn bản theo trục của bạn. Bạn có thể thường muốn một text bubble ở một vị trí cố định; ví dụ: ở phía trên bên trái của ngăn trục và giữ vị trí đó cố định khi bạn xoay hoặc thu phóng.

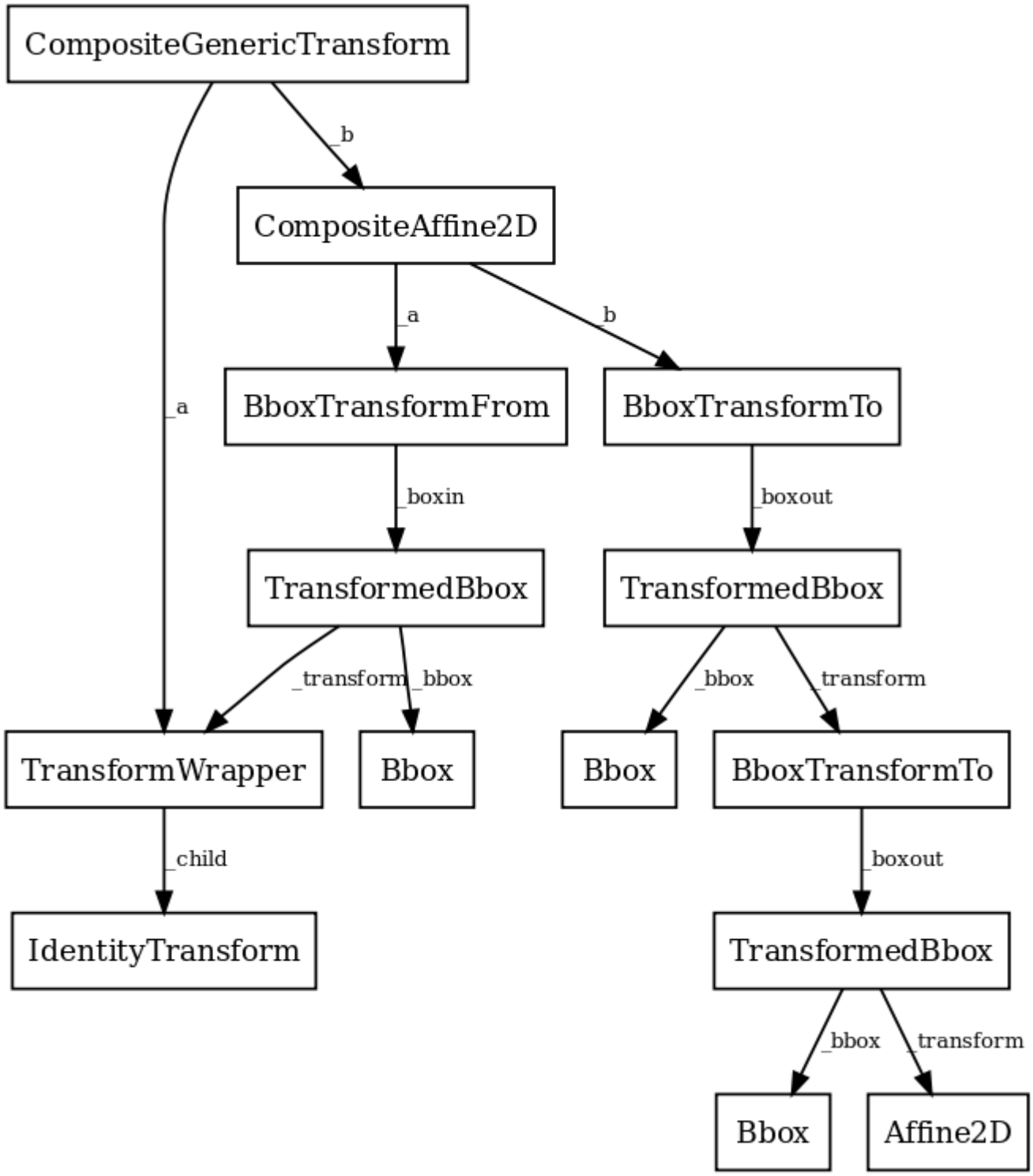
matplotlib.transforms :



matplotlib bao gồm một khuôn khổ cho các phép biến đổi hình học tùy ý được sử dụng để xác định vị trí cuối cùng của tất cả các phần tử được vẽ trên canvas.

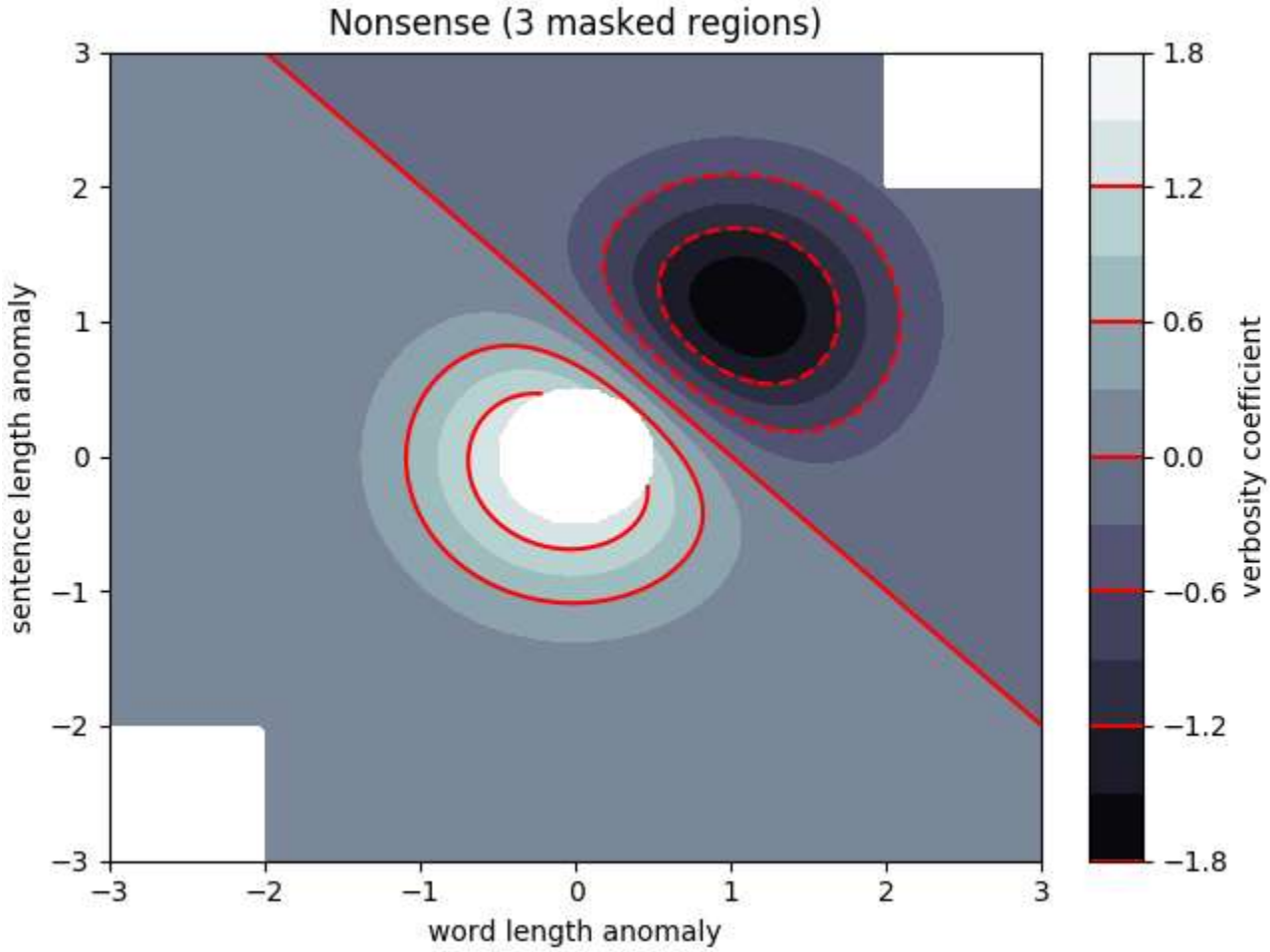
Các biến đổi được tạo thành các cây của đối tượng TransformNode mà giá trị thực của chúng phụ thuộc vào con. Khi nội dung của con thay đổi, cha mẹ của chúng sẽ tự động mất hiệu lực. Lần tiếp theo khi truy cập vào một biến đổi không hợp lệ, nó sẽ được tính toán lại để phản ánh những thay đổi đó. Cách tiếp cận bộ nhớ đệm / vô hiệu hóa này ngăn chặn việc tính toán lại các biến đổi không cần thiết và góp phần vào hiệu suất tương tác tốt hơn.

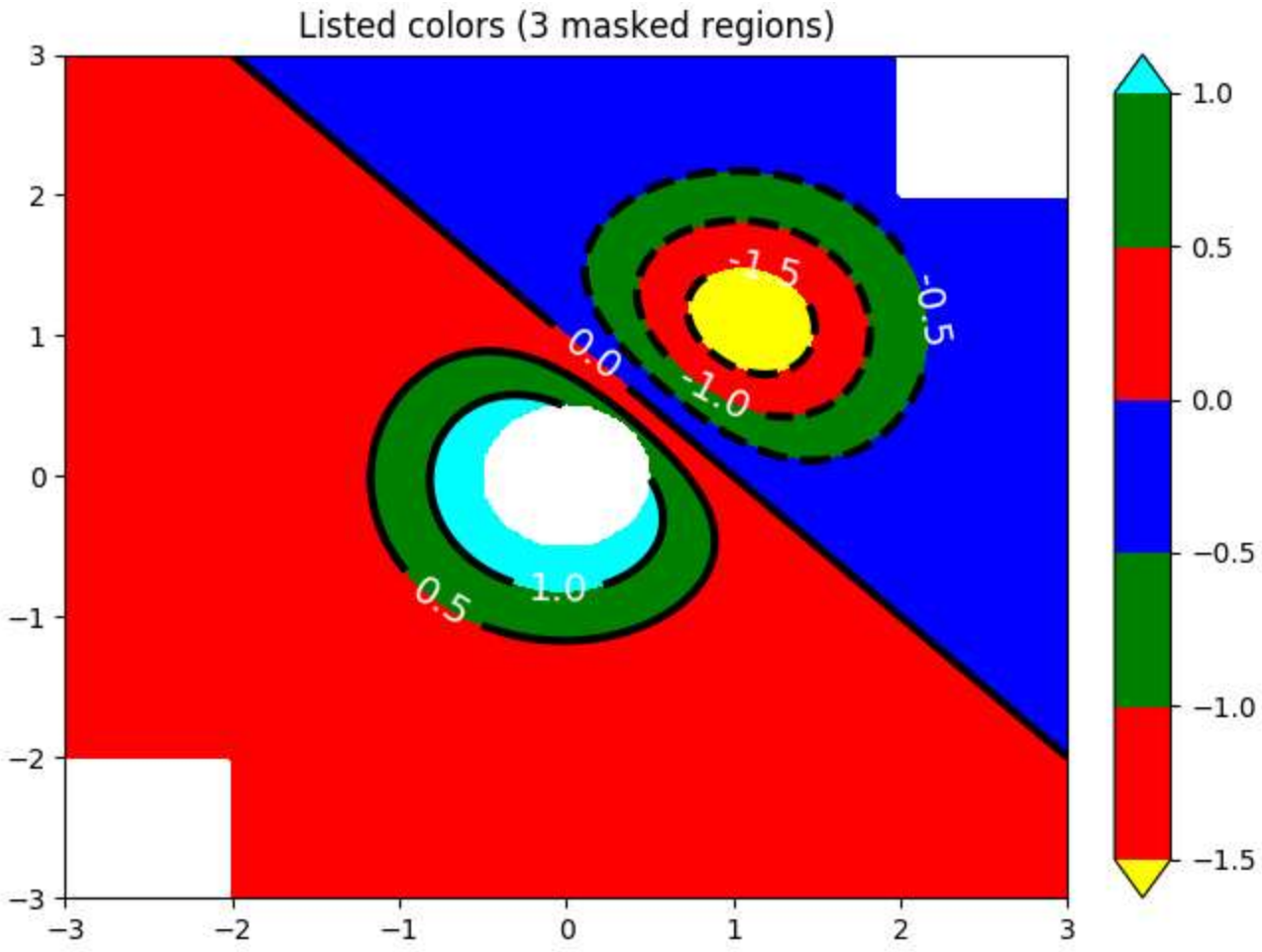
Ví dụ: đây là biểu đồ của cây biến đổi được sử dụng để vẽ dữ liệu vào biểu đồ:

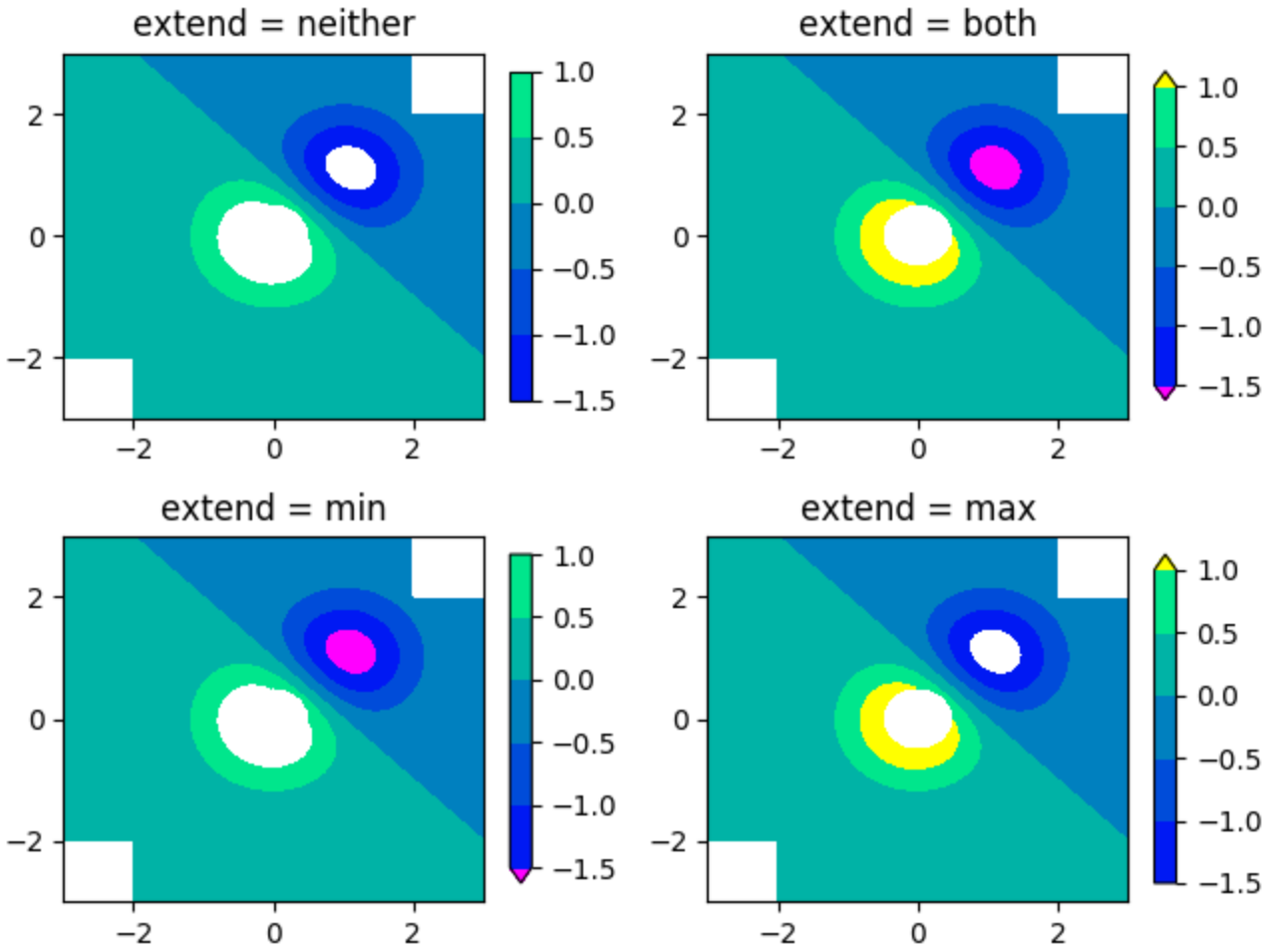


Frame có thể được sử dụng cho cả các phép biến đổi affine và không affine. Tuy nhiên, đối với tốc độ, tôi muốn sử dụng các trình kết xuất phụ trợ để thực hiện các phép biến đổi affine bất cứ khi nào có thể. Do đó, có thể thực hiện chỉ phần affine hoặc phần không affine của một phép biến đổi trên một tập dữ liệu. Affine luôn được giả định là xảy ra sau affine không phải. Đối với bất kỳ biến đổi nào:

full transform == non-affine part + affine part




Listed colors (3 masked regions)
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
-1.5
-0.5
0.0
-1.0
0.5
1.0
1.0
0.5
0.0
−0.5
−1.0
−1.5



Hướng dẫn xây dựng và phân tích biểu đồ Pareto

Sau khi đã hiểu rõ tường tận "**Biểu đồ Pareto là gì**?" và lợi ích, ý nghĩa của biểu đồ này, chúng ta sẽ tìm hiểu thêm về *cách vẽ biểu đồ Pareto*.

Để biểu đồ Pareto có thể phản ánh các dữ liệu một cách chân thực nhất thì quá trình vẽ biểu đồ cần phải được thực hiện cẩn thận. Sau đây là *ví dụ vẽ biểu đồ Pareto* với các hình ảnh minh họa chi tiết:

4.1 Cách vẽ biểu đồ Pareto

Để biểu đồ Pareto có thể phản ánh các dữ liệu một cách chân thực nhất thì quá trình vẽ biểu đồ cần phải được thực hiện cẩn thận. Sau đây là các bước vẽ một biểu đồ Pareto với các hình ảnh ví dụ minh họa.

Bước 1: Tạo một bảng số liệu gồm các vấn đề hiện có với số lần xuất hiện tương ứng:

Bảng số liệu gồm các vấn đề với số lần xuất hiện tương ứng

Bước 2: Sắp xếp các vấn đề theo thứ tự giảm dần dựa vào số lần vấn đề xuất hiện

Sắp xếp các vấn đề theo thứ tự giảm dần.

Bước 3: Tính giá trị tỷ lệ tích lũy cho mỗi vấn đề

Công thức tính giá trị tích lũy là:

Tích lũy vấn đề x = Tích lũy vấn đề x-1 + Vấn đề x

Khi áp dụng vào bảng số liệu ta sẽ có:

- Tích lũy vấn đề 2 = Vấn đề 2 = 2607
- Tích lũy vấn đề 1 = Tích lũy vấn đề 2 + Vấn đề 1 = 2607 + 1003 = 3610

Lần lượt kết quả cuối cùng sẽ là:

Tính giá trị tích lũy của từng vấn đề

Bước 4: Tính phần trăm tích lũy cho mỗi vấn đề

Vì biểu đồ Pareto sử dụng số liệu phần trăm để phản ánh dữ liệu thu thập được nên bạn cần phải tính phần trăm tích lũy cho mỗi vấn đề theo công thức sau:

% Vấn đề 1 = (Tích lũy 1/Tổng số tích lũy) x 100%

Ví dụ:

% Vấn đề 1 = (3610/4802) x 100% = 75%

Sau khi tính số liệu phần trăm ta sẽ có kết quả như sau:

Tính phần trăm tích lũy của từng vấn đề

Bước 5: Tiến hành vẽ biểu đồ Pareto

Cách để tạo biểu đồ Pareto sẽ được tiến hành theo các bước sau

- Chọn phần dữ liệu để vẽ đồ thị là các cột Vấn đề, Số lần và %Tích lũy
- Chọn Insert (chèn) → Charts (đồ thị) → Stacked column (Dạng đồ thị cột xếp chồng 2D)

Trước tiên phải tạo biểu đồ cột với dạng đồ thị cột xếp chồng.

Sau khi chọn sẽ hiển thị biểu đồ như sau

Biểu đồ Pareto dạng cột chồng.

Tiếp theo thực hiện chuyển cột tích lũy thành dạng đồ thị tuyến tính bằng cách bấm click chọn biểu đồ và bấm chuột phải, sau đó chọn Change Chart Type, chọn mục combo và đổi cột tích lũy thành dạng đồ thị Line (đồ thị tuyến tính).

Thực hiện thay đổi cột %Tích lũy thành dạng đồ thị tuyến tính (Line).

Sau khi đổi, biểu đồ của chúng ta sẽ có dạng như sau

Biểu đồ gồm đồ thị dạng cột (Column) và đồ thị tuyến tính (Line)

Tiếp theo tách thang tỉ lệ của mục %Tích lũy sang trục bên phải bằng cách Click chọn đồ thị Line → Bấm chuột phải → chọn Format Data Series → Secondary Axis

Tách thang tỉ lệ cho đồ thị %Tích lũy

Sau khi thực hiện thao tác biểu đồ của chúng ta sẽ có dạng như sau.

Kết quả sau khi thực hiện thao tác.

Bước tiếp theo sẽ là tiến hàng điều chỉnh giá trị max và min của hai cột đồ thị. Trong đó trục phía bên trái sẽ có giá trị Max là tổng số lần xuất hiện các vấn đề và giá trị Min là 0, còn trục phía bên phải sẽ có giá trị Max là 100% và giá trị Min là 0%. Điều chỉnh giá trị bằng cách chọn các trục trái phải → bấm chuột phải → chọn Format Axis.

Bước tiếp theo là điều chỉnh giá trị cho cả hai trục tung.

Bước tiếp theo chúng ta sẽ kéo một đầu đồ thị tuyến tính về 0 bằng cách chèn thêm một hàng có giá trị 0% vào bảng dữ liệu. Sau đó chọn đồ thị → click chuột phải → Select Data → chọn mục %Tích lũy → Edit → Chọn lại dải dữ liệu.

Các vấn đề nằm bên trái của đường thẳng sẽ gây ra đến 80% hậu quả.

Tiếp theo bạn có thể điều chỉnh lại cách trình bày của đồ thị sao cho sinh động và trực quan hơn với các phương pháp sau:

- Chọn vào từng biểu đồ và click chuột phải → Add data label để hiển thị các số liệu.
- Tăng độ rộng của cột bằng cách click chuột phải vào cột → Chọn Format Data Series → Gap Width = 0.
- Căn chỉnh đồ thị cột và đường sao cho cả hai đồ thị tương quan với nhau bằng cách bấm chọn đồ thị → Layout → Axes → Secondary Horizontal. Bấm chuột phải vào dòng tên cột bên trên, chọn Format Axis → Axis Position → On tick marks. Sau đó chọn Labels → Labels Position → None.

Sau khi thực hiện các bước vẽ biểu đồ trên, chúng ta sẽ có các kết quả như sau:

Các vấn đề nằm bên trái của đường thẳng sẽ gây ra đến 80% hậu quả.

>>>> Tìm Hiểu Về: Mô hình Servqual là gì? 10 thành phần cơ bản của Servqual

4.2 Phân tích Pareto Chart

Khi phân tích biểu đồ Pareto chúng ta sẽ sử dụng quy tắc *biểu đồ Pareto 80/20* nghĩa là 20% vấn đề dẫn đến 80% hậu quả. Cách để phân tích biểu đồ sẽ được thực hiện như sau:

- Từ trục tung phần trăm bên phải chúng ta kẻ một đường thẳng bắt đầu từ vị trí 80% đến khi chạm đồ thị Line.
- Từ vị trí hai đường chạm nhau chúng ta tiếp tục vẽ một đường thẳng vuông góc với trục hoành.

Khi hoàn thành chúng ta sẽ có được biểu đồ như sau.

Các vấn đề nằm bên trái của đường thẳng sẽ gây ra đến 80% hậu quả.

Từ đây ta có thể suy ra rằng những vấn đề nào nằm phía bên trái của đường thẳng vuông góc với trục hoành sẽ gây ra đến 80% hậu quả và cần phải được ưu tiên giải quyết trước.

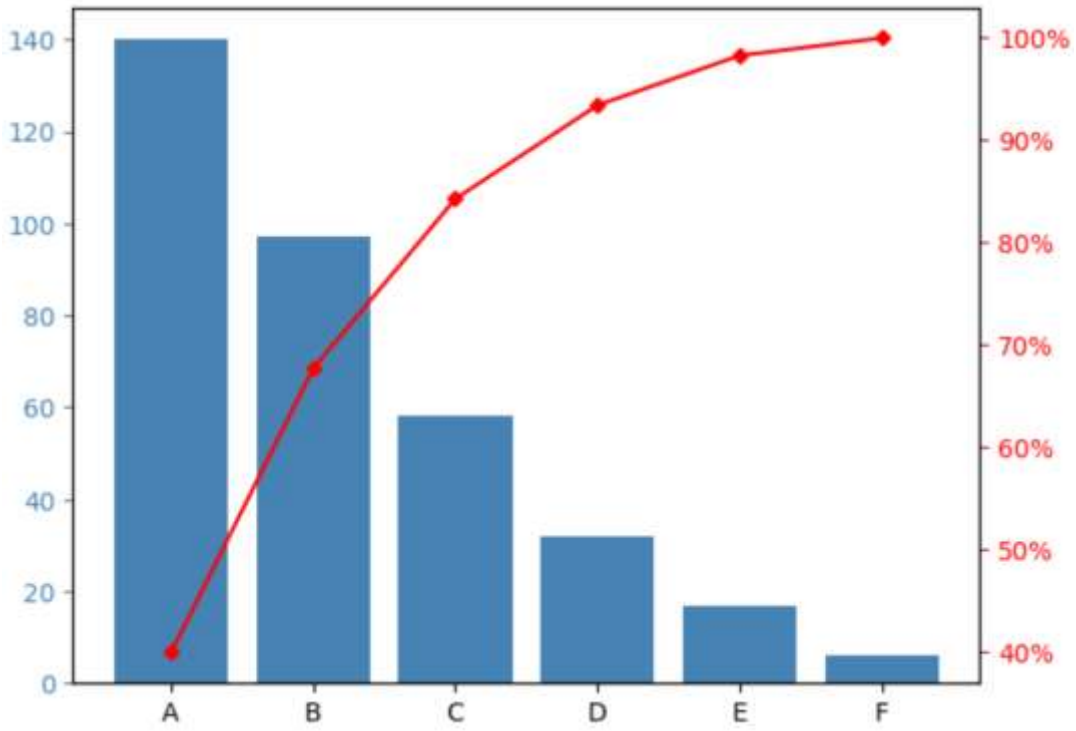
5. Ứng dụng của Pareto Chart
Biểu đồ Pareto thường được dùng trong việc phân tích các sự kiện đa yếu tố và xác định mức ảnh hưởng của yếu tố đó. Việc sử dụng biểu đồ Pareto giúp chọn ra được các vấn đề nào cần được ưu tiên tiến hành giải quyết và hỗ trợ người dùng quản lý công ty một cách hiệu quả nhất.

Rất nhiều doanh nghiệp lâm vào trường hợp đã tìm ra được các vấn đề lớn nhất và các nguyên nhân gây ra vấn đề nhưng lại không biết bắt đầu giải quyết từ đâu. Việc lập một biểu đồ Pareto sẽ giúp các doanh nghiệp cắt giảm được các yếu tố thừa thãi và khoanh vùng được các vấn đề quan trọng. Từ đó, việc này sẽ tối ưu hóa kế hoạch phân bố nguồn lực, giúp giảm chi phí và thời gian.

Sử dụng biểu đồ Pareto giúp chọn ra được các vấn đề nào cần được ưu tiên tiến hành giải quyết.

How to Create a Pareto Chart in Python (Step-by-Step)

A **Pareto chart** is a type of chart that displays the ordered frequencies of categories along with the cumulative frequencies of categories.



This tutorial provides a step-by-step example of how to create a Pareto chart in Python.

Step 1: Create the Data

Suppose we conduct a survey in which we ask 350 different people to identify their favorite cereal brand between brands A, B, C, D, and E.

We can create the following pandas DataFrame to hold the results of the survey:

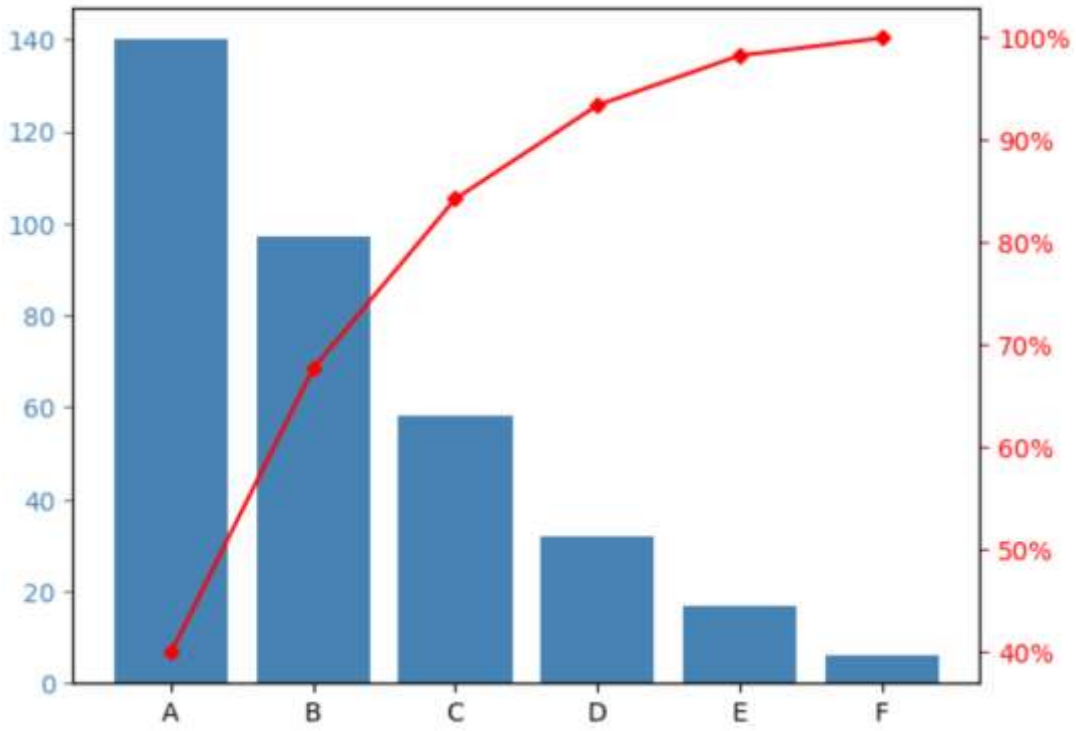
```
import pandas as pd

#create DataFrame
df = pd.DataFrame({'count': [97, 140, 58, 6, 17, 32]})
df.index = ['B', 'A', 'C', 'F', 'E', 'D']

#sort DataFrame by count descending
df = df.sort_values(by='count', ascending=False)

#add column to display cumulative percentage
df['cumperc'] = df['count'].cumsum()/df['count'].sum()*100

#view DataFrame
df

     count     cumperc
```

```
A    140    40.000000
B    97     67.714286
C    58     84.285714
D    32     93.428571
E    17     98.285714
F    6      100.000000
```

Step 2: Create the Pareto Chart

We can use the following code to create the Pareto chart:

```
import matplotlib.pyplot as plt
from matplotlib.ticker import PercentFormatter

#define aesthetics for plot
color1 = 'steelblue'
color2 = 'red'
line_size = 4

#create basic bar plot
fig, ax = plt.subplots()
ax.bar(df.index, df['count'], color=color1)

#add cumulative percentage line to plot
ax2 = ax.twinx()
ax2.plot(df.index, df['cumperc'], color=color2, marker="D", ms=line_size)
ax2.yaxis.set_major_formatter(PercentFormatter())

#specify axis colors
ax.tick_params(axis='y', colors=color1)
ax2.tick_params(axis='y', colors=color2)

#display Pareto chart
plt.show()
```



The x-axis displays the different brands ordered from highest to lowest frequency.

The left-hand y-axis shows the frequency of each brand and the right-hand y-axis shows the cumulative frequency of the brands.

For example, we can see:

- Brand A accounts for about 40% of total survey responses.
- Brands A and B account for about 70% of total survey responses.
- Brands A, B, and C account for about 85% of total survey responses.

And so on.

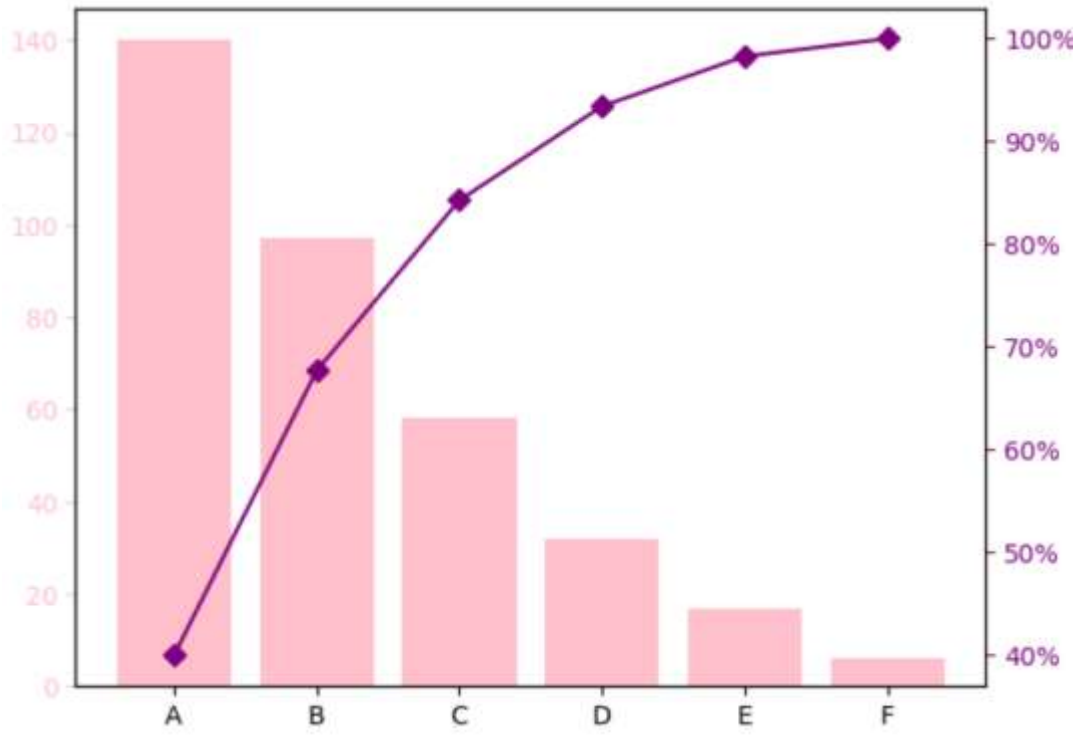
Step 3: Customize the Pareto Chart (Optional)

You can change the colors of the bars and the size of the cumulative percentage line to make the Pareto chart look however you'd like.

For example, we could change the bars to be pink and change the line to be purple and slightly thicker:

```
import matplotlib.pyplot as plt
from matplotlib.ticker import PercentFormatter

#define aesthetics for plot
```

```
color1 = 'pink'
color2 = 'purple'
line_size = 6

#create basic bar plot
fig, ax = plt.subplots()
ax.bar(df.index, df['count'], color=color1)

#add cumulative percentage line to plot
ax2 = ax.twinx()
ax2.plot(df.index, df['cumperc'], color=color2, marker="D", ms=line_size)
ax2.yaxis.set_major_formatter(PercentFormatter())

#specify axis colors
ax.tick_params(axis='y', colors=color1)
ax2.tick_params(axis='y', colors=color2)

#display Pareto chart
plt.show()
```



Additional Resources

The following tutorials explain how to create other common visualizations in Python:

How to Make a Bell Curve in Python
How to Create an Ogive Graph in Python
How to Create a Stem-and-Leaf Plot in Python

Bài 11 - Visualization trong python

16 Sep 2019 - phamdinhkhanh

Menu

1. Giới thiệu về biểu đồ

Visualization hiểu một cách đơn giản là hình ảnh hóa dựa trên dữ liệu. Khái niệm của visualization rất ngắn gọn nhưng trên thực tế visualization lại là một mảng rất rộng và có thể coi là một lĩnh vực kết hợp của khoa học và nghệ thuật bởi nó vừa lên quan đến đồ họa (sử dụng hình học để diễn tả kết quả), vừa liên quan đến khoa học thống kê (sử dụng con số để nói lên vấn đề). Nhờ có visualization, chúng ta có thể dễ dàng đưa ra các so sánh trực quan, tính toán tỷ trọng, nhận biết trend, phát hiện outlier, nhận diện đặc điểm phân phối của biến tốt hơn. Từ đó hỗ trợ quá trình nắm thông tin và đưa ra quyết định tốt hơn. Trong các kĩ năng của data scientist thì visualization là một trong những kĩ năng cơ bản và quan trọng nhất. Thế nhưng nhiều data scientist lại chưa nhận diện được điều này và thường xem nhẹ vai trò của visualization. Trước đây tôi cũng đã từng mắc sai lầm như vậy. Qua kinh nghiệm nhiều năm xây dựng mô hình và phân tích kinh doanh đã giúp tôi nhìn nhận lại vai trò của visualization. Chính vì thế tôi quyết định tổng hợp bài viết này theo cách bao quát và sơ đẳng nhất về visualization trên python như một tài liệu sử dụng khi cần và đồng thời cũng là cách củng cố lại kiến thức.

Nhắc đến visualization chúng ta không thể không nói đến một số dạng biểu đồ cơ bản như: line, barchart, pie, area, boxplot.

Trong đó:

- line: Là biểu đồ đường kết nối các điểm thành 1 đường liền khúc.
- barchart: Biểu diễn giá trị của các nhóm dưới dạng cột.
- pie: Biểu đồ hình tròn biểu diễn phần trăm của các nhóm.
- area: Biểu đồ biểu diễn diện tích của các đường.

- boxplot: Biểu đồ biểu diễn các giá trị thống kê của một biến trên đồ thị bao gồm: Trung bình, Max, Min, các ngưỡng percent tile 25%, 50%, 75%.

Sau đây chúng ta sẽ học cách sử dụng các dạng biểu đồ này trên matplotlib.

1.1. Biểu đồ line

Biểu đồ line là biểu đồ biểu diễn các giá trị dưới dạng những đường. Trên matplotlib. Line được vẽ thông qua plt.plot(). Sau đây ta cùng biểu diễn giá chứng khoán thông qua biểu đồ line.

Lấy dữ liệu chứng khoán của apple

```
1
2
3
4
5
6
7
8
9
10
11
12
13
```

```
import matplotlib.pyplot as plt
import pandas as pd

import datetime
import pandas_datareader.data as web
from pandas import Series, DataFrame


start = datetime.datetime(2010, 1, 1)
end = datetime.datetime(2017, 1, 11)

df = web.DataReader("AAPL", 'yahoo', start, end)
df.tail()
```

	High	Low	Open	Close	Volume	Adj Close
Date						
2017-01-05	116.860001	115.809998	115.919998	116.610001	22193600.0	111.727715
2017-01-06	118.160004	116.470001	116.779999	117.910004	31751900.0	112.973305
2017-01-09	119.430000	117.940002	117.949997	118.989998	33561900.0	114.008080
2017-01-10	119.379997	118.300003	118.769997	119.110001	24462100.0	114.123047

2017-01-11	119.930000	118.599998	118.739998	119.750000	27588600.0	114.736275

Biểu diễn giá chứng khoán dưới dạng biểu đồ line

```
1 plt.plot(df['Close'].tail(100))
2 plt.ylabel('Gía chứng khoán')
3 plt.xlabel('Thời gian')
4 plt.title('Gía chứng khoán APPLE')
```

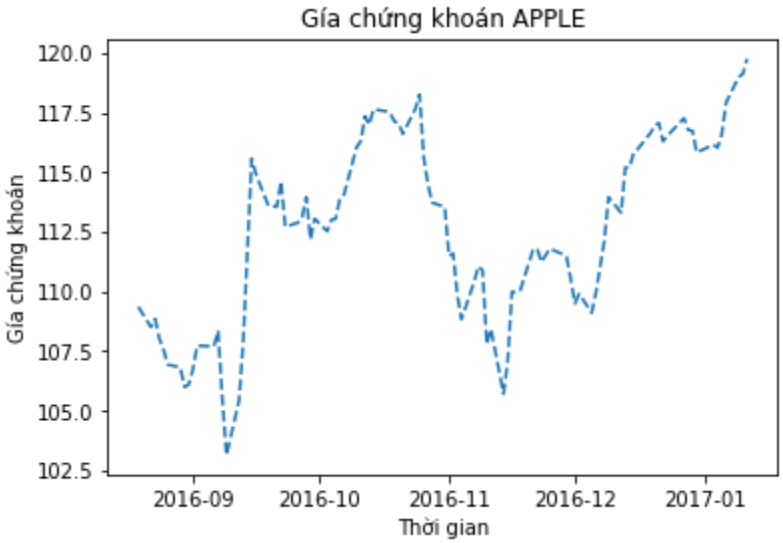
Thay đổi định dạng line

Nếu muốn thay đổi định dạng của line chúng ta sẽ sử dụng thêm 1 tham số khác là linestype. Một số line styles thông dụng: {‘-‘, ‘–’, ‘-.’, ‘:’, ‘’}

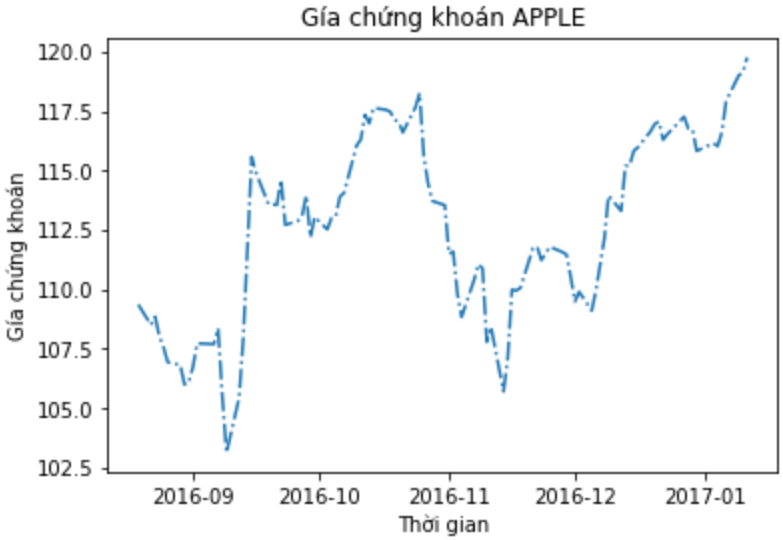
- - : Đường nét liền.
- --: Đường nét đứt dài.
- -.: Đường line nét đứt dài kết hợp với dấu chấm.
- : : Đường line gồm các dấu chấm.

Chẳng hạn để thay đổi line từ dạng đường nét liền sang nét đứt:

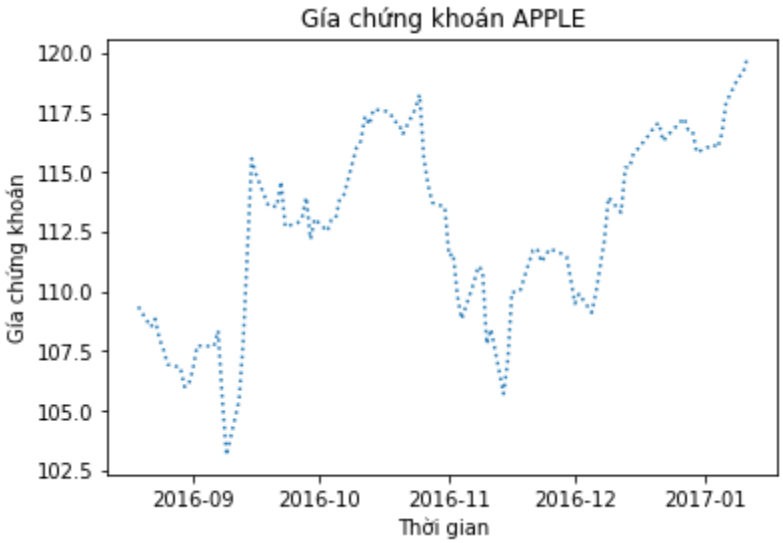
```
1 plt.plot(df['Close'].tail(100), linestyle = '--')
2 plt.ylabel('Gía chứng khoán')
3 plt.xlabel('Thời gian')
4 plt.title('Gía chứng khoán APPLE')
```




```
1  # Đường nét đứt có gạch nối
2  plt.plot(df['Close'].tail(100), linestyle = '-.')
3  plt.ylabel('Gía chứng khoán')
4  plt.xlabel('Thời gian')
5  plt.title('Gía chứng khoán APPLE')
```




```
1  # Đường nét chấm
2  plt.plot(df['Close'].tail(100), linestyle = ':')
3  plt.ylabel('Gía chứng khoán')
4  plt.xlabel('Thời gian')
5  plt.title('Gía chứng khoán APPLE')
```




```
1 plt.plot(df['Close'].tail(100), linestyle = '-')
2 plt.ylabel('Gía chứng khoán')
3 plt.xlabel('Thời gian')
4 plt.title('Gía chứng khoán APPLE')
```

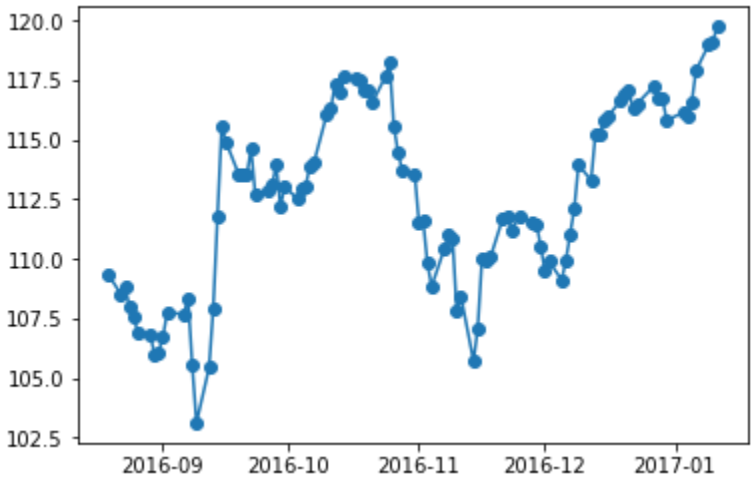
Kết hợp line và point

Bên cạnh line chúng ta còn có thể đánh dấu các điểm mút bằng các point. Hình dạng của point có thể là hình tròn, vuông hoặc tam giác và được khai báo thông qua tham số marker. Các giá trị của marker sẽ tương ứng như sau:

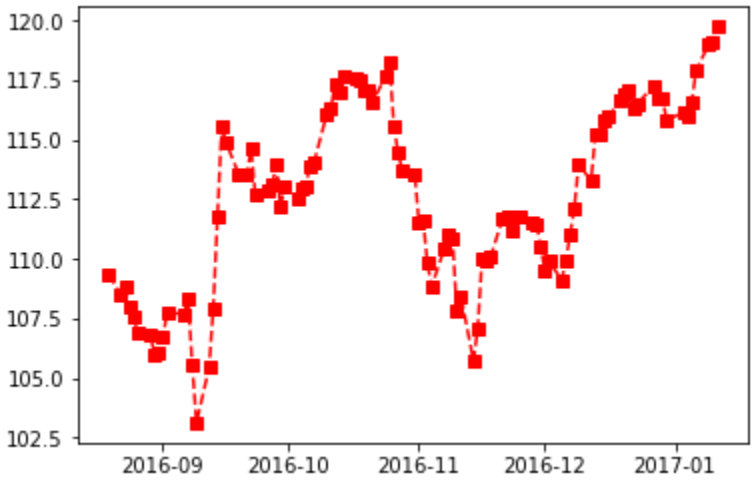
- ^: Hình tam giác
- o: Hình tròn.
- s: Hình vuông (s tức là square).

Bên dưới là một số kết hợp của linstyle và marker.

```
1  plt.plot(df['Close'].tail(100), linestyle = '-', marker = 'o')
```

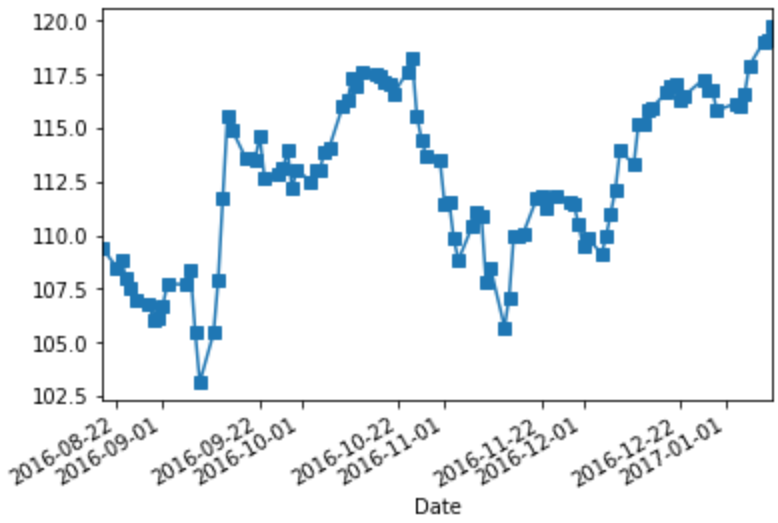



```
1  plt.plot(df['Close'].tail(100), linestyle = '--', marker = 's', color = 'red')
```



Hoặc chúng ta cũng có thể vẽ biểu đồ line từ pandas dataframe.

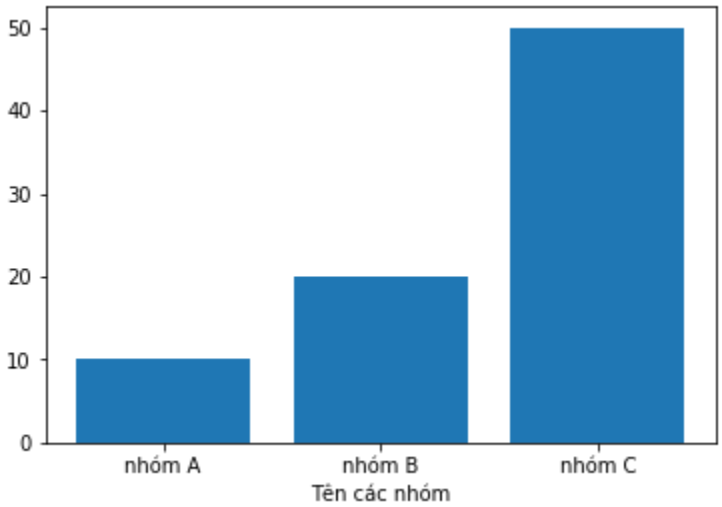
```
1  df['Close'].tail(100).plot(marker = 's')
```



1.2. Biểu đồ barchart

Biểu đồ barchart là dạng biểu đồ có thể coi là phổ biến nhất và được dùng chủ yếu trong trường hợp so sánh giá trị giữa các nhóm thông qua độ dài cột. Để biểu diễn biểu đồ barchart trong python chúng ta sử dụng hàm plt.bar(). Các tham số truyền vào bao gồm tên các nhóm (tham số x) và giá trị của các nhóm (tham số height).

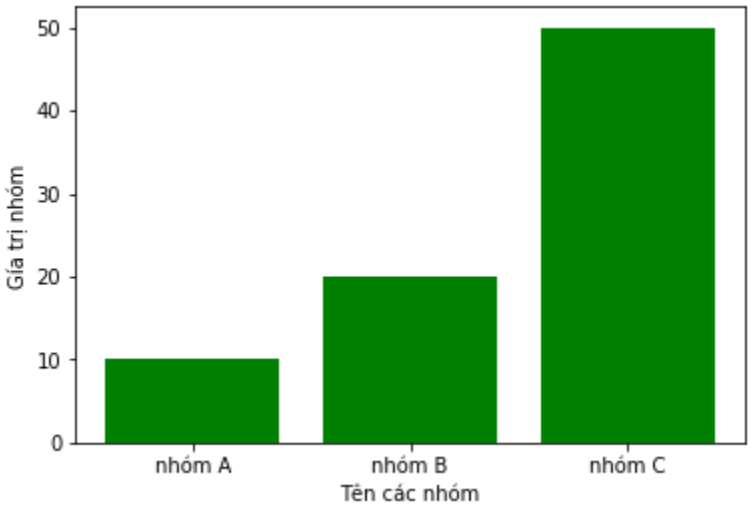
```
1  plt.bar(x = ['nhóm A', 'nhóm B', 'nhóm C'], height = [10, 20, 50])
2  plt.xlabel('Tên các nhóm')
3  plt.ylabel('Gía trị nhóm')
```



Thay đổi màu sắc các nhóm.

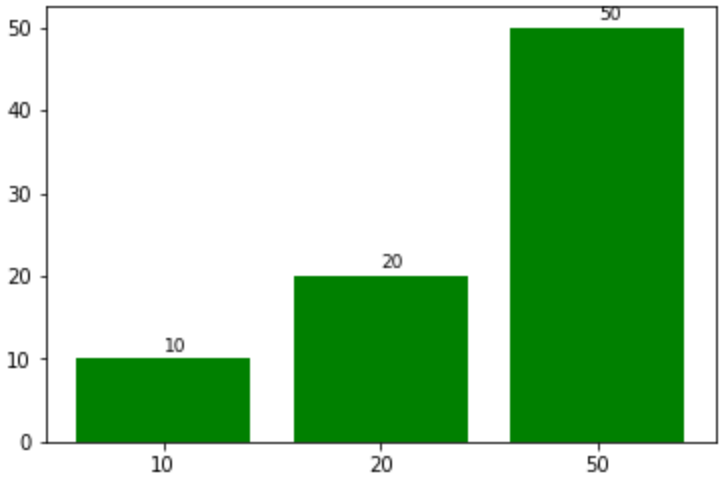
```
1  plt.bar(x = ['nhóm A', 'nhóm B', 'nhóm C'], height = [10, 20, 50], color = 'green')
2  plt.xlabel('Tên các nhóm')
```

```
3  plt.ylabel('Gía trị nhóm')
```



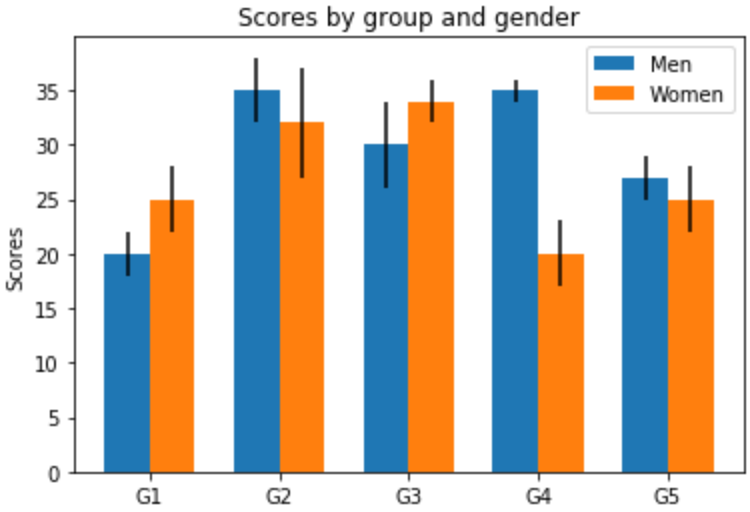
Thêm nhãn giá trị cho các cột bằng tham số plt.text(). Trong đó tham số x và y của plt.text() qui định tọa độ điểm bắt đầu của rectangle chứa label tên của nhóm. s chứa tên labels của nhóm và size qui định kích thước của text.

```
1   x_values = [0, 1, 2]
2   y_values = [10, 20, 50]
3   data_labels = ['10', '20', '50']
4   plt.bar(x = data_labels, height = y_values, color = 'green')
5
6   for i in range(len(data_labels)): # your number of bars
7     plt.text(x = x_values[i], #takes your x values as horizontal positioning argument
8     y = y_values[i]+1, #takes your y values as vertical positioning argument
9     s = data_labels[i], # the labels you want to add to the data
10    size = 9)
```



Chúng ta cũng có thể vẽ biểu đồ của 2 biến trở lên là các barchart liền kề nhau.

```
1   import numpy as np
2
3   men_means, men_std = (20, 35, 30, 35, 27), (2, 3, 4, 1, 2)
4   women_means, women_std = (25, 32, 34, 20, 25), (3, 5, 2, 3, 3)
5
6   ind = np.arange(len(men_means))  # the x locations for the groups
7   width = 0.35  # the width of the bars
8
9   fig, ax = plt.subplots()
10  rects1 = ax.bar(ind - width/2, men_means, width, yerr=men_std,
11                  label='Men')
12  rects2 = ax.bar(ind + width/2, women_means, width, yerr=women_std,
13                  label='Women')
14
15  # Add some text for labels, title and custom x-axis tick labels, etc.
16  ax.set_ylabel('Scores')
17  ax.set_title('Scores by group and gender')
18  ax.set_xticks(ind)
19  ax.set_xticklabels(('G1', 'G2', 'G3', 'G4', 'G5'))
20  ax.legend()
21
22  plt.show()
```

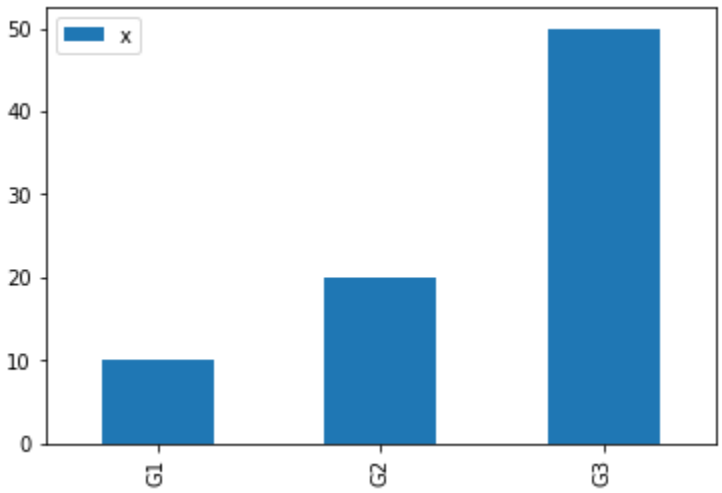


Ta cũng có thể biểu diễn biểu đồ thông qua dataframe.

```
1  df = pd.DataFrame({'x': [10, 20, 50]}, index = ['G1', 'G2', 'G3'])
2  df
```

	x
G1	10
G2	20
G3	50

```
1  df.plot.bar()
```

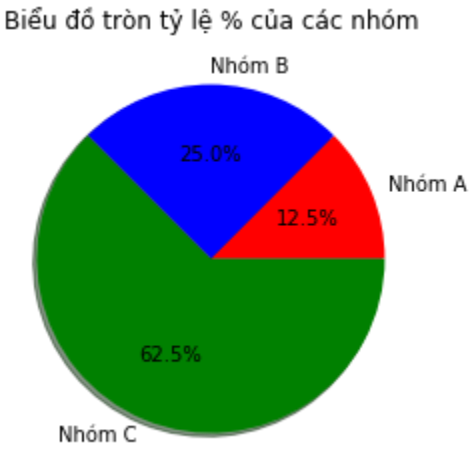


1.3. Biểu đồ tròn

Biểu đồ tròn được sử dụng để visualize tỷ lệ phần trăm các class. Ưu điểm của biểu đồ này là dễ dàng hình dung được giá trị % mà các class này đóng góp vào số tổng. Nhưng nhược điểm là không thể hiện số tuyệt đối.

Để tạo biểu đồ tròn trong mathplotlib.

```
1  import numpy as np
2  plt.pie(x = np.array([10, 20, 50]), # giá trị của các nhóm
3          labels = ['Nhóm A', 'Nhóm B', 'Nhóm C'], # Nhãn của các nhóm
4          colors = ['red', 'blue', 'green'], # Màu sắc của các nhóm
5          autopct = '%1.1f%%', # Format hiển thị giá trị %
6          shadow = True
7         )
8  plt.title('Biểu đồ tròn tỷ lệ % của các nhóm')
```

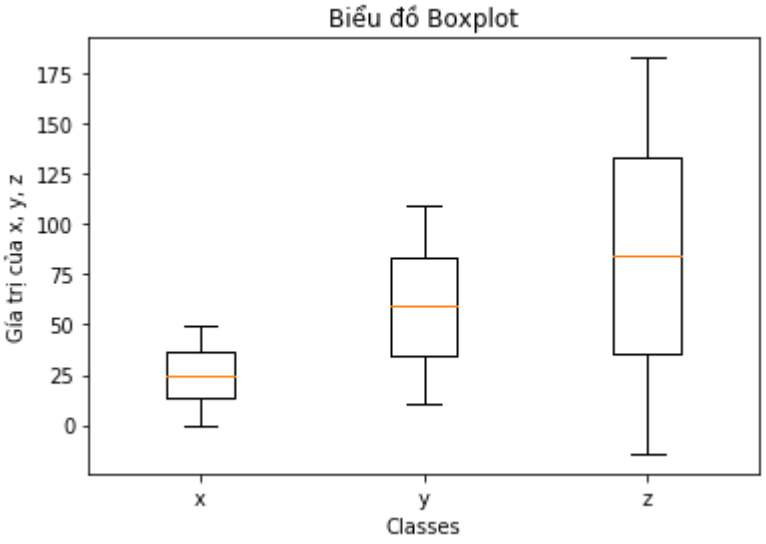


1.4. Biểu đồ boxplot

Biểu đồ boxplot sẽ cho ta biết đặc trưng về phân phối của 1 biến dựa trên các giá trị trung bình, min, max, các khoảng phân vị 25%, 50%, 75%. Đây là biểu đồ được sử dụng nhiều trong chứng khoán và thống kê học để so sánh các biến với nhau.

```
1  import numpy as np
2  x = np.random.randn(100) + np.arange(0, 100) * 0.5
3  y = np.random.randn(100) + np.arange(0, 100) * 1.0 + 10
4  z = np.random.randn(100) + np.arange(0, 100) * 2 - 15
5
6  plt.boxplot([x, y, z],
7              labels = ['x', 'y', 'z'],
8              showfliers = True)
9
```

```
10  plt.title('Biểu đồ Boxplot')
11  plt.xlabel('Classes')
12  plt.ylabel('Gía trị của x, y, z')
```



1.5. Vẽ biểu đồ trên dataframe

Định dạng dataframe của pandas không chỉ hỗ trợ các truy vấn và thống kê dữ liệu có cấu trúc nhanh hơn mà còn support vẽ biểu đồ dưới dạng matplotlib-based. Sau đây chúng ta cùng sử dụng dataframe để vẽ các đồ thị cơ bản.

Để tìm hiểu kĩ hơn về thống kê và vẽ biểu đồ trên dataframe các bạn có thể tham khảo bài Giới thiệu pandas.

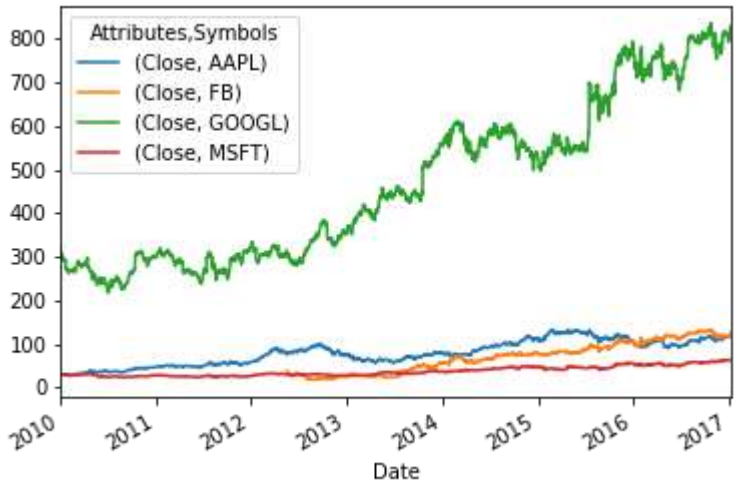
```
1           import pandas as pd
2           import datetime
3           import pandas_datareader.data as web
4           from pandas import Series, DataFrame
5
6
7           start = datetime.datetime(2010, 1, 1)
8           end = datetime.datetime(2017, 1, 11)
9
10          df = web.DataReader(["AAPL", "GOOGL", "MSFT", "FB"], 'yahoo', start, end)
11          # Chỉ lấy giá close
12          df = df[['Close']]
13          df.tail()
```

Attributes	**Close**

Symbols	AAPL	FB	GOOGL	MSFT
Date				
2017-01-05	116.610001	120.669998	813.020020	62.299999
2017-01-06	117.910004	123.410004	825.210022	62.840000
2017-01-09	118.989998	124.900002	827.179993	62.639999
2017-01-10	119.110001	124.349998	826.010010	62.619999
2017-01-11	119.750000	126.089996	829.859985	63.189999

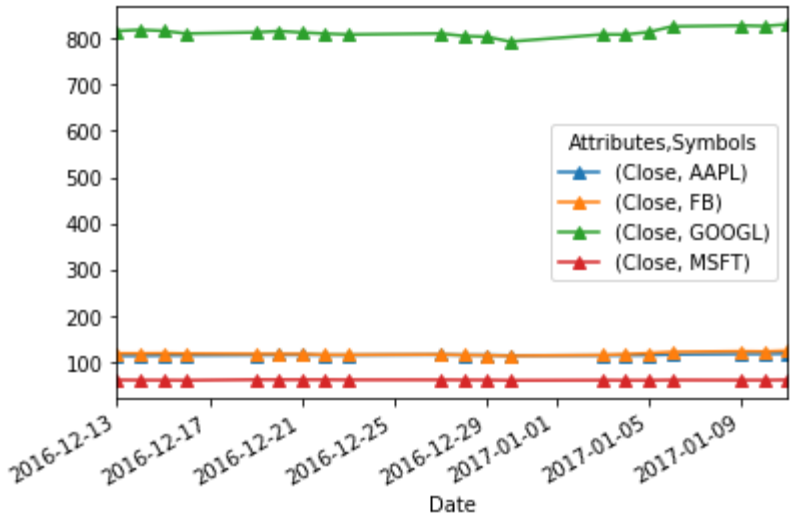
Biểu đồ line

```
1  df.plot()
```



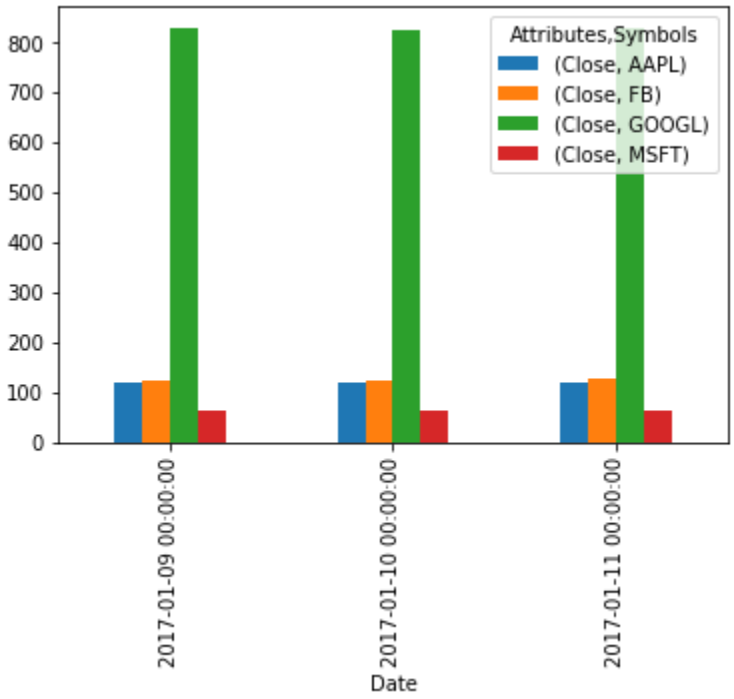
Biểu đồ line kết hợp point

```
1  df.tail(20).plot(linestyle = '-', marker = '^')
```



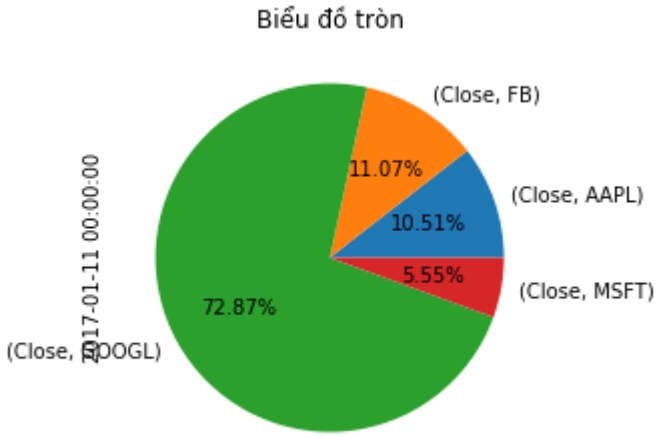
Biểu đồ barchart

```
1  df.tail(3).plot.bar()
```



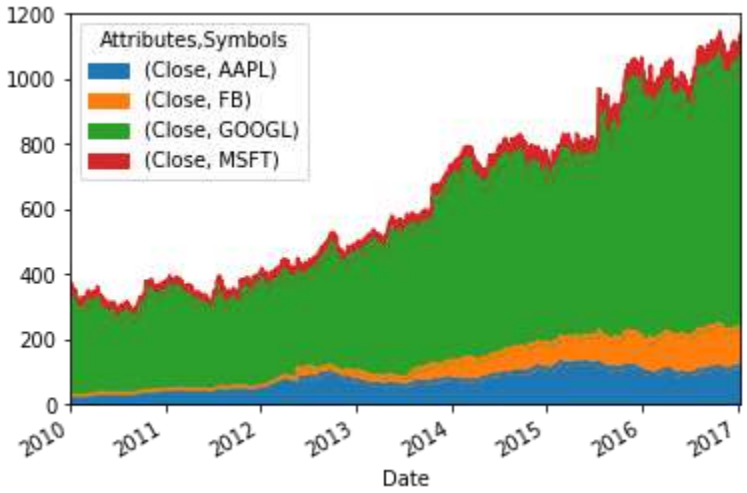
Biểu đồ tròn

```
1  df.iloc[-1, :].plot.pie(autopct = '%.2f%%')
2  plt.title('Biểu đồ tròn')
```



Biểu đồ diện tích

```
1  df.plot.area()
```



Vùng có diện tích càng lớn thì khoảng chênh lệch về giá theo thời gian của nó càng lớn và các vùng có diện tích nhỏ hơn cho thấy các mã chứng khoán ít có sự chênh lệch về giá theo thời gian.

1.6. Biểu đồ heatmap.

Heatmap là biểu đồ sử dụng cường độ màu sắc để thể hiện độ lớn của giá trị. Khi đó các giá trị lớn sẽ được làm nổi bật bằng các vùng màu có cường độ ánh sáng mạnh và các giá trị nhỏ hơn sẽ được thể hiện bằng các mảng màu nhạt hơn. Các trường hợp thường sử dụng heatmap:

- Biểu đồ hệ số tương quan.
- Biểu đồ địa lý về cảnh báo thiên tai.
- Biểu đồ mật độ dân số.

- Biểu đồ crazy egg trong đo lường các component được sử dụng nhiều trong 1 website hoặc app. …

Trong machine learning ứng dụng lớn nhất của heatmap có lẽ là thể hiện các giá trị của hệ số tương quan. Ta sẽ cùng tìm hiểu cách vẽ biểu đồ heatmap biểu diễn hệ số tương quan.

```
1    # Tính correlation
2    df_cor = df.corr()
3    df_cor
```

	Attributes	**Close**			
	Symbols	**AAPL**	**FB**	**GOOGL**	**MSFT**
Attributes	**Symbols**				
Close	**AAPL**	1.000000	0.707744	0.795063	0.820183
	FB	0.707744	1.000000	0.954793	0.958231
	GOOGL	0.795063	0.954793	1.000000	0.960193
	MSFT	0.820183	0.958231	0.960193	1.000000

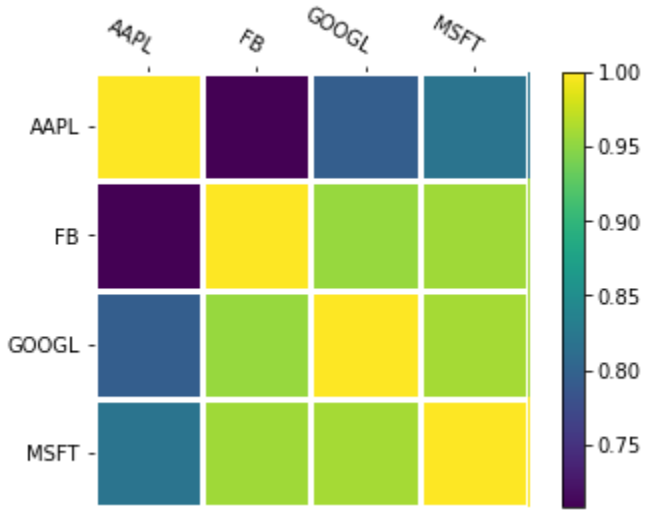
Để vẽ biểu đồ heatmap chúng ta có thể sử dụng hàm số heatmap() như bên dưới.

```
1   def heatmap(data, row_labels, col_labels, ax=None,
2               cbar_kw={}, cbarlabel="", **kwargs):
3       """
4       Create a heatmap from a numpy array and two lists of labels.
5
6       Parameters
7       ----------
8       data
9           A 2D numpy array of shape (N, M).
10      row_labels
11          A list or array of length N with the labels for the rows.
12      col_labels
13          A list or array of length M with the labels for the columns.
14      ax
15          A `matplotlib.axes.Axes` instance to which the heatmap is plotted.  If
16          not provided, use current axes or create a new one.  Optional.
17      cbar_kw
18          A dictionary with arguments to `matplotlib.Figure.colorbar`.  Optional.
19      cbarlabel
20          The label for the colorbar.  Optional.
21      **kwargs
```

```
    All other arguments are forwarded to `imshow`.
    """

    if not ax:
        ax = plt.gca()

    # Plot the heatmap
    im = ax.imshow(data, **kwargs)

    # Create colorbar
    cbar = ax.figure.colorbar(im, ax=ax, **cbar_kw)
    cbar.ax.set_ylabel(cbarlabel, rotation=-90, va="bottom")

    # We want to show all ticks...
    ax.set_xticks(np.arange(data.shape[1]))
    ax.set_yticks(np.arange(data.shape[0]))
    # ... and label them with the respective list entries.
    ax.set_xticklabels(col_labels)
    ax.set_yticklabels(row_labels)

    # Let the horizontal axes labeling appear on top.
    ax.tick_params(top=True, bottom=False,
                   labeltop=True, labelbottom=False)

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=-30, ha="right",
             rotation_mode="anchor")

    # Turn spines off and create white grid.
    for edge, spine in ax.spines.items():
        spine.set_visible(False)

    ax.set_xticks(np.arange(data.shape[1]+1)-.5, minor=True)
    ax.set_yticks(np.arange(data.shape[0]+1)-.5, minor=True)
    ax.grid(which="minor", color="w", linestyle='-', linewidth=3)
    ax.tick_params(which="minor", bottom=False, left=False)

    return im, cbar
```

Hàm số sẽ có tác dụng thiết lập các bảng heatmap và labels của trục x, y trên đồ thị.

```
inds = ['AAPL', 'FB', 'GOOGL', 'MSFT']
fig, ax = plt.subplots()

im, cbar = heatmap(df_cor, row_labels = inds, col_labels = inds)
```

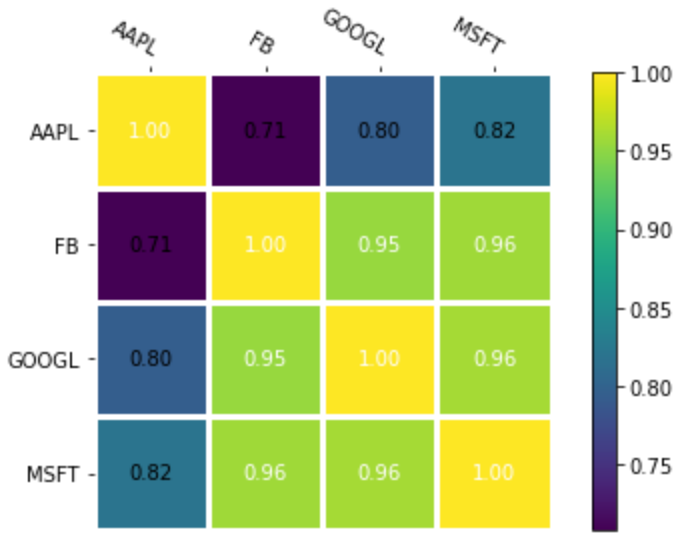


Chúng ta sẽ thêm titles giá trị các biến nằm trong df_cor vào các ô giá trị tương ứng thông qua hàm số annotate_heatmap()

```
1   import matplotlib
2
3   def annotate_heatmap(im, data=None, valfmt="{x:.2f}",
4                        textcolors=["black", "white"],
5                        threshold=None, **textkw):
6       """
7       A function to annotate a heatmap.
8
9       Parameters
10      ----------
11      im
12          The AxesImage to be labeled.
13      data
14          Data used to annotate.  If None, the image's data is used.  Optional.
15      valfmt
16          The format of the annotations inside the heatmap.  This should either
17          use the string format method, e.g. "$ {x:.2f}", or be a
18          `matplotlib.ticker.Formatter`.  Optional.
19      textcolors
20          A list or array of two color specifications.  The first is used for
21          values below a threshold, the second for those above.  Optional.
22      threshold
23          Value in data units according to which the colors from textcolors are
24          applied.  If None (the default) uses the middle of the colormap as
25          separation.  Optional.
26      **kwargs
```

```
    All other arguments are forwarded to each call to `text` used to create
    the text labels.
"""

if not isinstance(data, (list, np.ndarray)):
    data = im.get_array()

# Normalize the threshold to the images color range.
if threshold is not None:
    threshold = im.norm(threshold)
else:
    threshold = im.norm(data.max())/2.

# Set default alignment to center, but allow it to be
# overwritten by textkw.
kw = dict(horizontalalignment="center",
          verticalalignment="center")
kw.update(textkw)

# Get the formatter in case a string is supplied
if isinstance(valfmt, str):
    valfmt = matplotlib.ticker.StrMethodFormatter(valfmt)

# Loop over the data and create a `Text` for each "pixel".
# Change the text's color depending on the data.
texts = []
for i in range(data.shape[0]):
    for j in range(data.shape[1]):
        kw.update(color=textcolors[int(im.norm(data[i, j]) > threshold)])
        text = im.axes.text(j, i, valfmt(data[i, j], None), **kw)
        texts.append(text)

return texts
```

```
inds = ['AAPL', 'FB', 'GOOGL', 'MSFT']
fig, ax = plt.subplots()
im, cbar = heatmap(df_cor, row_labels = inds, col_labels = inds)
texts = annotate_heatmap(im, valfmt="{x:.2f}")
fig.tight_layout()
plt.show()
```



Hoặc ta có thể visualize biểu đồ heatmap thông qua package seaborn.

```
1 import seaborn as sns
2
3 sns.heatmap(df_cor, annot=True)
```

2. Các biểu đồ biểu diễn phân phối.

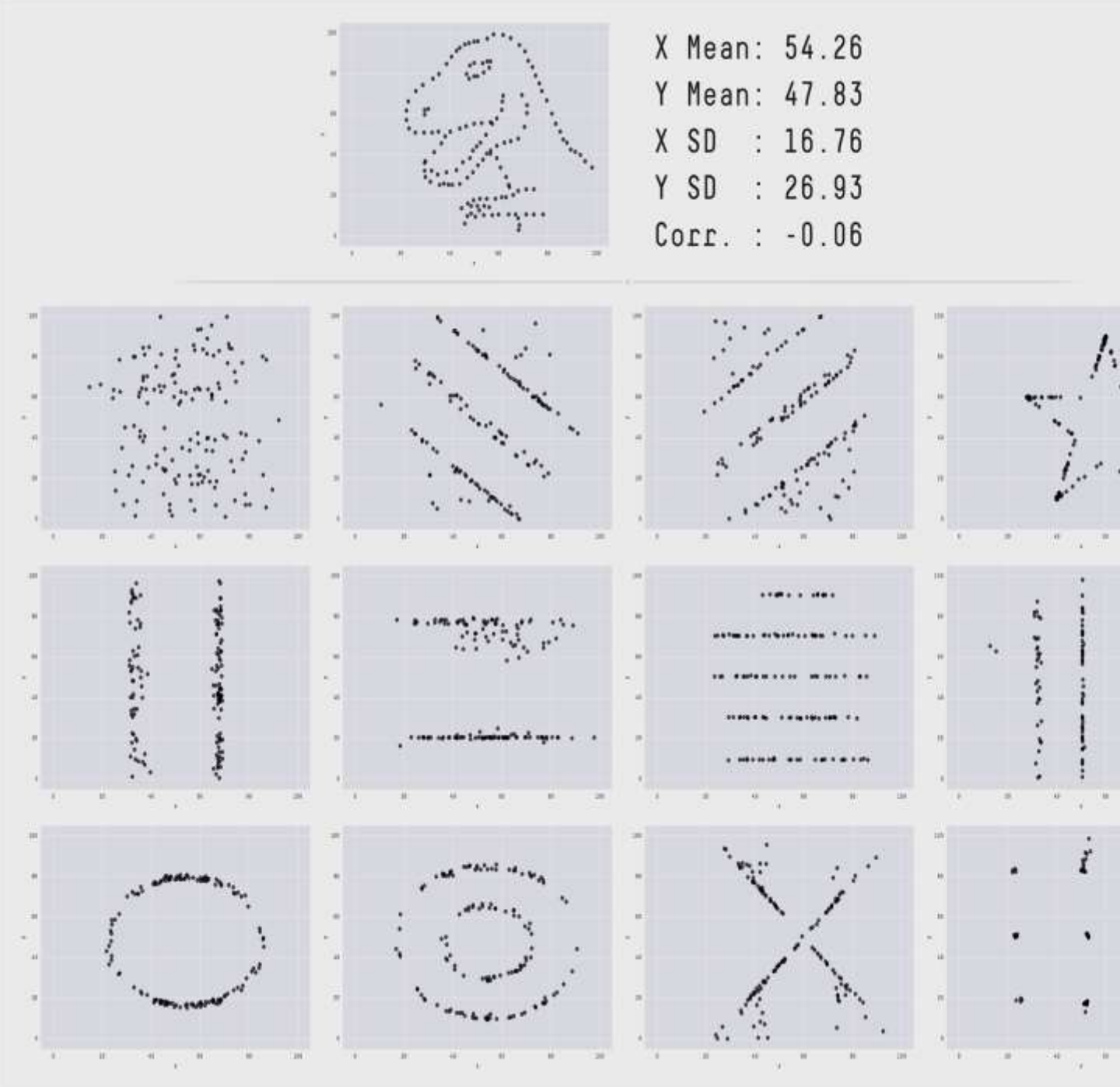
2.1. Density plot

Mỗi một bộ dữ liệu đều có một đặt trưng riêng của nó. Để mô hình hóa những đặc trưng này, thống kê học sử dụng thống kê mô tả như tính mean, max, median, standard deviation, percentile. Để tính thống kê mô tả cho một dataset dạng pandas dataframe trong python đơn giản ta sử dụng hàm describe().

```
    from sklearn import datasets
1   iris = datasets.load_iris()
2
3   X = iris.data
4   y = iris.target
5
6   import pandas as pd
7   dataset = pd.DataFrame(data = X,
8   columns = iris['feature_names'])
9   dataset['species'] = y
10  print('dataset.shape: ', dataset.sha
11  pe)
12
    dataset.describe()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333	1.000000
std	0.828066	0.435866	1.765298	0.762238	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

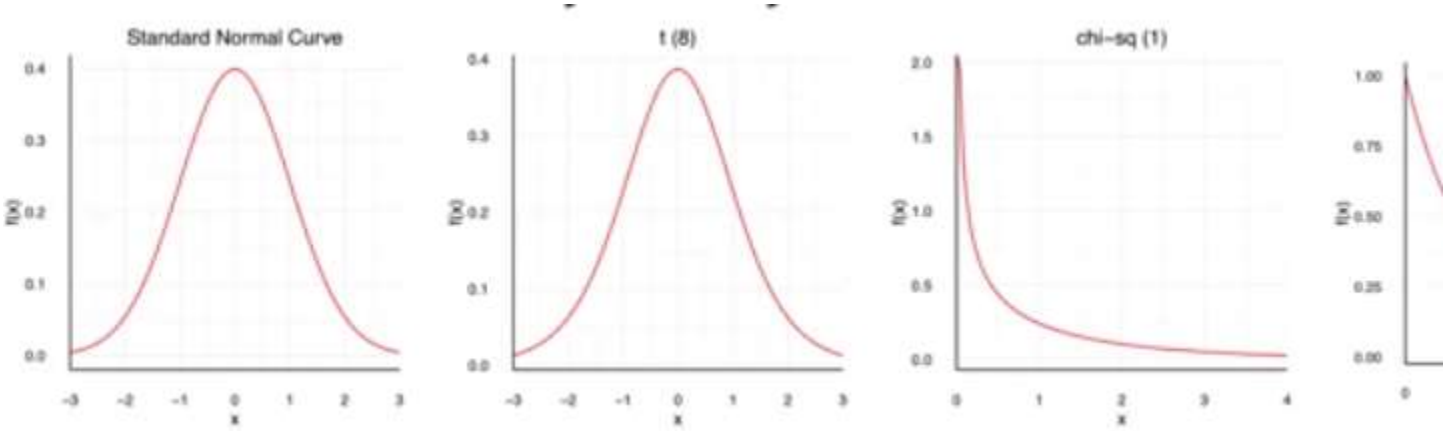
Tuy nhiên không phải lúc nào thống kê mô tả là duy nhất đối với một bộ dữ liệu. Một ví dụ tiêu biểu về phân phối hình chú khủng long.



Hình 1: Đồ thị hình chú khủng long và các hình bên dưới có hình dạng hoàn toàn khác biệt nhau nhưng đều dựa trên 2 chuỗi , có chung thống kê mô tả mean, phương sai và hệ số tương quan.
Do đó không nên hoàn toàn tin tưởng vào thống kê mô tả mà bên cạnh đó chúng ta cần visualize phân phối của dữ liệu.

Trong thống kê mỗi một bộ dữ liệu đều được đặc trưng bởi một hàm mật độ xác suất (pdf - probability density function). Các phân phối điển hình như standard normal, T-student, poisson,

fisher, chi-squared đều được đặc trưng bởi những hình dạng đồ thị phân phối của hàm mật độ xác suất khác nhau.



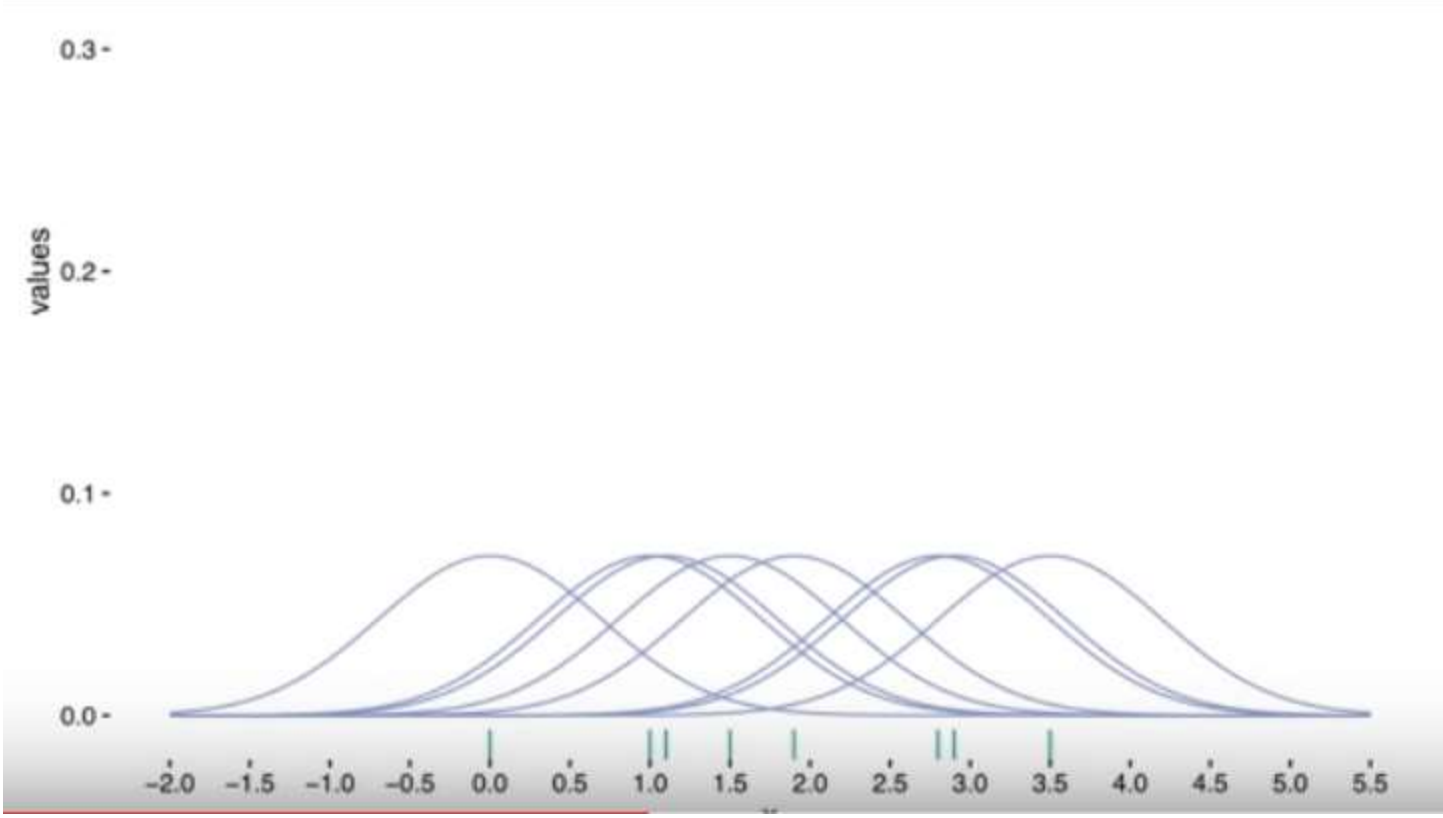
Hình 2: Đồ thị hàm mật độ xác suất của những phân phối xác suất standard normal, T-student, poisson, fisher, chi-squared.
Về mặt lý thuyết (*theoreotical*) những phân phối này đều dựa trên những phương trình xác định.

Trong thực nghiệm (*empirical*) nhiều bộ dữ liệu cho thấy có hình dạng tương đồng với những phân phối này.

Để tìm ra một hình dạng tương đối cho hàm mật độ xác suất của một bộ dữ liệu chúng ta sẽ sử dụng phương pháp KDE (*kernel density estimate*)

KDE là gì?

Hãy tưởng tượng tại mỗi một quan sát ta có đường cong phân phối đặc trưng. Hàm kernel sẽ giúp xác định hình dạng của đường cong trong khi độ rộng của đường cong được xác định bởi bandwidth - h. Phương pháp KDE sẽ tính tổng của các đường cong chạy dọc theo trục để hình thành nên đường cong mật độ xác suất tổng quát cho dữ liệu.

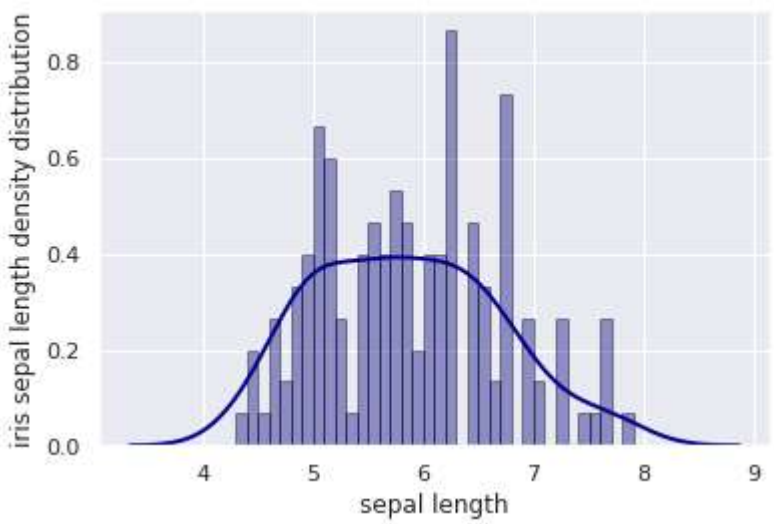


Hình 3: Phương pháp KDE giúp xây dựng hình dạng phân phối của dữ liệu. Ở những nơi có nhiều điểm dữ liệu tập trung thì số lượng các đường cong chồng lấn lên nhau sẽ nhiều hơn và do đó khi tính tổng cộng dồn của nó ta sẽ thu được một giá trị lũy kế kernel density lớn hơn và trái lại với những nơi có nhiều ít điểm dữ liệu tập trung.
Ngoài ra hình dạng bandwidth - h sẽ giúp xác định mức độ khái quát hoặc chi tiết của đường cong. Nếu ta muốn đường cong smoothing hơn thì cần thiết lập h lớn hơn và đường cong mấp mô hơn thì h cần nhỏ hơn. Tuy nhiên bạn đọc cũng không cần quá quan tâm đến bandwidth vì cách tốt hơn là sử dụng giá trị mặc định được tính trong matplotlib.

Bên dưới ta sẽ thực hành vẽ hàm mật độ xác suất của độ dài các đài hoa thông qua hàm distplot() của package seaborn.

```
1  import matplotlib.pyplot as plt
2  import seaborn as sns
3
4  sns.distplot(dataset['sepal length (cm)'],
5               hist = True,
6               bins=int(180/5),
7               kde = True,
8               color = 'darkblue',
9               hist_kws={'edgecolor':'black'},
10              kde_kws={'linewidth':2})
11 # Khai báo tiêu đề cho trục x
12 plt.xlabel('sepal length')
```

```
13  # Khai báo tiêu đề cho trục y
14  plt.ylabel('iris sepal length density distribution')
15  plt.show()
```



Tham số quan trọng nhất của hàm số là kde = True để xác nhận chúng ta sử dụng phương pháp KDE để tính toán đường cong hàm mật độ. Các tham số khác như color, hist_kws, kde_kws chỉ là những tham số râu ria qui định màu sắc, format, kích thước. Ngoài ra hist = True để thiết lập đồ thị histogram mà chúng ta sẽ tìm hiểu bên dưới.

2.2 Histogram plot

Histogram là biểu đồ áp dụng trên một biến liên tục nhằm tìm ra phân phối tần suất trong những khoảng giá trị được xác định trước của một biến.

Có 2 cách tạo biểu đồ histogram theo các khoảng giá trị đó là:

- Phân chia các khoảng giá trị có độ dài bằng nhau và độ dài được tính toán từ số lượng bins khai báo.
- Tự định nghĩa các khoảng giá trị dựa trên bins_edge là các đầu mút của khoảng.

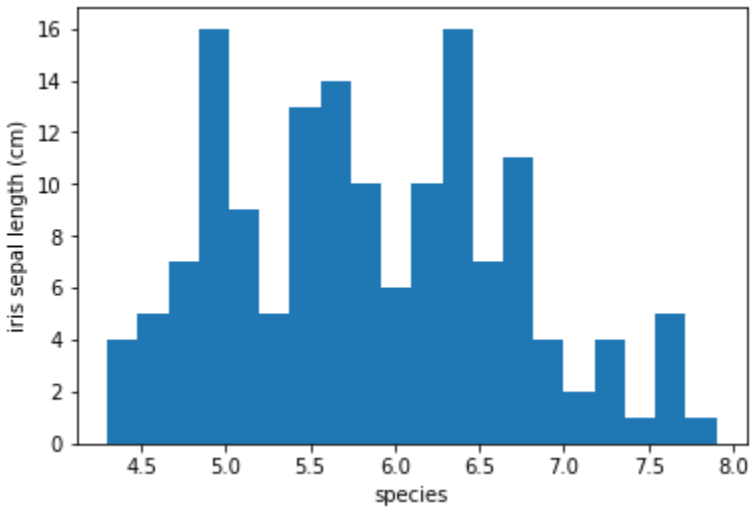
Biểu đồ histogram có thể được visualize qua package mathplotlib. Các biểu đồ của mathplotlib được thể được setup dưới nhiều style đồ họa khác nhau (thay đổi về theme, kiểu chữ, … nhưng về bản chất vẫn là các đối tượng của mathplotlib). Trong đó seaborn, một matplotlib-based package xuất sắc được phát triển bởi Michael Waskom là một trong những style được ưa chuộng nhất. Trong bài viết này chúng ta sẽ setup style của đồ thị dưới dạng seaborn.

Bên dưới là biểu đồ histogram của độ rộng đài hoa visualize theo 2 cách: Khai báo bins và khai báo bins edge.

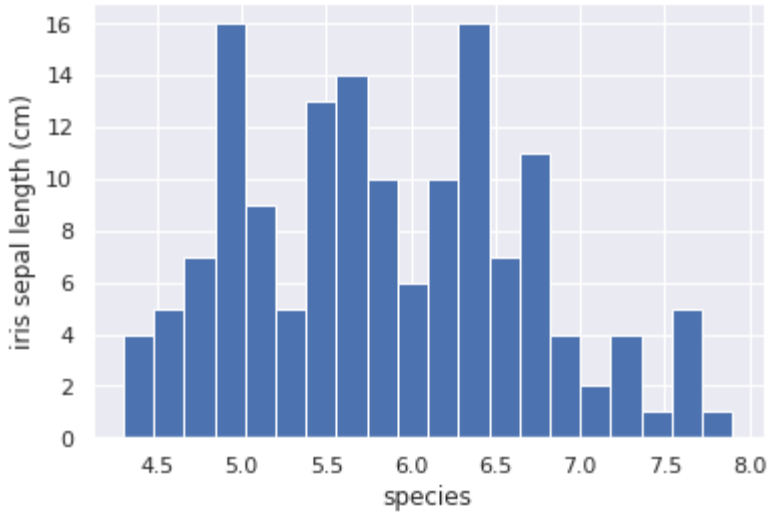
Đồ thị histogram theo số lượng bins = 20

Nếu không set style hiển thị mặc định là seaborn đồ thị sẽ là:

```
import matplotlib.pyplot as plt

plt.hist(dataset['sepal length (cm)'], bins = 20)
# Khai báo tiêu đề cho trục x
plt.xlabel('species')
# Khai báo tiêu đề cho trục y
plt.ylabel('iris sepal length (cm)')
plt.show()
```




```
import matplotlib.pyplot as plt
import seaborn as sns

# Setup style của matplotlib dưới dạng seaborn

sns.set()
plt.hist(dataset['sepal length (cm)'], bins = 20)
# Khai báo tiêu đề cho trục x
plt.xlabel('species')
# Khai báo tiêu đề cho trục y
plt.ylabel('iris sepal length (cm)')
plt.show()
```

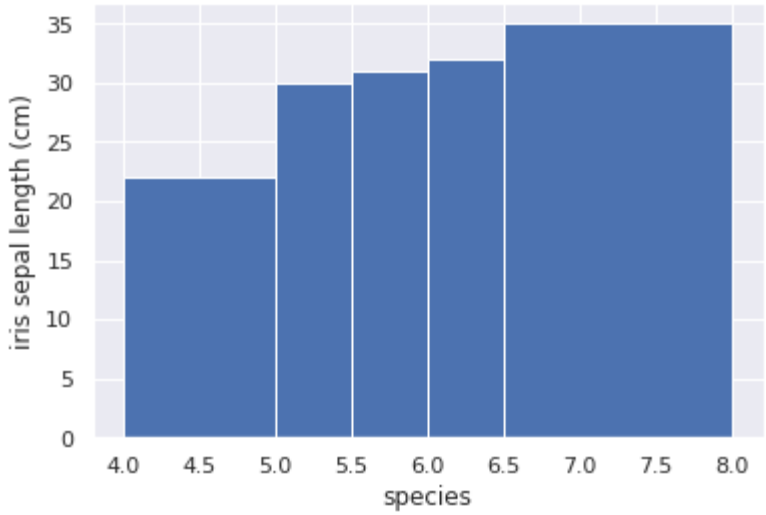


Ta thấy theme của đồ thị được chuyển sang màu xám nhạt và giữa các cột histogram có viền trắng phân chia nhìn rõ ràng hơn. Đây là những thay đổi về đồ họa rất nhỏ nhưng giúp đồ thị trở nên đẹp mắt hơn so với mặc định của mathplotlib.

Đồ thị histogram theo bin edges

Các bin edges được khai báo thông qua cũng cùng tham số bins, giá trị được truyền vào khi đó là 1 list các điểm đầu mút. Từ đó giúp đồ thị linh hoạt hơn khi có thể hiệu chỉnh độ dài các bins tùy thích.

```
1  import matplotlib.pyplot as plt
2  import seaborn as sns
3
4  bin_edges = [4, 5, 5.5, 6, 6.5, 8]
5  plt.hist(dataset['sepal length (cm)'], bins = bin_edges)
6  # Khai báo tiêu đề cho trục x
7  plt.xlabel('species')
8  # Khai báo tiêu đề cho trục y
9  plt.ylabel('iris sepal length (cm)')
10 plt.show()
```



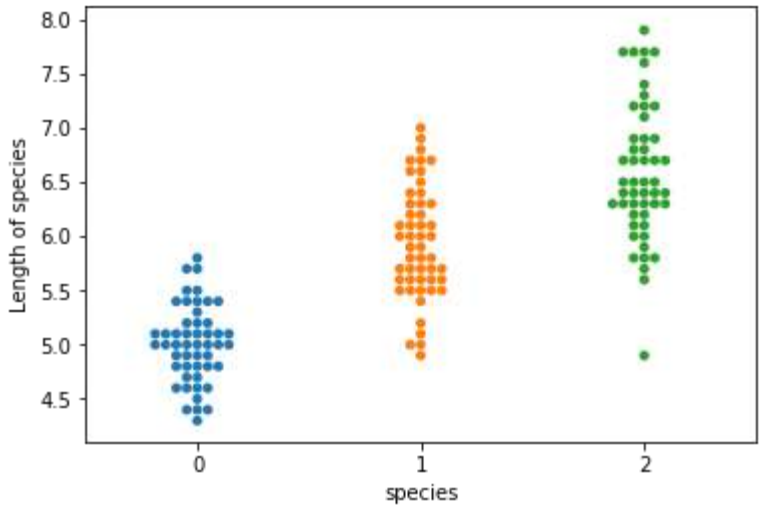
Ta thấy nhược điểm của histogram đó là đồ thị sẽ bị thay đổi tùy theo số lượng bins được thiết lập hoặc list các đầu mút range được khai báo. Do đó để nhận biết được hình dạng phân phối của dữ liệu, một biểu đồ khác thường được sử dụng thay thế đó chính là swarn plot.

2.3. Swarn plot

Swarn plot là biểu đồ point biểu diễn các giá trị dưới dạng các điểm. Các giá trị trên đồ thị bằng đúng với giá trị thật của quan sát. Do đó không xảy ra mất mát thông tin như histogram. Thông qua swarn plot ta có thể so sánh được phân phối của các class khác nhau trên cùng một đồ thị.

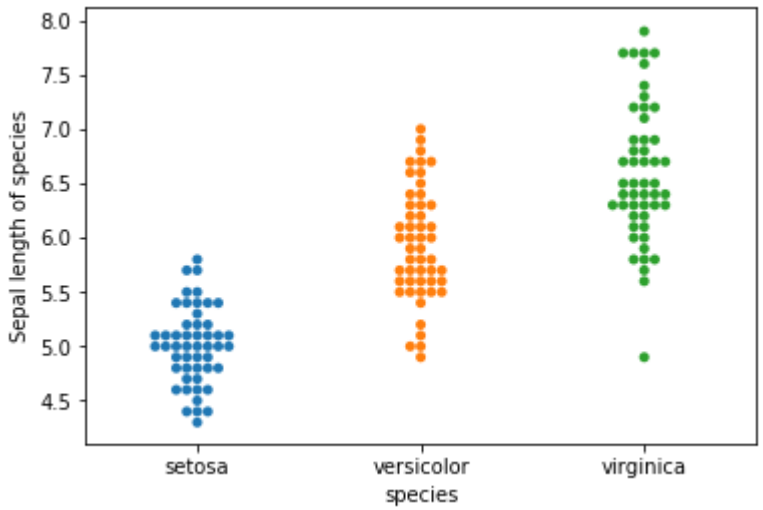
Hãy hình dung qua ví dụ cụ thể khi visualization dữ liệu iris theo chiều dài, rộng cánh hoa và đài hoa.

```
1 import seaborn as sn
2 import matplotlib.pyplot as plt
3
4 sn.swarmplot(x = 'species', y = 'sepal length (cm)', data = dataset)
5 plt.xlabel('species')
6 plt.ylabel('Length of species')
7 plt.show()
```



Muốn thay nhãn của các x = [0, 1, 2] sang target_names = [‘setosa’, ‘versicolor’, ‘virginica’] ta sử dụng hàm plt.xticks().

```
1  import seaborn as sn
2  import matplotlib.pyplot as plt
3
4  sn.swarmplot(x = 'species', y = 'sepal length (cm)', data = dataset)
5  plt.xlabel('species')
6  # Thêm plt.xticks() để thay nhãn của x
7  plt.xticks(ticks = [0, 1, 2], labels = ['setosa', 'versicolor', 'virginica'])
8  plt.ylabel('Sepal length of species')
9  plt.show()
```



Từ biểu đồ ta nhận thấy độ dài đài hoa có sự khác biệt ở cả 3 giống hoa iris. Trung bình độ dài của đài hoa tăng dần từ setosa, versicolor đến virginica. Vì swarm là đồ thị giữ nguyên giá trị

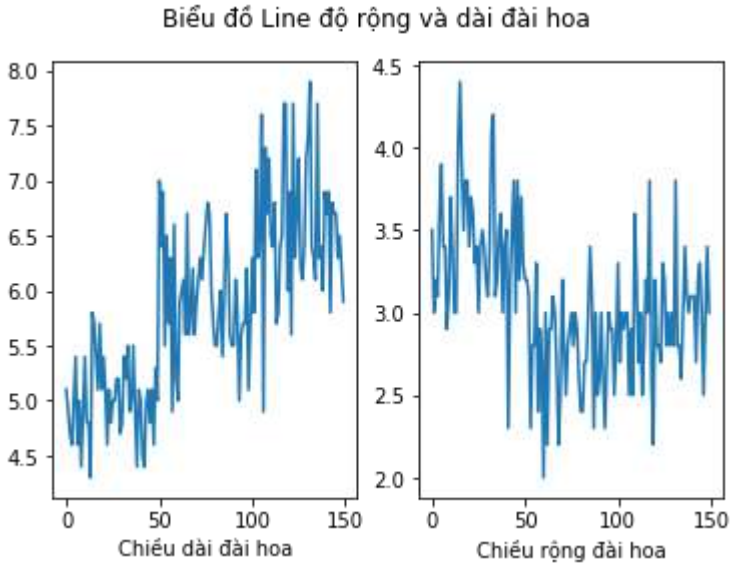
thực của trục y nên các điểm outliers được thể hiện đúng với thực tế trên từng class. Thông tin thể hiện trên biểu đồ swarm dường như là không có sự mất mát so với biểu đồ bins hoặc density.

3. Vẽ nhiều biểu đồ trên cùng 1 biểu đồ.

Matplotlib cho phép chúng ta vẽ được nhiều biểu đồ trên cùng 1 đồ thị thông qua các subplots. Chúng ta có thể xác định vị trí của subplots dựa trên việc khai báo chỉ số dòng và chỉ số cột tương tự như khai báo phần tử của ma trận.

Chẳng hạn bên dưới trên cùng 1 biểu đồ chúng ta biểu diễn độ rộng và dài của đài hoa.

```
1  import matplotlib.pyplot as plt
2  fg, ax = plt.subplots(1, 2)
3
4  ax[0].plot(dataset.iloc[:, 0])
5  ax[0].set_xlabel('Chiều dài đài hoa')
6  ax[1].plot(dataset.iloc[:, 1])
7  ax[1].set_xlabel('Chiều rộng đài hoa')
8
9  fg.suptitle('Biểu đồ Line độ rộng và dài đài hoa')
```



hàm plt.subplots() sẽ định hình hiển thị các biểu đồ con theo vị trí dòng, cột dựa trên khai báo số lượng (dòng, cột). Chẳng hạn nếu chúng ta muốn có biểu đồ gồm 2 đồ thị được hiển thị trên 1 dòng, 2 cột thì sẽ cần truyền vào là plt.subplots(1, 2) . Các tham số trả về: fg, ax lần lượt là hình dạng đồ thị (figures) và trục tọa độ (axis). Trong trường hợp có nhiều đồ thị thì ax sẽ là 1 list tương ứng các đồ thị con. Tại mỗi đồ thị con ta có thể visualize các biểu đồ theo ý muốn thông qua các hàm set_ như set_title, set_label, set_xlim, set_ylim, set_xticks, set_yticks, ...

Chúng ta có thể sử dụng biểu đồ plt.subplots() để biểu diễn hình ảnh của các nhãn trong phân loại hình ảnh.

```
from google.colab import drive
import os

drive.mount('/content/gdrive')
path = '/content/gdrive/My Drive/Colab Notebooks/visualization'
os.chdir(path)
os.listdir()
```

```
['common_pdf_shape.png',
 'kde_shape.png',
 'n01443537_9.JPEG',
 'n01443537_1.JPEG',
 'n01443537_3.JPEG',
 'n01443537_6.JPEG',
 'n01443537_2.JPEG',
 'n01443537_4.JPEG',
 'n01443537_0.JPEG',
 'n01443537_8.JPEG',
 'n01443537_5.JPEG',
 'n01443537_7.JPEG']
```

```
import matplotlib.pyplot as plt
import numpy as np
import cv2
import glob

# Sử dụng glob để lấy toàn bộ các file có extension là '.JPEG'.
images = []
for path in glob.glob('*.JPEG'):
  image = plt.imread(path)
  images.append(image)

# Khởi tạo subplot với 2 dòng 5 cột.
fg, ax = plt.subplots(2, 5, figsize=(20, 8))
fg.suptitle('Images of fish')

for i in np.arange(2):
  for j in np.arange(5):
    ax[i, j].imshow(images[i + j + j*i])
    ax[i, j].set_xlabel('Fish '+str(i+j+j*i))
```

4. Tổng kết.

Như vậy thông qua bài này chúng ta đã làm quen được với các dạng biểu đồ: Biểu đồ barchart, line, tròn, diện tích, heatmap và các dạng biểu đồ về phân phối như: Histogram, density, boxplot, swarn. Ngoài ra chúng ta cũng làm quen được cách sử dụng các packages như matplotlib, seaborn trong visualization.

Trên đây mới chỉ là những dạng biểu đồ phổ biến. Ngoài ra còn rất nhiều các biểu đồ visualize khác mà chúng ta sẽ bắt gặp khi làm việc với khoa học dữ liệu. Đồng thời các packages về visualize trong python cũng không chỉ giới hạn ở matplotlib. Một số packages khác cũng được sử dụng nhiều như: plotly, waterfall,....