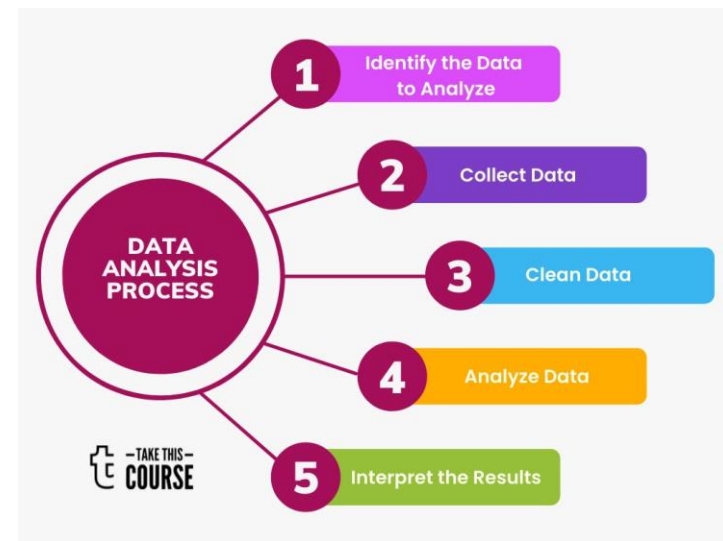


# PHÂN TÍCH DỮ LIỆU

(Data Analysis)



## THE DATA ANALYSIS PROCESS



Lê Văn Hạnh

levanhhanhvn@gmail.com

# NỘI DUNG MÔN HỌC

## PHẦN 1 TỔNG QUAN & THU THẬP DỮ LIỆU CHO VIỆC PHÂN TÍCH

1. Khoa học dữ liệu
2. Thu thập dữ liệu
3. Tìm hiểu dữ liệu

## PHẦN 2: TIỀN XỬ LÝ DỮ LIỆU (*Data Preprocessing*)

4. Nhiệm vụ chính trong tiền xử lý dữ liệu
5. PANDAS
6. Thao tác với các định dạng khác nhau của tập tin dữ liệu
7. Làm sạch và Chuẩn bị dữ liệu
8. Sắp xếp dữ liệu: nối, kết hợp và định hình lại
9. Tổng hợp dữ liệu và các tác vụ trên nhóm

## PHẦN 3 TRỰC QUAN HÓA DỮ LIỆU (*Data Visualization*)

10. Đồ thị và Biểu đồ
11. Vẽ đồ thị và Trực quan hóa

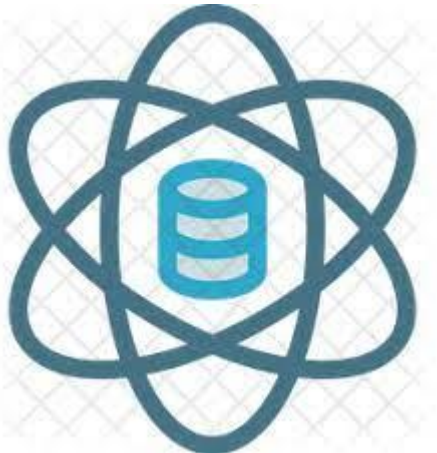
# PHẦN 3

## TRỰC QUAN HÓA DỮ LIỆU (*Data Visualization*)

Chương 11

# VẼ ĐỒ THỊ & TRỰC QUAN HÓA

(*Plotting & Visualization*)



Lê Văn Hạnh

levanhhanhvn@gmail.com

# NỘI DUNG CHƯƠNG 11

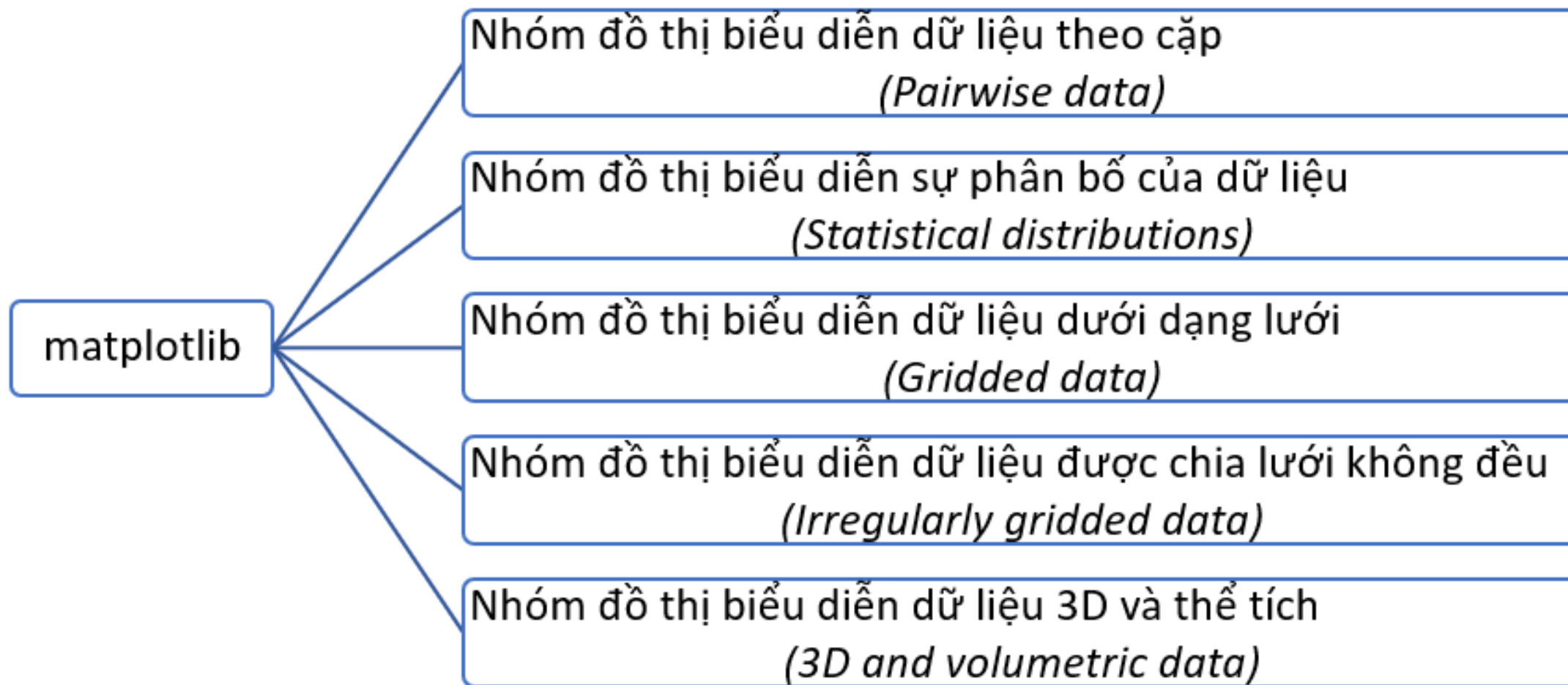
1. Matplotlib

2. Vẽ đồ thị với pandas và seaborn

# 1. Matplotlib

- *matplotlib* là gói vẽ đồ thị trên máy tính, được thiết kế để tạo các đồ thị chất lượng (chủ yếu là đồ thị hai chiều).
- Ngoại trừ một vài sơ đồ, gần như tất cả đồ họa trong tài liệu này đều có thể được tạo ra bằng cách sử dụng *matplotlib*.
- Theo thời gian, *matplotlib* đã tạo ra một số bộ công cụ hỗ trợ để trực quan hóa dữ liệu. Một trong số đó là [\*seaborn\*](#), sẽ được giới thiệu ở phần sau của chương này.
- Mặc dù các chức năng vẽ biểu đồ tích hợp của các thư viện như *seaborn* và *pandas* sẽ xử lý nhiều chi tiết bình thường của biểu đồ, nhưng nếu muốn tùy chỉnh chúng ngoài các tùy chọn chức năng được cung cấp, sẽ cần tìm hiểu một chút về *API matplotlib*.

## 1.1. Các biểu đồ được hỗ trợ trong matplotlib

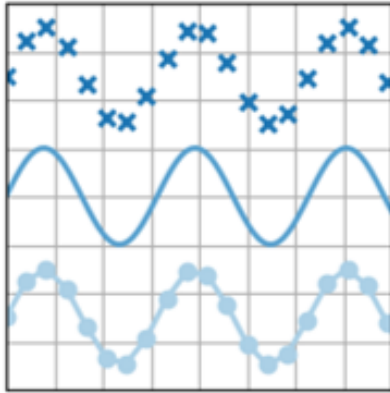


## 1. Matplotlib

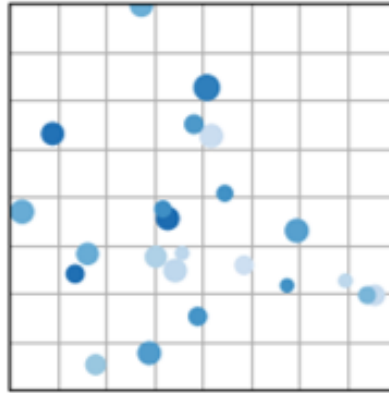
### 1.1. Các biểu đồ được hỗ trợ trong matplotlib

#### 1.1.1. Nhóm đồ thị biểu diễn dữ liệu theo cặp (Pairwise data)

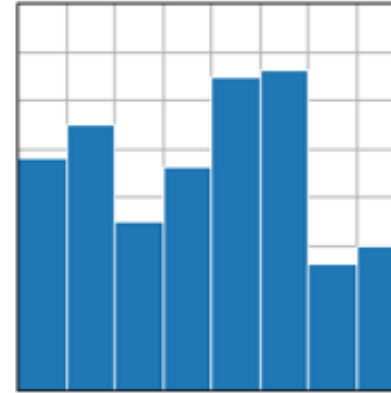
Đồ thị của dữ liệu theo cặp  $(x,y)$ , dạng bảng  $(var_0, \dots, var_n)$  và hàm  $f(x)=y$ .



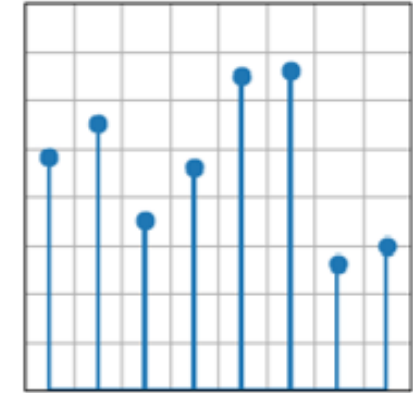
`plot(x, y)`



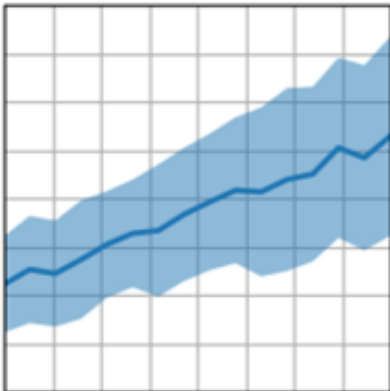
`scatter(x, y)`



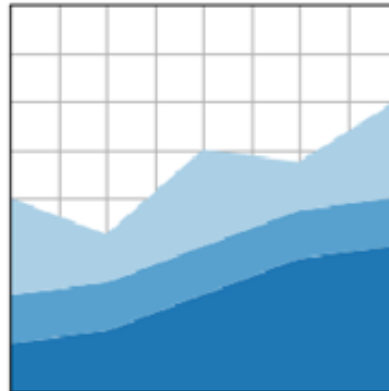
`bar(x, height)`



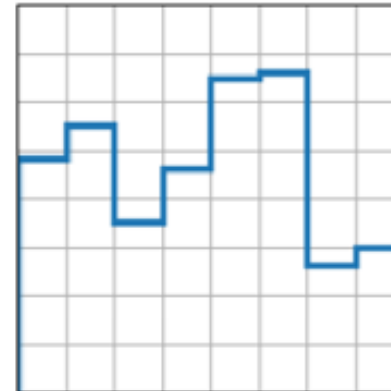
`stem(x, y)`



`fill_between(x, y1, y2)`



`stackplot(x, y)`



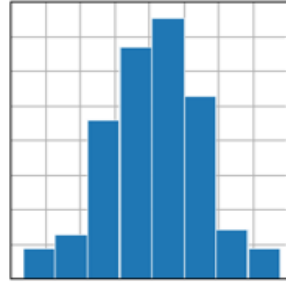
`stairs(values)`

## 1. Matplotlib

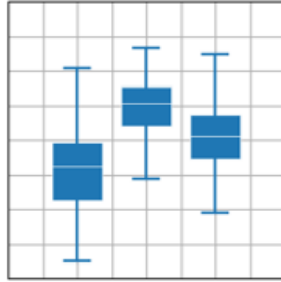
### 1.1. Các biểu đồ được hỗ trợ trong matplotlib

#### 1.1.2. Nhóm đồ thị biểu diễn sự phân bố của dữ liệu (*Statistical distributions*)

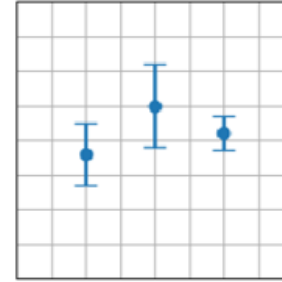
Biểu diễn đồ thị phân phối của ít nhất một biến trong tập dữ liệu. Một số đồ thị này cũng tính toán sự phân bố.



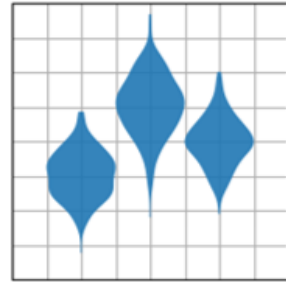
*hist(x)*



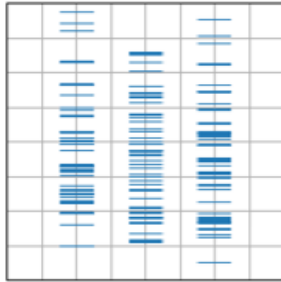
*boxplot(X)*



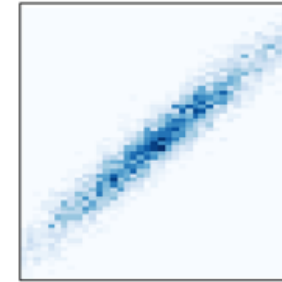
*errorbar(x, y, yerr, xerr)*



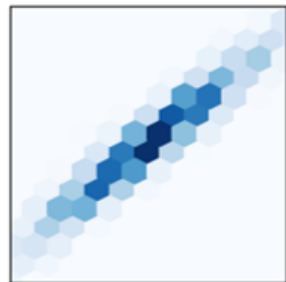
*violinplot(D)*



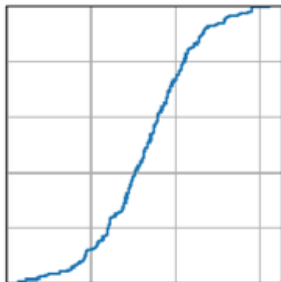
*eventplot(D)*



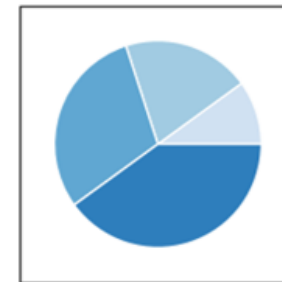
*hist2d(x, y)*



*hexbin(x, y, C)*



*ecdf(x)*



*pie(x)*

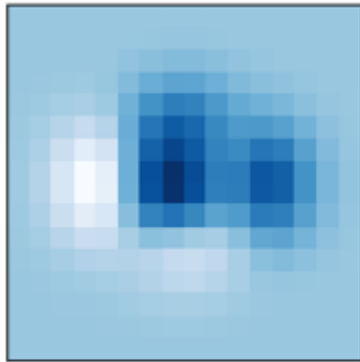


## 1. Matplotlib

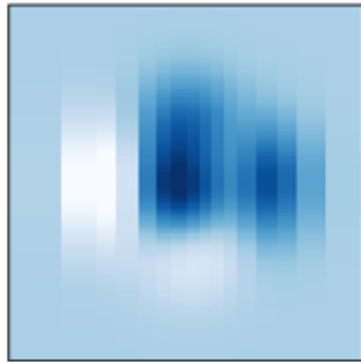
### 1.1. Các biểu đồ được hỗ trợ trong matplotlib

#### 1.1.3. Nhóm đồ thị biểu diễn dữ liệu dưới dạng lưới (Gridded data)

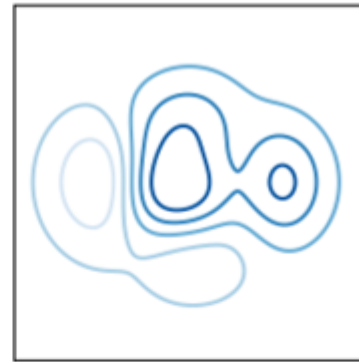
Biểu diễn đồ thị dưới dạng mảng, ảnh  $Z_{i,j}$  và các fields  $U_{i,j}, V_{i,j}$  trên lưới thông thường và lưới tọa độ tương ứng  $X_{i,j}, Y_{i,j}$ .



`imshow(Z)`



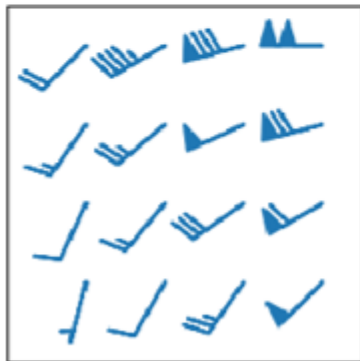
`pcolormesh(X, Y, Z)`



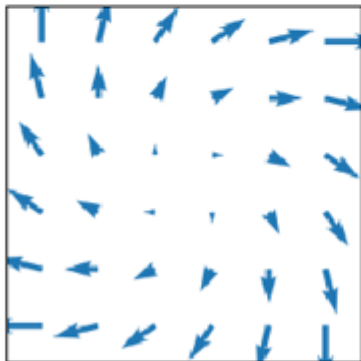
`contour(X, Y, Z)`



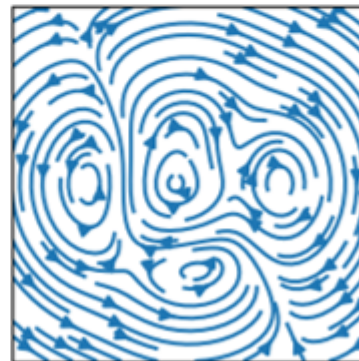
`contourf(X, Y, Z)`



`barbs(X, Y, U, V)`



`quiver(X, Y, U, V)`



`streamplot(X, Y, U, V)`

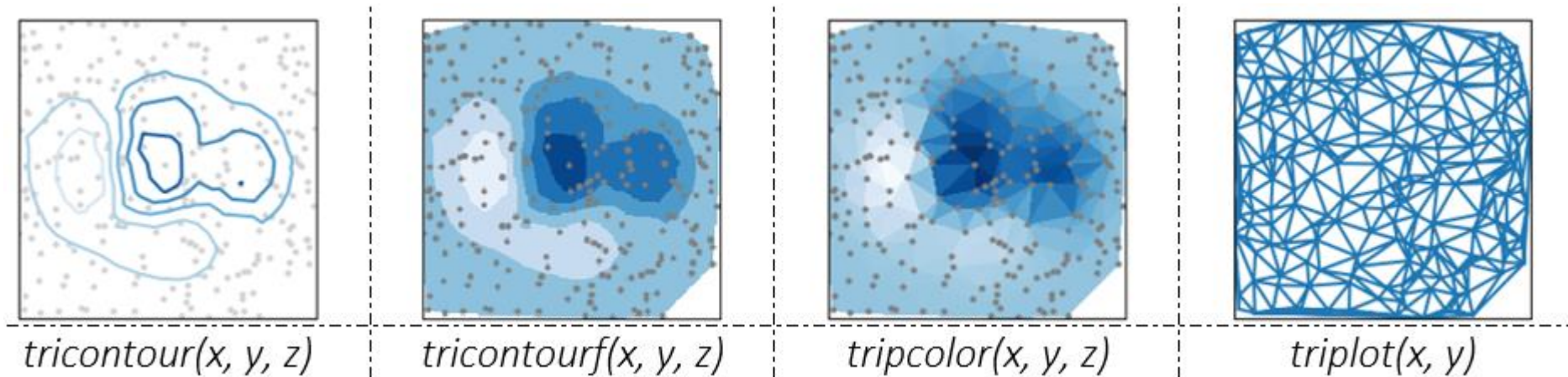
## 1. Matplotlib

### 1.1. Các biểu đồ được hỗ trợ trong matplotlib

#### 1.1.4. Nhóm đồ thị biểu diễn dữ liệu được chia lưới không đều

(Irregularly gridded data)

Biểu đồ dữ liệu  $Z_{x,y}$  trên lưới không có cấu trúc -*unstructured grids*-, lưới tọa độ không có cấu trúc  $(x, y)$  - *unstructured coordinate grids* - và các hàm 2D  $f(x, y) = z$ .

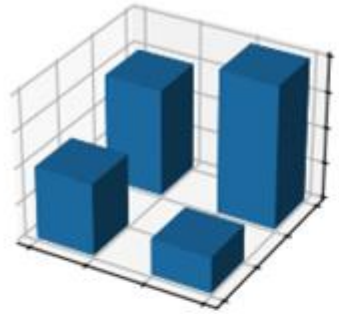


## 1. Matplotlib

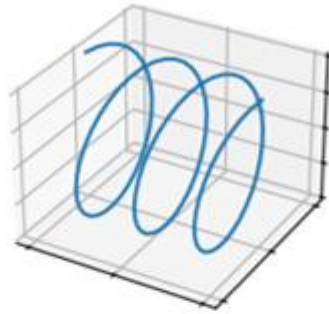
### 1.1. Các biểu đồ được hỗ trợ trong matplotlib

#### 1.1.5. Nhóm đồ thị biểu diễn dữ liệu 3D và thể tích (3D and volumetric data)

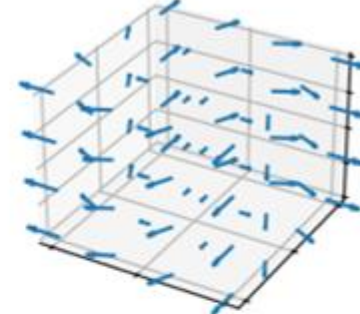
Biểu diễn đồ thị dạng ba chiều (x,y,z), bề mặt (*surface*)  $f(x,y)=z$  và dữ liệu thể tích (*volumetric*)  $V_{x,y,z}$  bằng thư viện `mpl_toolkits.mplot3d`



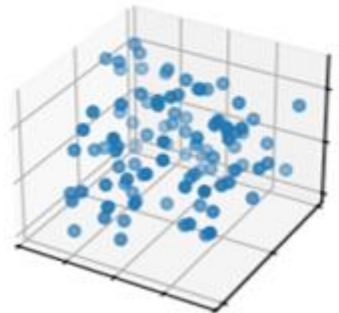
`bar3d(x, y, z, dx, dy, dz)`



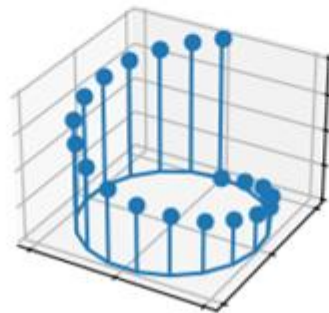
`plot(xs, ys, zs)`



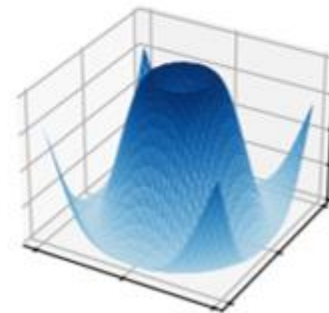
`quiver(X, Y, Z, U, V, W)`



`scatter(xs, ys, zs)`



`stem(x, y, z)`



`plot_surface(X, Y, Z)`

## 1.2. Sử dụng matplotlib

Trong tài liệu này, quy ước import *matplotlib* như sau:

```
In [1]: import matplotlib.pyplot as plt
```

Tạo một biểu đồ đơn giản:

```
In [2]: import numpy as np
```

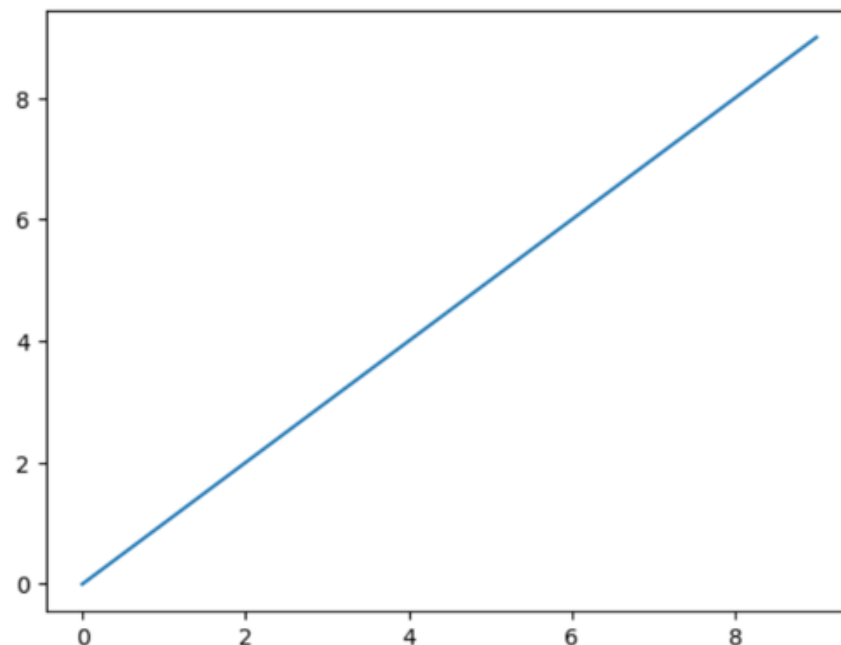
```
In [3]: data = np.arange(10)
```

```
In [4]: data
```

```
Out[4]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [5]: plt.plot(data)
```

```
Out[5]: [<matplotlib.lines.Line2D at 0x2e082b10be0>]
```



### 1.3. Hình và các ô phụ (Figures and Subplots)

- Có thể tạo một *figure* mới bằng `plt.figure`.

```
In [6]: fig = plt.figure()
```

Trong Jupyter notebook sẽ hiện ra thông tin sau:

```
<Figure size 640x480 with 0 Axes>
```

- Trong đối tượng *Figure* có thể chứa nhiều *plots* (ô).
- Không thể tạo một *plot* với một hình (*figure*) trống mà phải tạo một hoặc nhiều ô phụ bằng `add_subplot`:

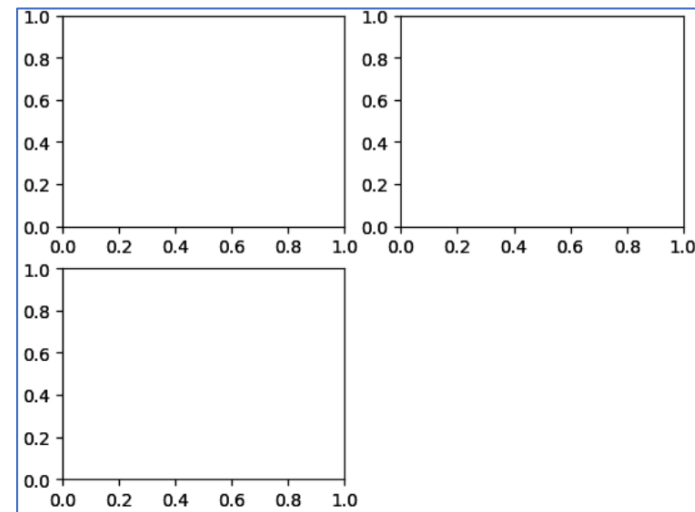
```
In [7]: ax1 = fig.add_subplot(2, 2, 1)
```

- Lệnh trên có nghĩa là *figure* phải là  $2 \times 2$  (tổng cộng có tối đa bốn ô) và `ax1` là ô đầu tiên trong số bốn ô phụ (được đánh số từ 1).

```
In [8]: ax2 = fig.add_subplot(2, 2, 2)
```

```
In [9]: ax3 = fig.add_subplot(2, 2, 3)
```

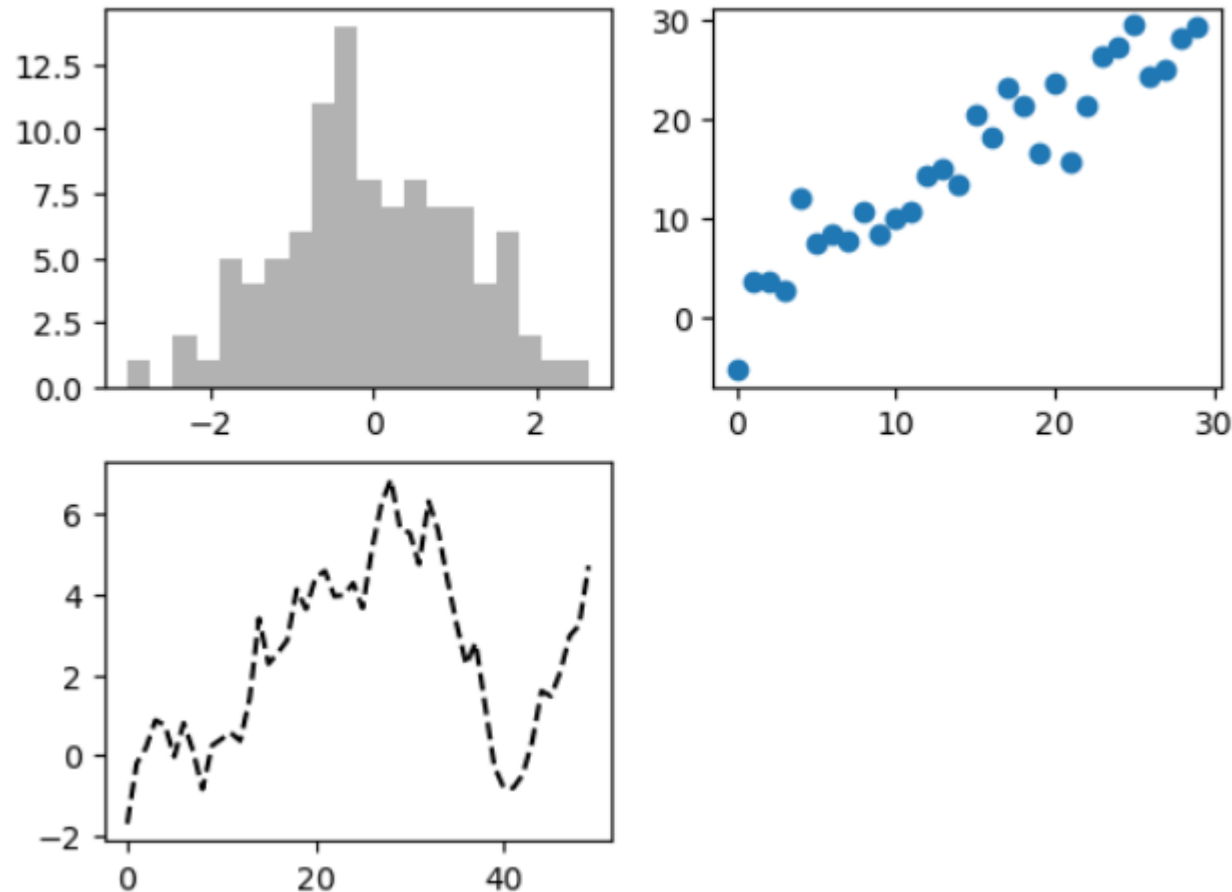
```
In [10]: fig
```



### 1.3. Hình và các ô phụ (Figures and Subplots)

- Vẽ đồ thị vào từng ô

```
In [11]: ax1.hist(np.random.randn(100), bins=20, color='k', alpha=0.3)
. . . : ax2.scatter(np.arange(30), np.arange(30) + 3 * np.random.randn(30))
. . . : ax3.plot(np.random.randn(50).cumsum(), 'k--')
. . . : fig
```





### 1.3. Hình và các ô phụ (Figures and Subplots)

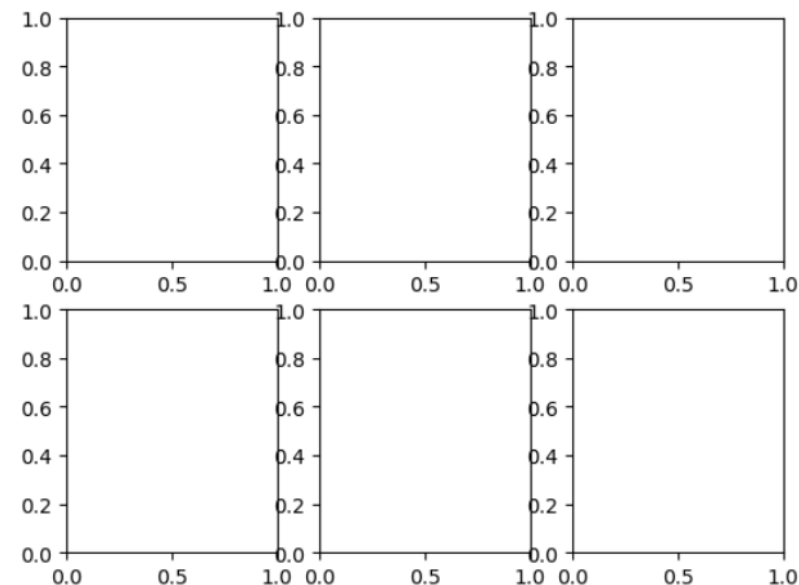
- Phương thức `plt.subplots (nrows, ncols)`:
  - Tạo một hình với một lưới các ô con.
  - Đối tượng được tạo là 1 mảng trục (*axes array*), được lập chỉ mục giống như mảng hai chiều; ví dụ: trục [0, 1].
  - Cũng có thể chỉ ra rằng các ô phụ phải có cùng trục x hoặc trục y bằng cách sử dụng `sharex` và `sharey` tương ứng. Điều này đặc biệt hữu ích khi cần so sánh dữ liệu trên cùng một tỷ lệ;
  - matplotlib tự động tính toán giới hạn của các ô một cách độc lập.

```
In [19]: fig, axes = plt.subplots(2, 3)
```

```
In [20]: axes
```

```
Out[20]:
```

```
array([[<AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>],
       [<AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>]],
      dtype=object)
```



1.3. Hình và các ô phụ (Figures and Subplots)

- Phương thức `plt.subplots (nrows, ncols)`:

Các tùy chọn của phương thức `pyplot.subplots`

Argument	Description
nrows	Số lượng dòng của <i>subplots</i>
ncols	Số lượng cột của <i>subplots</i>
sharex	Tất cả các ô phụ ( <i>subplots</i> ) phải sử dụng cùng một dấu tích trên trục x (điều chỉnh <code>xlim</code> sẽ ảnh hưởng đến tất cả các ô phụ)
sharey	Tất cả các ô phụ ( <i>subplots</i> ) phải sử dụng cùng một dấu tích trên trục y (điều chỉnh <code>ylim</code> sẽ ảnh hưởng đến tất cả các ô phụ)
subplot_kw	Dictionary của các từ khóa ( <i>dict of keywords</i> ) được chuyển đến lệnh gọi <code>add_subplot</code> để tạo từng ô phụ
**fig_kw	Các từ khóa bổ sung cho các ô phụ được sử dụng khi tạo hình, chẳng hạn như <code>plt.subplots(2, 2, figsize=(8, 6))</code>



### 1.3. Hình và các ô phụ (Figures and Subplots)

- Điều chỉnh khoảng cách xung quanh các ô phụ:

Phương thức `subplots_adjust`: cho phép thay đổi khoảng cách giữa các đối tượng *figure*:

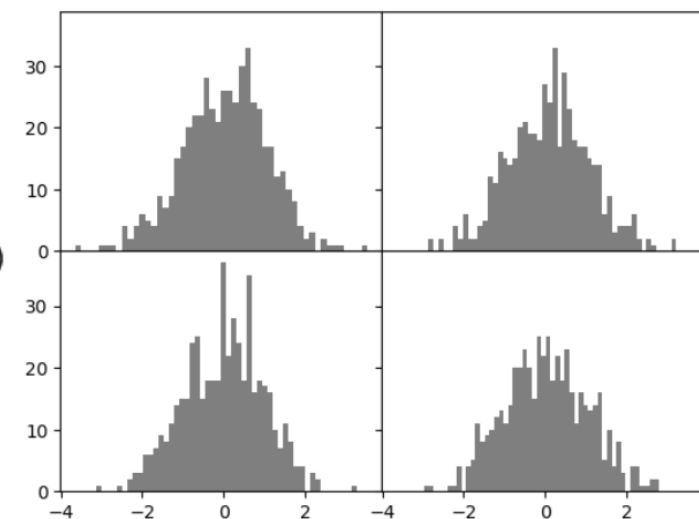
# thiết lập khoảng cách giữa các subplot về 0.

```
subplots_adjust(left=None, bottom=None, right=None, top=None,
                wspace=None, hspace=None)
```

- `wspace` và `hspace` kiểm soát chiều rộng và chiều cao tính theo phần trăm của hình tương ứng để sử dụng làm khoảng cách giữa các ô con. Ví dụ thu nhỏ khoảng cách về 0:

```
In [21]: fig, axes = plt.subplots(2, 2, sharex=True, sharey=True)
        for i in range(2):
            for j in range(2):
                axes[i,j].hist(np.random.randn(500), bins=50, color='k',
                               alpha=0.5)

        plt.subplots_adjust(wspace=0, hspace=0)
```



### 1.4. Colors, Markers, và Line Styles

- **Colors và Line**: Hàm `plot` của Matplotlib chấp nhận các mảng tọa độ  $x$  và  $y$  và tùy chọn một chữ viết tắt chuỗi biểu thị màu sắc kèm ký hiệu về kiểu đường kẻ.

Ví dụ: để vẽ  $x$  so với  $y$  bằng các dấu gạch ngang màu xanh lục, cần thực thi lệnh:

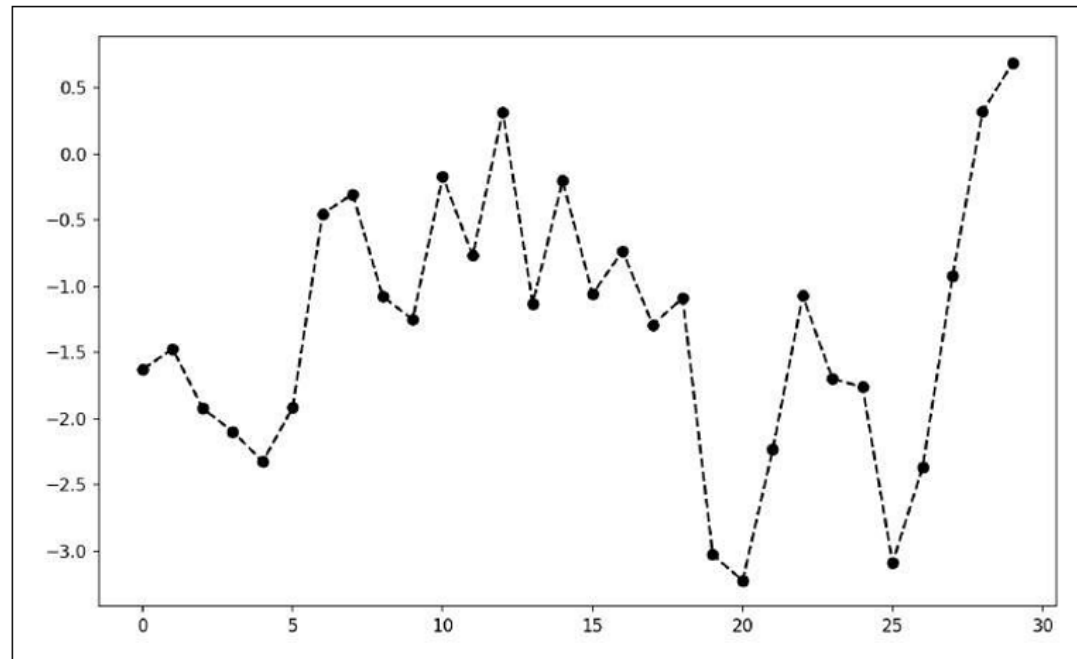
```
fig2 = plt.figure()
plt.plot([1.5, 3.5, -2, 1.6])
x=np.array(np.random.randint(100, size=5))
y=np.array(np.random.randint(1000, size=5))
plt.plot(x, y, 'g--')
# Hoặc plt.plot(x, y, linestyle='--', color='g')
fig2
```

- Có thể sử dụng bất kỳ màu nào trên quang phổ bằng cách chỉ định mã hex tương ứng (ví dụ: '#CECECE').

## 1.4. Colors, Markers, và Line Styles

- **Markers** (Điểm đánh dấu): các biểu đồ đường có thể có các điểm đánh dấu để làm nổi bật các điểm dữ liệu thực tế. Điểm đánh dấu có thể là một phần của chuỗi kiểu, phải có màu theo sau là loại điểm đánh dấu và kiểu đường:

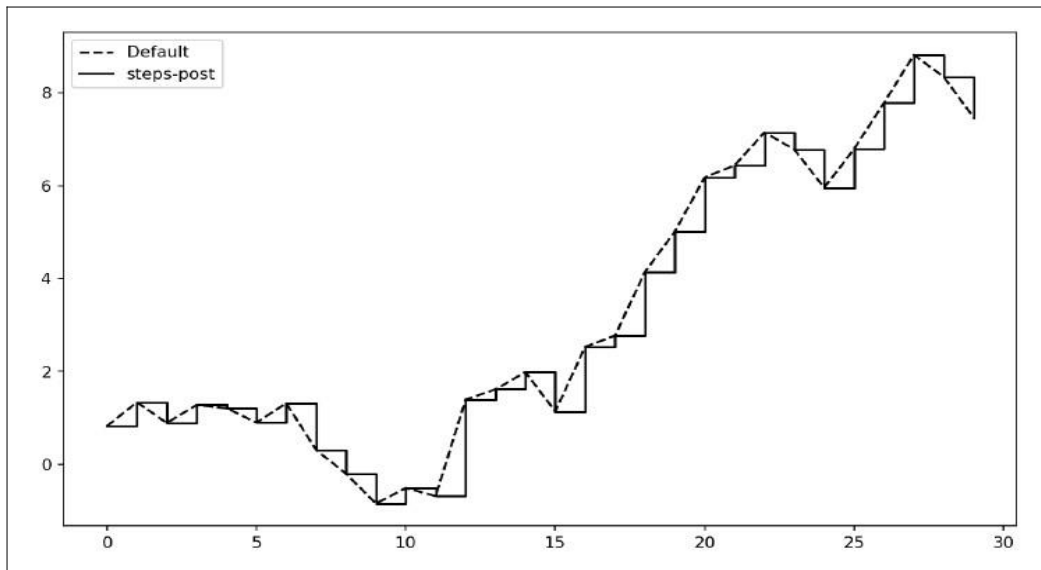
```
In [22]: %matplotlib inline
In [23]: from numpy.random import randn
In [24]: plt.plot(randn(30).cumsum(), 'ko--')
# hoặc
# plot(randn(30).cumsum(), color='k', linestyle='dashed', marker='o')
Out[24]: [<matplotlib.lines.Line2D at 0x29839550a00>]
```



### 1.4. Colors, Markers, và Line Styles

- **Markers** (Điểm đánh dấu): các biểu đồ đường có thể có các điểm đánh dấu để làm nổi bật các điểm dữ liệu thực tế. Điểm đánh dấu có thể là một phần của chuỗi kiểu, phải có màu theo sau là loại điểm đánh dấu và kiểu đường:
- Đối với biểu đồ đường, các điểm tiếp theo được nội suy tuyến tính theo mặc định. Điều này có thể được thay đổi bằng tùy chọn *drawstyle*:

```
In [25]: data = np.random.randn(30).cumsum()
In [26]: plt.plot(data, 'k--', label='Default')
In [27]: plt.plot(data, 'k-', drawstyle='steps-post', label='steps-post')
In [28]: plt.legend(loc='best')
Out[28]: <matplotlib.legend.Legend at 0x10ab570fc40>
```



**Ghi chú:** phải gọi `plt.legend` (hoặc `ax.legend`, nếu có tham chiếu đến các trục) để tạo chú giải, cho dù có hay không có bỏ qua các tùy chọn `label` khi vẽ biểu đồ dữ liệu.

1.4. Colors, Markers, và Line Styles

- Các tham số:

Ký tự đại diện cho màu	màu
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

Các tham số về nét vẽ (line styles)

character	Mô tả
'-'	Nét vẽ liền nét (solid line style)
'--'	Nét vẽ đứt nét (dashed line style)
'-.'	Nét vẽ với dấu gạch và chấm liên tục (dash-dot line style)
':'	Nét vẽ là dấu chấm (dotted line style)

Ký tự đại diện cho marker	Mô tả
'.'	point marker
','	pixel marker
'o'	circle marker
'v'	triangle_down marker
'^'	triangle_up marker
'<'	triangle_left marker
'>'	triangle_right marker
'1'	tri_down marker
'2'	tri_up marker
'3'	tri_left marker
'4'	tri_right marker
'8'	octagon marker
's'	square marker
'p'	pentagon marker
'P'	plus (filled) marker
'*'	star marker
'h'	hexagon1 marker
'H'	hexagon2 marker
'+'	plus marker
'x'	x marker
'X'	x (filled) marker
'D'	diamond marker
'd'	thin_diamond marker
' '	vline marker
'_'	hline marker

## 1.5. Ticks, Labels và Legends

- Có hai cách chính để trang trí cho biểu đồ: sử dụng thủ tục `matplotlib.pyplot` và API `matplotlib` (mang tính hướng đối tượng).
- Giao diện `pyplot`, được thiết kế để sử dụng tương tác, bao gồm các phương thức như `xlim` (phạm vi biểu đồ), `xticks` (vị trí đánh dấu) và `xticklabels` (nhãn đánh dấu). Chúng có thể được sử dụng theo hai cách:
  - Được gọi không có đối số: sẽ trả về giá trị tham số hiện tại (ví dụ: `plt.xlim()` trả về phạm vi vẽ biểu đồ trục x hiện tại)
  - Được gọi với các tham số: sẽ đặt phạm vi của biểu đồ theo giá trị tham số (ví dụ: `plt.xlim([0, 10])`, đặt phạm vi trục x từ 0 đến 10)
- Tất cả các phương thức như vậy đều hoạt động trên `AxesSubplot`. Mỗi trong số chúng tương ứng với hai phương thức trên chính đối tượng `subplot`; trong trường hợp `xlim` thì đây là `ax.get_xlim` và `ax.set_xlim`. Việc sử dụng các phương thức đối tượng của ô phụ giúp vấn đề rõ ràng (và đặc biệt là khi làm việc với nhiều ô phụ), tuy nhiên có thể sử dụng bất kỳ phương thức nào mà ta thấy thuận tiện hơn.

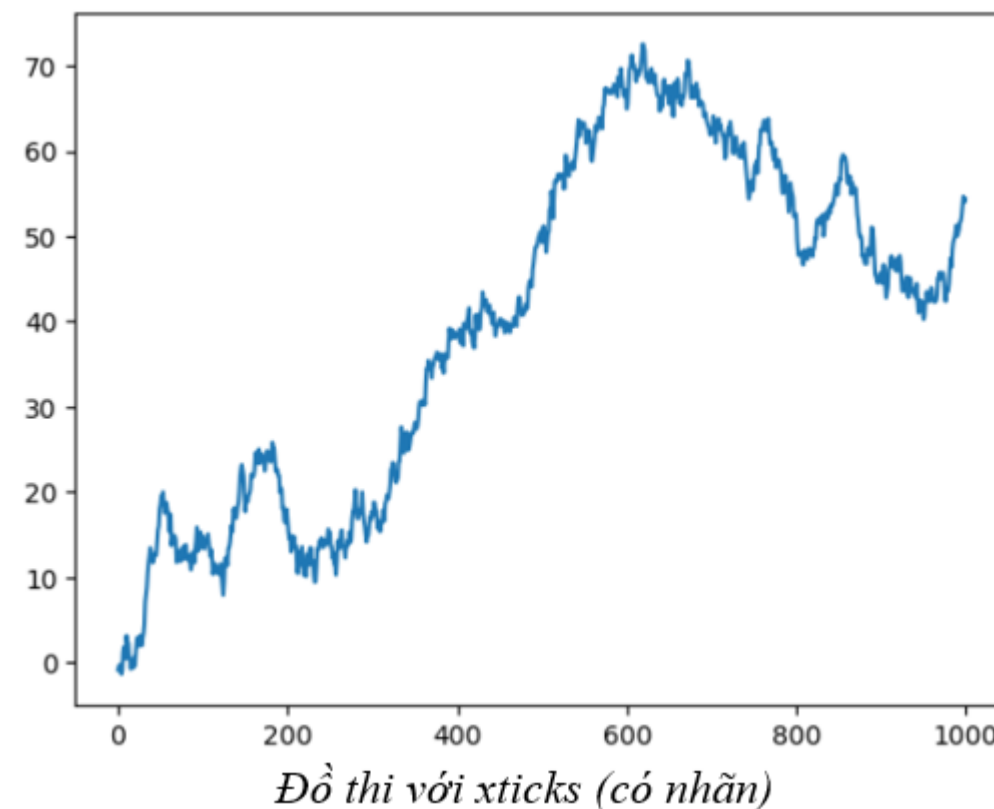
## 1. Matplotlib

### 1.5. Ticks, Labels và Legends

#### 1.5.1. Đặt tiêu đề (title), nhãn trục (axis labels), dấu tích (ticks) và nhãn đánh dấu (ticklabels)

- `set_xticks` và `set_xticklabels`: Để thay đổi các dấu tích trên trục x dọc theo phạm vi dữ liệu; theo mặc định những vị trí này cũng sẽ là nhãn.
- `set_xticklabels`: cho phép đặt đặt bất kỳ giá trị nào khác làm nhãn.

```
In [29]: fig = plt.figure()
....    : ax = fig.add_subplot(1, 1, 1)
....    : ax.plot(np.random.randn(1000).cumsum())
Out[29]: [<matplotlib.lines.Line2D at 0x10ab57c4ac0>]
```





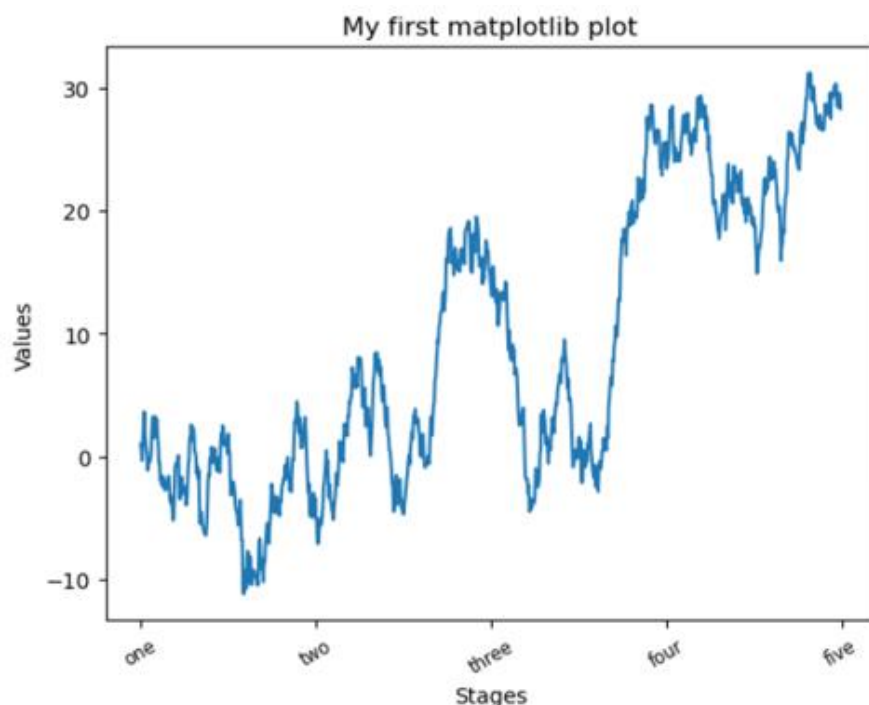
## 1. Matplotlib

### 1.5. Ticks, Labels và Legends

#### 1.5.1. Đặt tiêu đề (title), nhãn trục (axis labels), dấu tích (ticks) và nhãn đánh dấu (ticklabels)

- `set_xticks`: Để thay đổi các dấu tích trên trục x dọc theo phạm vi dữ liệu; theo mặc định những vị trí này cũng sẽ là nhãn.
- `set_xticklabels`: cho phép đặt bất kỳ giá trị nào khác làm nhãn.
- `rotation`: đặt nhãn đánh dấu x ở góc xoay 30 độ.
- `set_xlabel`: đặt tên cho trục x
- `set_title`: tiêu đề của biểu đồ

```
In [30]: fig = plt.figure()
....    : ax = fig.add_subplot(1, 1, 1)
....    : ax.plot(np.random.randn(1000).cumsum())
....    : ticks = ax.set_xticks([0, 250, 500, 750, 1000])
....    : labels = ax.set_xticklabels(['one', 'two', 'three',
....    :                               'four', 'five'], rotation=30, fontsize='small')
....    : ax.set_title('My first matplotlib plot')
....    : ax.set_xlabel('Stages')
....    : ax.set_title('My first matplotlib plot')
....    : ax.set_xlabel('Stages')
....    : ax.set_ylabel('Values')
Out [30]: Text(0.5, 0, 'Values')
```





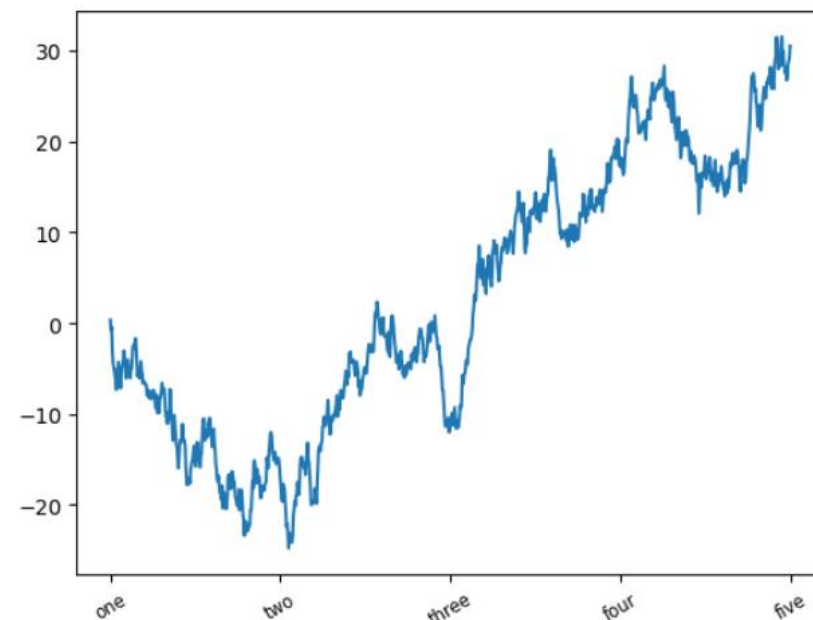
## 1. Matplotlib

### 1.5. Ticks, Labels và Legends

#### 1.5.1. Đặt tiêu đề (title), nhãn trục (axis labels), dấu tích (ticks) và nhãn đánh dấu (ticklabels)

- Lớp trục (*axes class*) có một phương thức thiết lập cho phép thiết lập hàng loạt thuộc tính biểu đồ. Từ ví dụ trước, chúng ta cũng có thể viết lại như sau:

```
In [31]: props = {'title': 'My first matplotlib plot',  
                  'xlabel': 'Stages', 'ylabel': 'Values'}  
  
.... : fig = plt.figure()  
.... : ax = fig.add_subplot(1, 1, 1)  
.... : ax.plot(np.random.randn(1000).cumsum())  
.... : ticks = ax.set_xticks([0, 250, 500, 750, 1000])  
.... : labels = ax.set_xticklabels(['one', 'two', 'three', 'four',  
                                   'five'], rotation=30, fontsize='small')  
  
.... : #ax.set_title('My first matplotlib plot')  
.... : #ax.set_xlabel('Stages')  
.... : #ax.set_ylabel('Values')  
.... : ax.set(**props)  
Out[31]: [Text(0.5, 1.0, 'My first matplotlib plot'),  
          Text(0.5, 0, 'Stages'),  
          Text(0, 0.5, 'Values')]
```



## 1. Matplotlib

### 1.5. Ticks, Labels và Legends

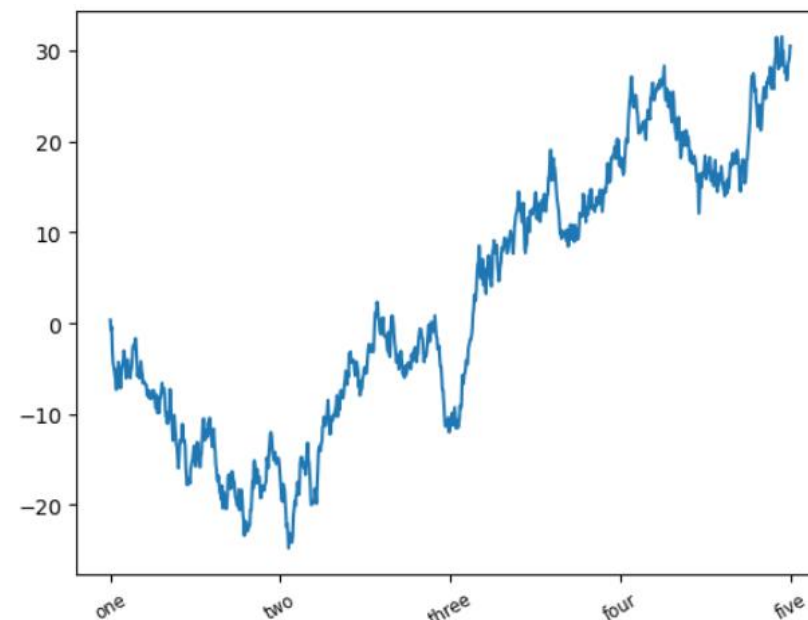
#### 1.5.1. Đặt tiêu đề (title), nhãn trục (axis labels), dấu tích (ticks) và nhãn đánh dấu (ticklabels)

- Lớp trục (*axes class*) có một phương thức thiết lập cho phép thiết lập hàng loạt thuộc tính biểu đồ. Từ ví dụ trước, chúng ta cũng có thể viết lại như sau:

```
In [31]: props = {'title': 'My first matplotlib plot',
                  'xlabel': 'Stages', 'ylabel': 'Values'}

.... : fig = plt.figure()
.... : ax = fig.add_subplot(1, 1, 1)
.... : ax.plot(np.random.randn(1000).cumsum())
.... : ticks = ax.set_xticks([0, 250, 500, 750, 1000])
.... : labels = ax.set_xticklabels(['one', 'two', 'three', 'four',
                                   'five'], rotation=30, fontsize='small')

.... : #ax.set_title('My first matplotlib plot')
.... : #ax.set_xlabel('Stages')
.... : #ax.set_ylabel('Values')
.... : ax.set(**props)
Out[31]: [Text(0.5, 1.0, 'My first matplotlib plot'),
          Text(0.5, 0, 'Stages'),
          Text(0, 0.5, 'Values')]
```



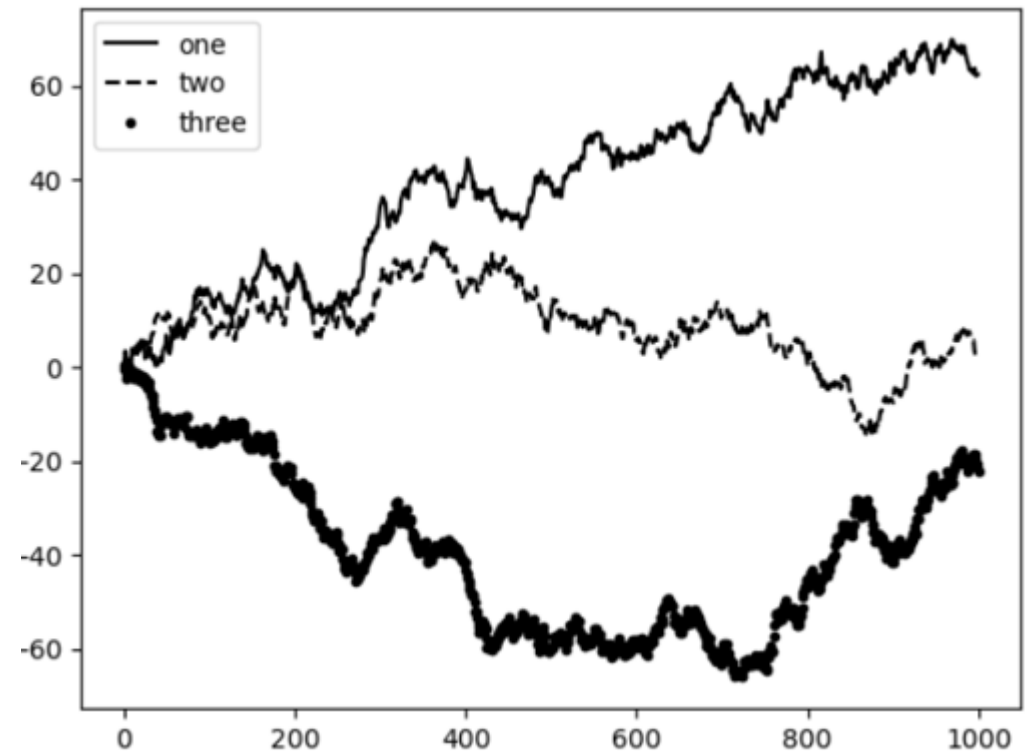
## 1. Matplotlib

### 1.5. Ticks, Labels và Legends

#### 1.5.2. Thêm chú thích (Adding legends)

- Phương thức `legend` có một số lựa chọn khác cho đối số vị trí `loc`. Với các giá trị có thể dùng cho `loc` là: `'best'`, `'upper right'`, `'upper left'`, `'lower left'`, `'lower right'`, `'right'`, `'center left'`, `'center right'`, `'lower center'`, `'upper center'`, `'center'`.

```
In [32]: from numpy.random import randn
In [33]: fig = plt.figure(); ax = fig.add_subplot(1, 1, 1)
In [34]: ax.plot(randn(1000).cumsum(), 'k', label='one')
Out[34]: [<matplotlib.lines.Line2D at 0x7fb624bdf860>]
In [35]: ax.plot(randn(1000).cumsum(), 'k--', label='two')
Out[35]: [<matplotlib.lines.Line2D at 0x7fb624be90f0>]
In [36]: ax.plot(randn(1000).cumsum(), 'k.', label='three')
Out[36]: [<matplotlib.lines.Line2D at 0x7fb624be9160>]
In [37]: ax.legend(loc='best')
Out[37]: <matplotlib.legend.Legend at 0x10ab5819100>
In [38]: fig
Out[38]:
```



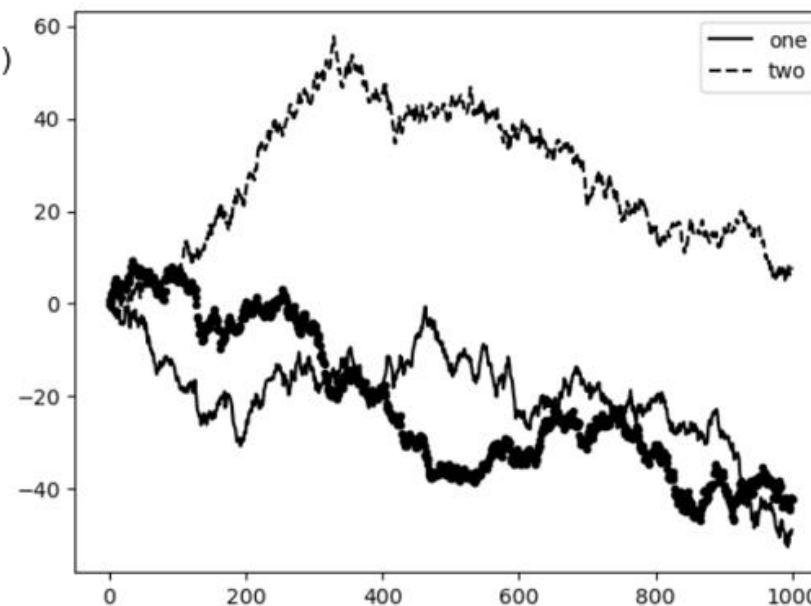
## 1. Matplotlib

### 1.5. Ticks, Labels và Legends

#### 1.5.2. Thêm chú thích (Adding legends)

- Đối số `loc` cho `matplotlib` biết nơi đặt chú thích. Trong đó, giá trị `'best'` là một lựa chọn tốt, vì nó sẽ chọn một vị trí trống nhất trong biểu đồ để đặt chú thích.
- Để loại trừ một hoặc nhiều phần tử khỏi chú thích, sử dụng `label='_nolegend_'` khi tạo biểu đồ.

```
In [39]: from numpy.random import randn
. . . .: fig = plt.figure(); ax = fig.add_subplot(1, 1, 1)
. . . .: ax.plot(randn(1000).cumsum(), 'k', label='one')
. . . .: ax.plot(randn(1000).cumsum(), 'k--', label='two')
. . . .: ax.plot(randn(1000).cumsum(), 'k.', label='_nolegend_')
Out[39]: <matplotlib.legend.Legend at 0x10ab6583640>
```



Biểu đồ gồm 3 đồ thị con, nhưng chỉ 2 đồ thị trong số đó là có chú thích

## 1.6. Phương thức text

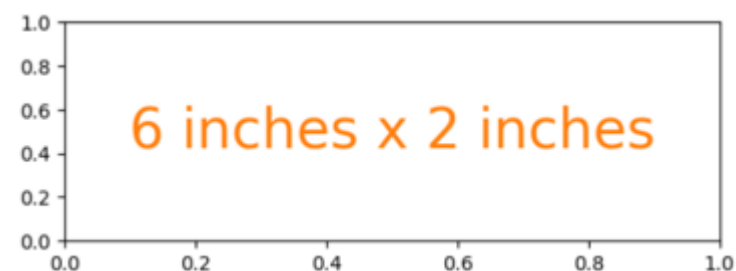
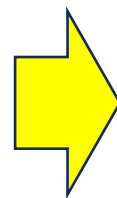
- Phương thức `text` vẽ văn bản tại tọa độ nhất định (x, y) trên biểu đồ với kiểu tùy chọn.

## 1.7. Tham số figsize của subplots

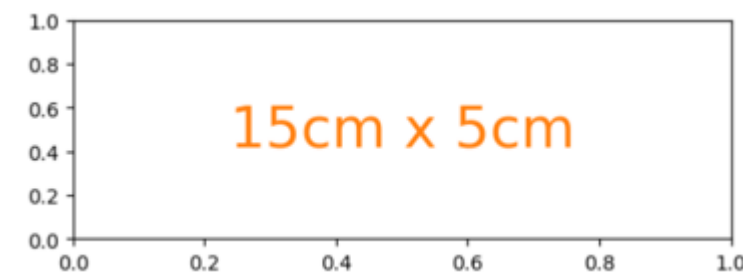
- Đơn vị đo kích thước trong Matplotlib là inch (mặc định), cm hoặc pixel.

```
In [40]: # Figure size in inches (default)
. . . .: import matplotlib.pyplot as plt
. . . .: text_kwargs = dict(ha='center', va='center', fontsize=28,
. . . .:                               color='C1')

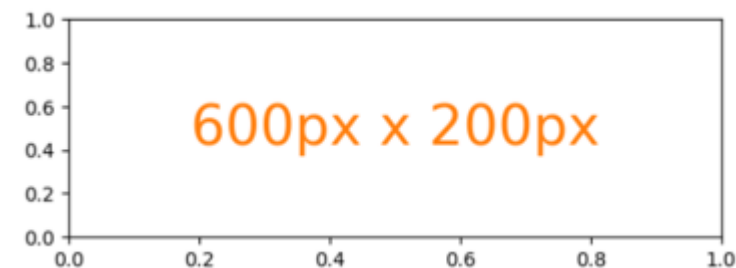
In [41]: plt.subplots(figsize=(6, 2))
. . . .: plt.text(0.5, 0.5, '6 inches x 2 inches', **text_kwargs)
. . . .: plt.show()
```



```
In [42]: # Figure size in centimeters
. . . .: cm = 1/2.54 # define variable cm, centimeters in inches
. . . .: plt.subplots(figsize=(15*cm, 5*cm))
. . . .: plt.text(0.5, 0.5, '15cm x 5cm', **text_kwargs)
. . . .: plt.show()
```



```
In [43]: # Figure size in pixel
. . . .: px = 1/plt.rcParams['figure.dpi'] # pixel in inches
. . . .: plt.subplots(figsize=(600*px, 200*px))
. . . .: plt.text(0.5, 0.5, '600px x 200px', **text_kwargs)
. . . .: plt.show()
```



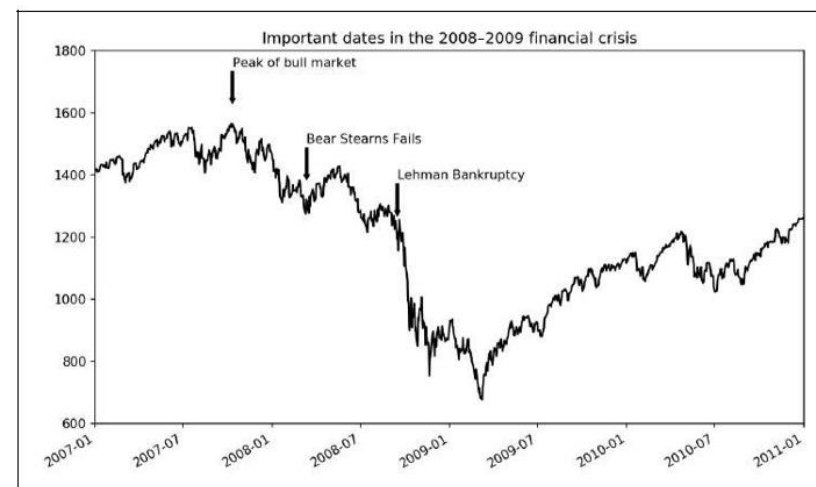
## 1.8. Bổ sung chú thích và hình vẽ vào biểu đồ con (Annotations and Drawing on a Subplot)

### 1.8.1. Chú thích với văn bản và mũi tên đi kèm

- Ví dụ: vẽ biểu đồ giá đóng cửa của chỉ số S&P 500 kể từ năm 2007 (dữ liệu thu được từ Yahoo! Finance) và chú thích nó bằng một số ngày quan trọng từ cuộc khủng hoảng tài chính 2008–2009):

```
In [44]: from datetime import datetime
. . . .: fig = plt.figure()
. . . .: ax = fig.add_subplot(1, 1, 1)
. . . .: data = pd.read_csv('examples/spx.csv', index_col=0,
. . . .:                                     parse_dates=True)

. . . .: spx = data['SPX']
. . . .: spx.plot(ax=ax, style='k-')
. . . .: crisis_data = [(datetime(2007, 10, 11), 'Peak of bull market'),
. . . .:                (datetime(2008, 3, 12), 'Bear Stearns Fails'),
. . . .:                (datetime(2008, 9, 15), 'Lehman Bankruptcy')]
. . . .:
. . . .: for date, label in crisis_data:
. . . .:     ax.annotate(label, xy=(date, spx.asof(date) + 75),
. . . .:                xytext=(date, spx.asof(date) + 225),
. . . .:                arrowprops=dict(facecolor='black', headwidth=4,
. . . .:                width=2, headlength=4), horizontalalignment='left',
. . . .:                verticalalignment='top')
. . . .: # Zoom in on 2007–2010
. . . .: ax.set_xlim(['1/1/2007', '1/1/2011'])
. . . .: ax.set_ylim([600, 1800])
. . . .: ax.set_title('Important dates in the 2008–2009 financial crisis')
```



Những ngày quan trọng trong cuộc khủng hoảng tài chính 2008–2009



## 1. Matplotlib

### 1.8. Bổ sung chú thích và hình vẽ vào biểu đồ con (Annotations and Drawing on a Subplot)

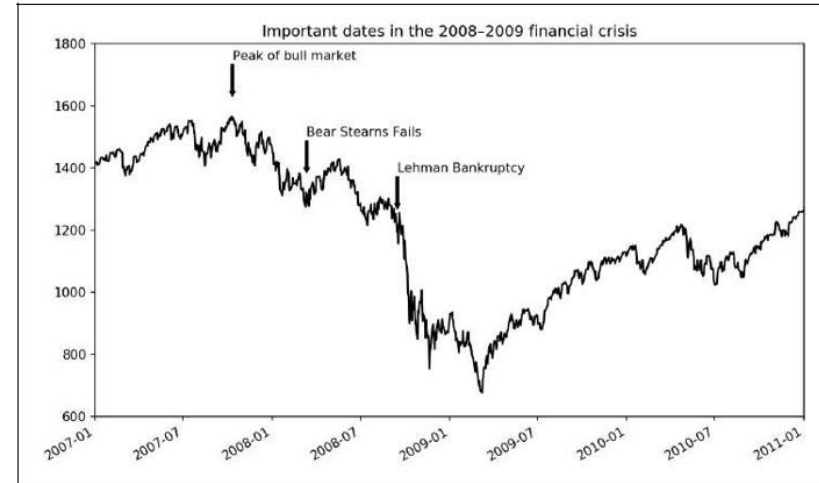
#### 1.8.1. Chú thích với văn bản và mũi tên đi kèm

- Ví dụ:

- Phương thức `ax.annotate`: vẽ nhãn tại tọa độ x và y được chỉ định.
- Phương thức `set_xlim` và `set_ylim` để đặt ranh giới bắt đầu và kết thúc cho biểu đồ theo cách thủ công thay vì sử dụng mặc định của `matplotlib`.
- Phương thức `ax.set_title` thêm tiêu đề chính vào biểu đồ

```
In [44]: from datetime import datetime
...: fig = plt.figure()
...: ax = fig.add_subplot(1, 1, 1)
...: data = pd.read_csv('examples/spx.csv', index_col=0,
...:                                     parse_dates=True)

...: spx = data['SPX']
...: spx.plot(ax=ax, style='k-')
...: crisis_data = [(datetime(2007, 10, 11), 'Peak of bull market'),
...:                (datetime(2008, 3, 12), 'Bear Stearns Fails'),
...:                (datetime(2008, 9, 15), 'Lehman Bankruptcy')]
...: for date, label in crisis_data:
...:     ax.annotate(label, xy=(date, spx.asof(date) + 75),
...:                 xytext=(date, spx.asof(date) + 225),
...:                 arrowprops=dict(facecolor='black', headwidth=4,
...:                 width=2, headlength=4), horizontalalignment='left',
...:                 verticalalignment='top')
...: # Zoom in on 2007-2010
...: ax.set_xlim(['1/1/2007', '1/1/2011'])
...: ax.set_ylim([600, 1800])
...: ax.set_title('Important dates in the 2008-2009 financial crisis')
```



Những ngày quan trọng trong cuộc khủng hoảng tài chính 2008-2009

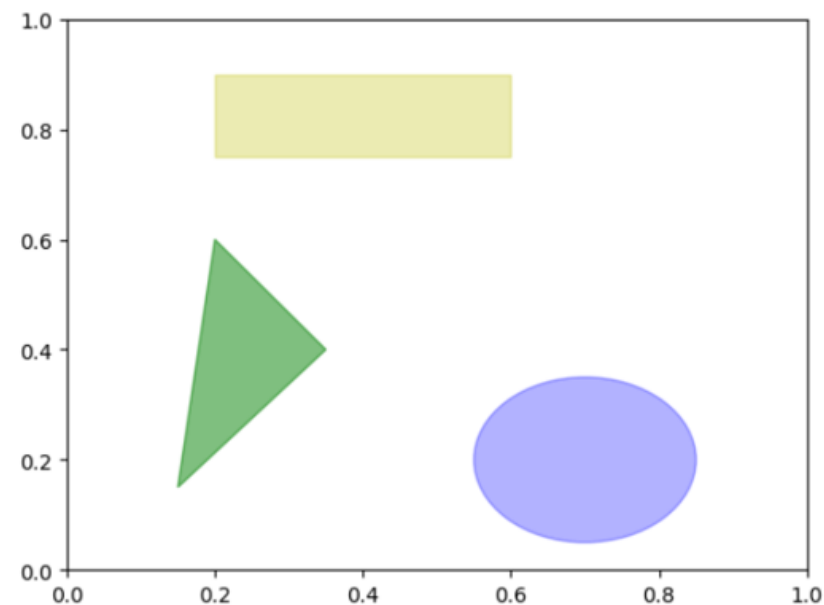
## 1. Matplotlib

### 1.8. Bổ sung chú thích và hình vẽ vào biểu đồ con (Annotations and Drawing on a Subplot)

#### 1.8.2. Vẽ các hình học

- Vẽ hình đòi hỏi phải cẩn thận hơn. *matplotlib* có các đối tượng đại diện cho nhiều hình dạng phổ biến, được gọi là các *patches*. Một số trong số này, như Hình chữ nhật và Hình tròn, được tìm thấy trong *matplotlib.pyplot*, nhưng bộ đầy đủ nằm trong *matplotlib.patches*.
- Để thêm một hình dạng vào một ô, bạn tạo đối tượng *patch shp* và thêm nó vào ô con bằng cách gọi *ax.add\_patch(shp)*

```
In [45]: fig = plt.figure() ax = fig.add_subplot(1, 1, 1)
. . . .: rect = plt.Rectangle((0.2, 0.75), 0.4, 0.15, color='k',
                                alpha=0.3)
. . . .: circ = plt.Circle((0.7, 0.2), 0.15, color='b', alpha=0.3)
. . . .: pgon = plt.Polygon([[0.15, 0.15], [0.35, 0.4], [0.2, 0.6]],
                                color='g', alpha=0.5)
. . . .: ax.add_patch(rect)
. . . .: ax.add_patch(circ)
. . . .: ax.add_patch(pgon)
In [46]: <matplotlib.patches.Polygon at 0x10b11426190>
```





## 1.9. Lưu biểu đồ vào file (Saving Plots to File)

- Phương thức `plt.savefig`:

- Có thể lưu hình (*figure*) vào file. Phương thức này tương đương với phương thức `savefig` của đối tượng *figure*.

Ví dụ: để lưu phiên bản *SVG* của một hình, sử dụng lệnh: `plt.savefig('figpath.svg')`

- Loại của file được suy ra từ phần mở rộng file. Vì vậy, nếu sử dụng *.pdf* thay vào đó, ta sẽ nhận được bản *PDF*. Có một số tùy chọn quan trọng được sử dụng thường xuyên để xuất bản đồ họa:

- *dpi*: giúp kiểm soát độ phân giải số trên mỗi inch
- *bbox\_inches*: có thể cắt khoảng trắng xung quanh hình thực tế.

Ví dụ: Để có được biểu đồ giống như *PNG* với khoảng trắng tối thiểu xung quanh biểu đồ và ở độ phân giải 400 dpi, có thể thực hiện lệnh sau:

```
plt.savefig('figpath.png', dpi=400, bbox_inches='tight')
```

### 1.9. Lưu biểu đồ vào file (Saving Plots to File)

- savefig* cũng có thể ghi vào bất kỳ đối tượng nào giống file, chẳng hạn như *BytesIO*:

```
from io import BytesIO
buffer = BytesIO()
plt.savefig(buffer)
plot_data = buffer.getvalue()
```

- Các tùy chọn của *plt.savefig*:

Các tùy chọn của *savefig*

Argument	Description
fname	Chuỗi chứa đường dẫn của file hoặc đối tượng giống file trong Python. Định dạng của hình được suy ra từ phần mở rộng của file (ví dụ: .pdf cho PDF hoặc .png cho PNG).
dpi	Độ phân giải hình tính bằng số chấm trên mỗi inch; mặc định là 100.
facecolor, edgecolor	Màu nền của hình bên ngoài ô phụ; 'w' (màu trắng), theo mặc định.
format	Định dạng file rõ ràng cần sử dụng ('png', 'pdf', 'svg', 'ps', 'eps', ...).
bbox_inches	Phần hình cần lưu; nếu là 'tight', sẽ cố gắng cắt bớt khoảng trống xung quanh hình.

### 1.10. Cấu hình *matplotlib*

- Phương thức *rc* trong *matplotlib* giúp tùy chỉnh cấu hình thông qua một tập hợp mở rộng các tham số chung chi phối kích thước hình, khoảng cách ô phụ, màu sắc, kích thước phông chữ, kiểu lưới, v.v.

Ví dụ: để đặt kích thước hình mặc định chung là  $10 \times 10$ , bạn có thể nhập: `plt.rc('figure', figsize=(10, 10))`


- Đối số đầu tiên của *rc* là thành phần bạn muốn tùy chỉnh, chẳng hạn như *'figure'*, *'axes'*, *'xtick'*, *'ytick'*, *'grid'*, *'legend'* hoặc nhiều thành phần khác.
- Sau đó có thể làm theo một chuỗi các đối số từ khóa chỉ ra các tham số mới. Một cách dễ dàng để viết ra các tùy chọn trong chương trình là tạo ra 1 dictionary chứa các chỉ định mà chương trình muốn dùng:

```
font_options = {'family': 'monospace', 'weight': 'bold', 'size': 'small'}  
plt.rc('font', **font_options)
```

- Để tùy chỉnh rộng rãi hơn và xem danh sách tất cả các tùy chọn, *matplotlib* đi kèm với tập tin cấu hình *matplotlibrc* trong thư mục *matplotlib/mpl-data*. Nếu tập tin này đã được tùy chỉnh và đặt nó vào thư mục chính có tiêu đề *.matplotlibrc*, nó sẽ được tải mỗi khi sử dụng *matplotlib*.

## 2. VẼ ĐỒ THỊ VỚI *pandas* & *seaborn*

- *matplotlib* có thể là một công cụ cấp độ khá thấp gồm tập hợp một biểu đồ từ các thành phần cơ bản của nó: hiển thị dữ liệu (tức là loại biểu đồ: *line*, *bar*, *box*, *scatter*, *contour*, v.v.), chú thích, tiêu đề, nhãn đánh dấu và các chú thích khác.
- Trong *pandas*, có thể có nhiều cột dữ liệu, cùng với nhãn của hàng (*row labels*) và nhãn của cột (*column labels*). Bản thân *pandas* có các phương thức tích hợp giúp đơn giản hóa việc tạo trực quan hóa từ các đối tượng *DataFrame* và *Series*. Một thư viện khác là *seaborn*, một thư viện đồ họa thống kê được tạo bởi Michael Waskom. *Seaborn* đơn giản hóa việc tạo ra nhiều kiểu hình ảnh phổ biến.

 **Gợi ý:** Import *seaborn* sẽ sửa đổi cách phối màu và kiểu biểu đồ *matplotlib* mặc định để cải thiện khả năng đọc và tính thẩm mỹ. Ngay cả khi không sử dụng *API seaborn*, ta vẫn có thể *import seaborn* như một cách đơn giản để cải thiện tính thẩm mỹ trực quan của các biểu đồ *matplotlib*.

## 2.1. Biểu đồ đường (*Line plots*)

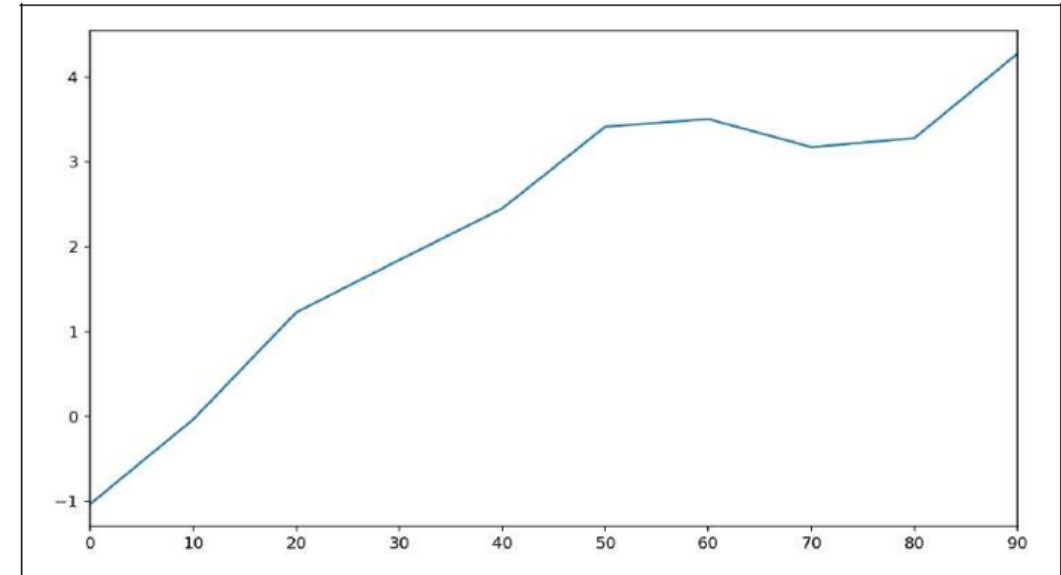
- *matplotlib* có thể là một công cụ cấp độ khá thấp gồm tập hợp một biểu đồ từ các thành phần cơ bản của nó: hiển thị dữ liệu (tức là loại biểu đồ: *line*, *bar*, *box*, *scatter*, *contour*, v.v.), chú thích, tiêu đề, nhãn đánh dấu và các chú thích khác.
- Mỗi *Series* và *DataFrame* đều có thuộc tính *plot* để tạo một số loại biểu đồ cơ bản. Theo mặc định, hàm *plot()* tạo các biểu đồ *line* (tức là *df.plot()* tương đương với *df.plot.line()*)
- Chỉ mục của đối tượng *Series* được chuyển tới *matplotlib* để vẽ sơ đồ trên trục *x*, mặc dù có thể vô hiệu hóa điều này bằng cách truyền *use\_index=False*. Các dấu tích (*ticks*) và giới hạn (*limits*) trên trục *x* có thể được điều chỉnh bằng các tùy chọn *xticks* và *xlim*, và trục *y* tương ứng với *yticks* và *ylim*.

## 2. Vẽ đồ thị với *pandas* & *seaborn*

### 2.1. Biểu đồ đường (Line plots)

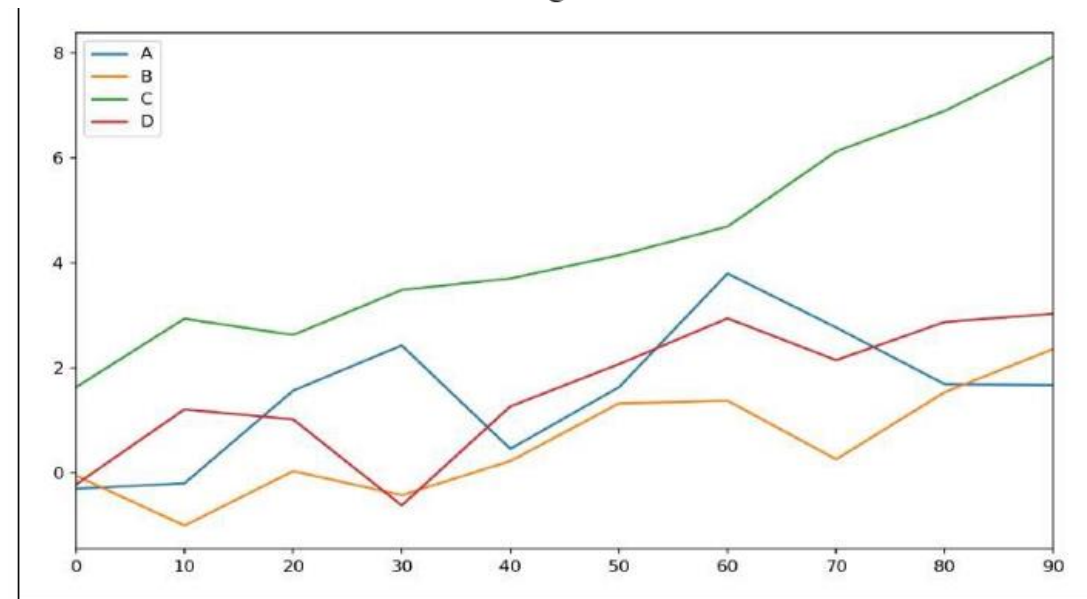
#### - Minh họa

```
In [47]: s = pd.Series(np.random.randn(10).cumsum(),  
                      index=np.arange(0, 100, 10))  
  
In [48]: s.plot()
```



Biểu đồ đơn giản về Series

```
In [49]: df = pd.DataFrame(np.random.randn(10, 4).cumsum(0),  
.....: columns=['A', 'B', 'C', 'D'],  
.....: index=np.arange(0, 100, 10))  
In [50]: df.plot()
```



Biểu đồ DataFrame đơn giản

2.1. Biểu đồ đường (Line plots)

- Danh sách các đối số của phương thức *Series.plot* trong việc vẽ biểu đồ.

Argument	Description
label	Nhãn cho chú thích của biểu đồ
ax	đối tượng <i>subplot matplotlib</i> để vẽ biểu đồ; nếu bỏ qua, sẽ sử dụng <i>subplot matplotlib</i> đang hoạt động
style	Chuỗi kiểu, như 'k--', được chuyển tới <i>matplotlib</i>
alpha	Độ mờ của biểu đồ (từ 0 đến 1)
kind	Có thể là 'area', 'bar', 'barh', 'density', 'hist', 'kde', 'line', 'pie'
logy	Sử dụng tỷ lệ logarit trên trục y
use_index	Sử dụng đối tượng chỉ mục cho nhãn đánh dấu
rot	Xoay ( <i>rotation</i> ) nhãn đánh dấu (từ 0 đến 360 độ)
xticks	Các giá trị sử dụng cho dấu tích trục x
yticks	Các giá trị sử dụng cho dấu tích trục y
xlim	Giá trị giới hạn cho trục x (ví dụ: [0, 10])
ylim	Giá trị giới hạn cho trục y
grid	Hiển thị lưới (mặc định)



2.1. Biểu đồ đường (Line plots)

- Danh sách các đối số của phương thức *DataFrame.plot* trong việc vẽ biểu đồ.

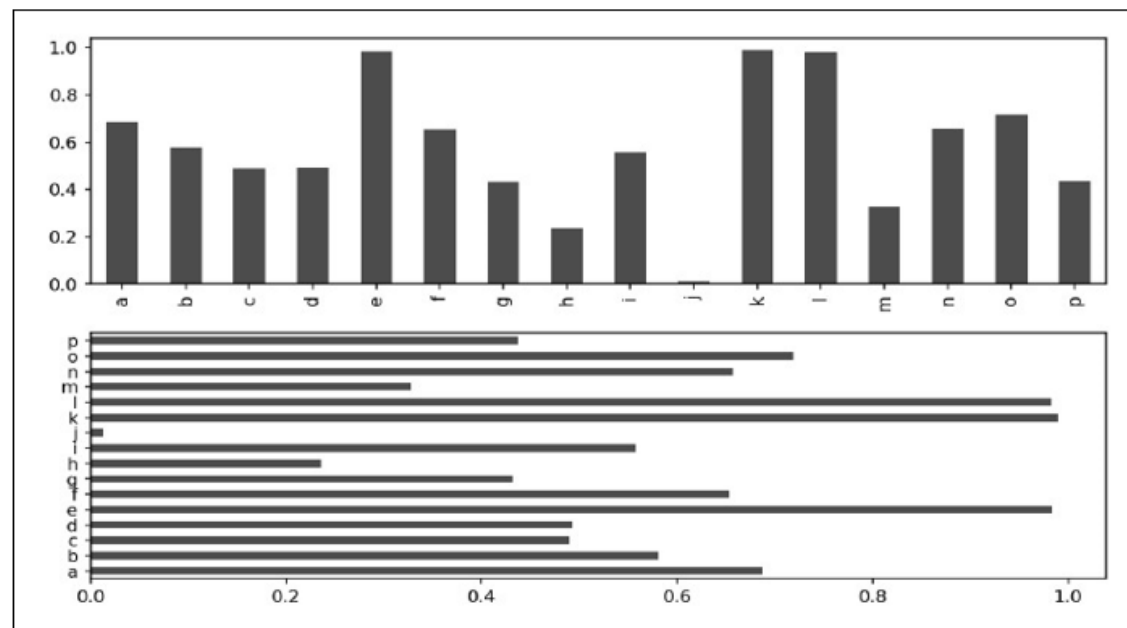
<i>Argument</i>	<i>Description</i>
subplots	Plot each DataFrame column in a separate subplot Vẽ từng cột <i>DataFrame</i> trong một ô con riêng biệt
sharex	Nếu <code>subplots=True</code> , chia sẻ cùng trục x, liên kết các dấu tích và giới hạn
sharey	Nếu <code>subplots=True</code> , chia sẻ cùng trục y
figsize	Kích thước của hình dưới dạng <i>tuple</i>
title	Tiêu đề của biểu đồ dưới dạng chuỗi
legend	Thêm chú thích ô phụ (mặc định là <code>True</code> )
sort_columns	Vẽ các cột theo thứ tự bảng chữ cái. Theo mặc định sử dụng thứ tự cột hiện có



## 2.2. Biểu đồ thanh (Bar plots)

- *Plot.bar()* và *plot.barh()* lần lượt tạo ra các ô thanh dọc và ngang. Trong trường hợp này, chỉ mục *Series* hoặc *DataFrame* sẽ được sử dụng làm dấu tích *x* (*bar*) hoặc *y* (*barh*). Các tùy chọn *color='k'* và *alpha=0.7* đặt màu của ô thành màu đen và sử dụng độ trong suốt một phần cho phân tô:

```
In [51]: fig, axes = plt.subplots(2, 1)
In [52]: data = pd.Series(np.random.rand(16),
                          index=list('abcdefghijklmnop'))
In [53]: data.plot.bar(ax=axes[0], color='k', alpha=0.7)
Out[53]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb62493d470>
In [54]: data.plot.barh(ax=axes[1], color='k', alpha=0.7)
```



Biểu đồ thanh ngang và dọc

## 2.2. Biểu đồ thanh (Bar plots)

- Với *DataFrame*, các biểu đồ bar nhóm các giá trị trong mỗi hàng lại với nhau thành một nhóm theo các thanh, cạnh nhau, cho mỗi giá trị. Lưu ý rằng tên “*Genus*” trên các cột của *DataFrame* được sử dụng để đặt tiêu đề cho chú thích.

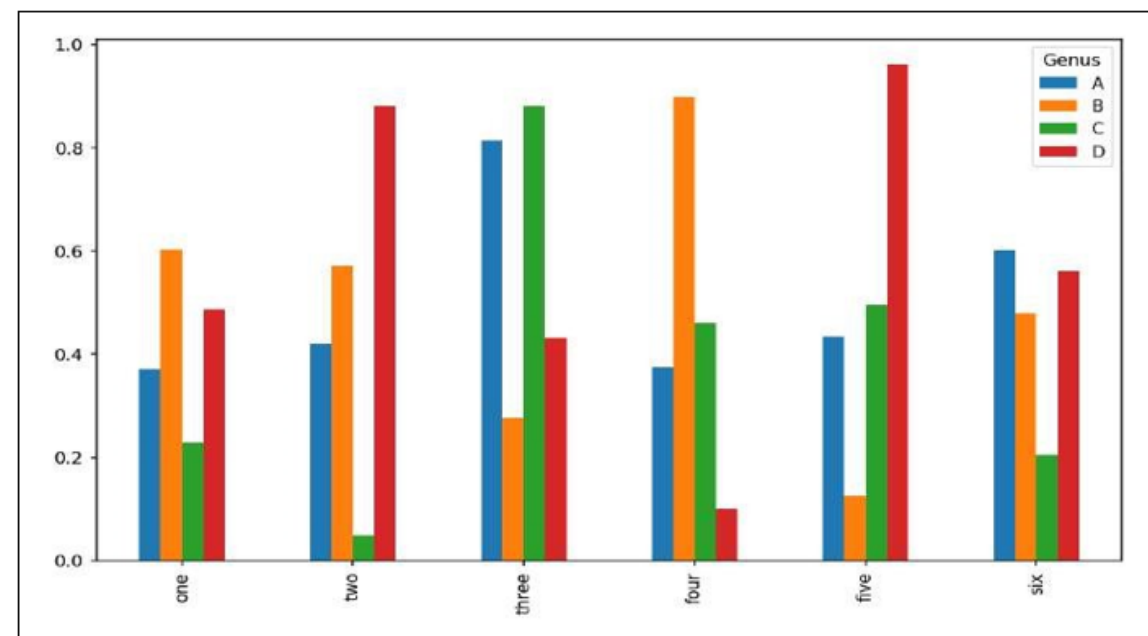
```
In [55]: df = pd.DataFrame(np.random.rand(6, 4),  
.....: index=['one', 'two', 'three', 'four', 'five', 'six'],  
.....: columns=pd.Index(['A', 'B', 'C', 'D'], name='Genus'))
```

```
In [56]: df
```

```
Out[56]:
```

Genus	A	B	C	D
one	0.370670	0.602792	0.229159	0.486744
two	0.420082	0.571653	0.049024	0.880592
three	0.814568	0.277160	0.880316	0.431326
four	0.374020	0.899420	0.460304	0.100843
five	0.433270	0.125107	0.494675	0.961825
six	0.601648	0.478576	0.205690	0.560547

```
In [57]: df.plot.bar()
```



Biểu đồ thanh *DataFrame*.

## 2.2. Biểu đồ thanh (Bar plots)

- Với dữ liệu yêu cầu tổng hợp hoặc tóm tắt trước khi lập biểu đồ, việc sử dụng gói *seaborn* có thể khiến mọi việc đơn giản. Ví dụ:

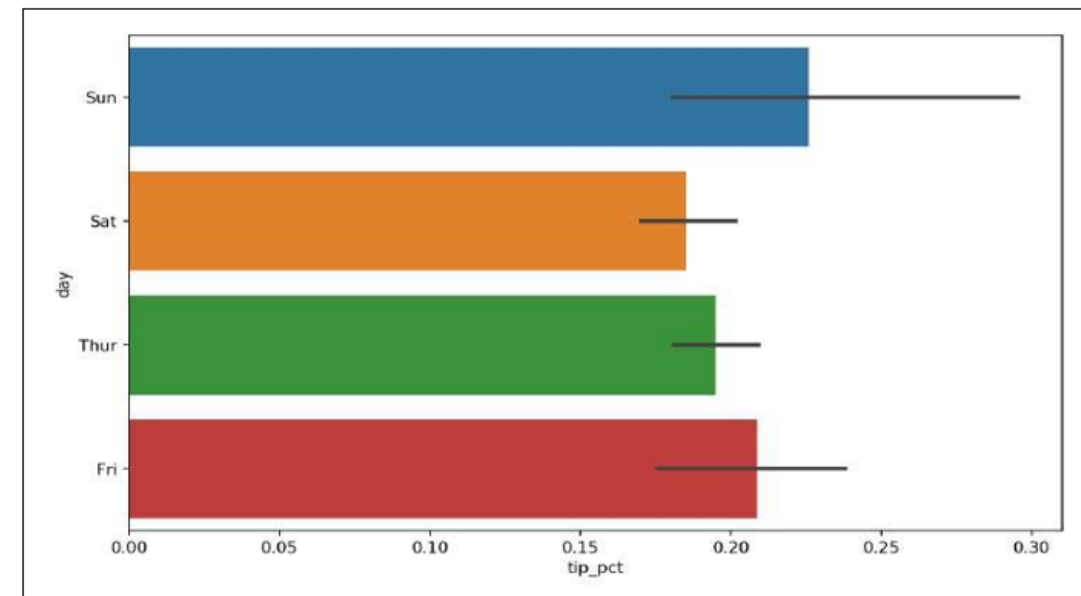
```
In [66]: import seaborn as sns
In [67]: tips['tip_pct'] = tips['tip'] / (tips['total_bill'] - tips['tip'])
In [68]: tips.head()
Out[68]:
```

	total_bill	tip	smoker	day	time	size	tip_pct
0	16.99	1.01	No	Sun	Dinner	2	0.063204
1	10.34	1.66	No	Sun	Dinner	3	0.191244
2	21.01	3.50	No	Sun	Dinner	3	0.199886
3	23.68	3.31	No	Sun	Dinner	2	0.162494
4	24.59	3.61	No	Sun	Dinner	4	0.172069

```
In [69]: sns.barplot(x='tip_pct', y='day', data=tips, orient='h')
```

- Các hàm vẽ biểu đồ trong *seaborn*:

- Một đối số *data*, có thể là *DataFrame* của *pandas*.
- Các đối số khác đề cập đến tên cột. Vì có nhiều quan sát cho mỗi giá trị trong ngày nên các thanh là giá trị trung bình của *tip\_pct*.
- Các đường màu đen vẽ trên các thanh biểu thị khoảng tin cậy 95% (điều này có thể được định cấu hình thông qua các đối số tùy chọn).

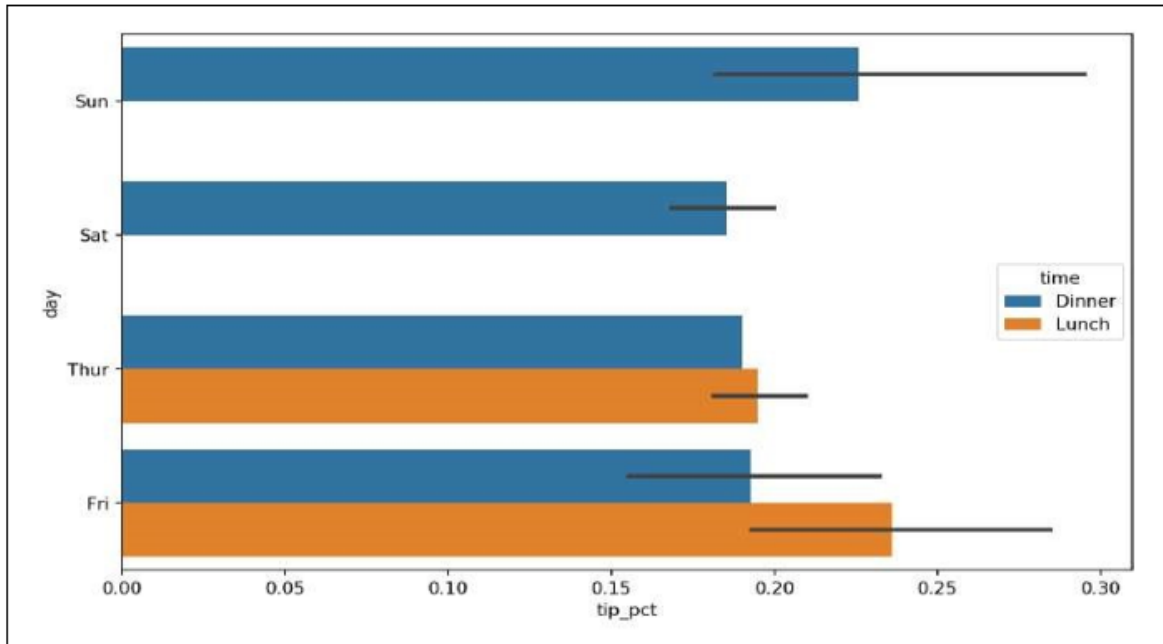


## 2. Vẽ đồ thị với *pandas* & *seaborn*

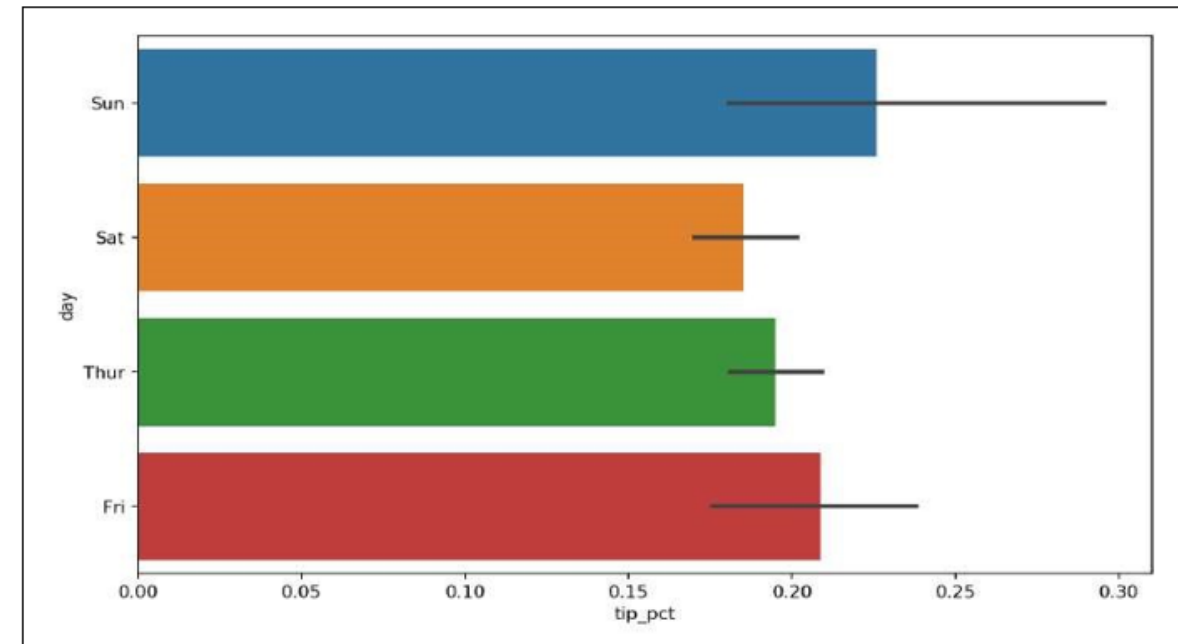
### 2.2. Biểu đồ thanh (Bar plots)

- *seaborn.barplot* có tùy chọn *hue* cho phép chia theo một giá trị phân loại bổ sung:

```
In [70]: sns.barplot(x='tip_pct', y='day', hue='time',  
                    data=tips, orient='h')
```



Tỷ lệ tiền boa theo ngày và giờ (có dùng tham số *hue*)



Tỷ lệ tiền boa theo ngày với các thanh lồi (không dùng tham số *hue*)

## 2.3. Biểu đồ thanh xếp chồng (Stacked Bar plots)

- Cú pháp để vẽ đồ thị này tương tự như đối với Bar plots, chỉ khác là truyền thêm tham số *stacked=True*.

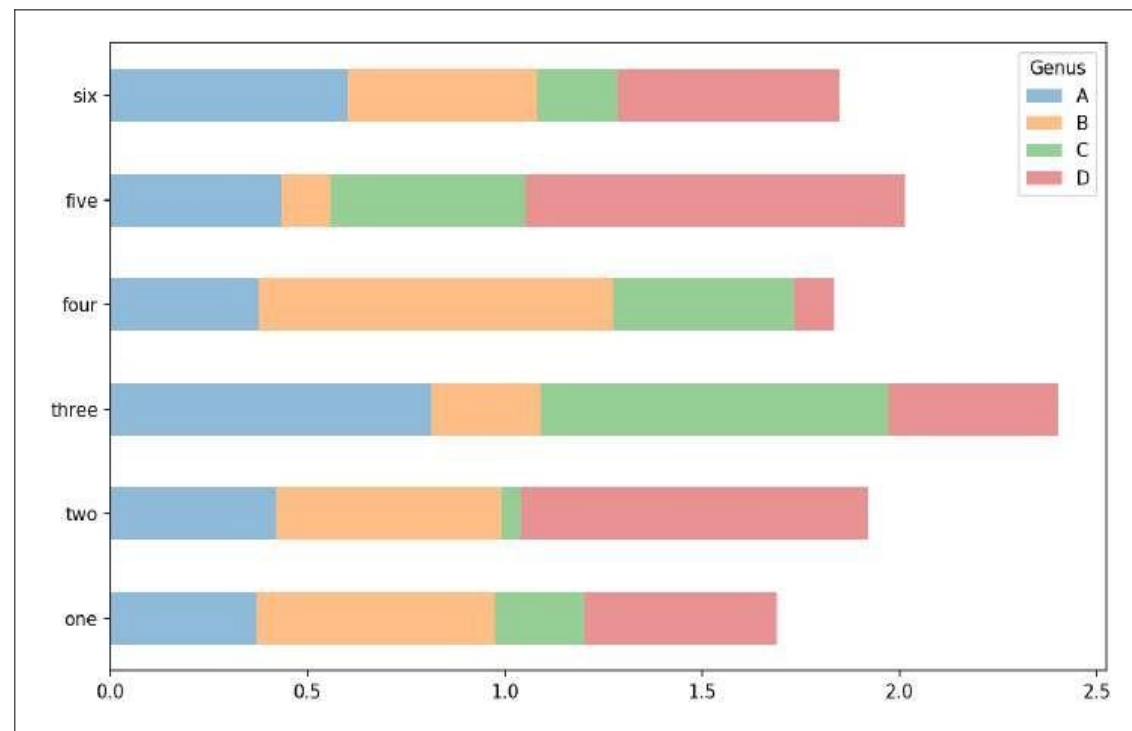
```
In [55]: df = pd.DataFrame(np.random.rand(6, 4),  
.....: index=['one', 'two', 'three', 'four', 'five', 'six'],  
.....: columns=pd.Index(['A', 'B', 'C', 'D'], name='Genus'))
```

```
In [56]: df
```

```
Out[56]:
```

Genus	A	B	C	D
one	0.370670	0.602792	0.229159	0.486744
two	0.420082	0.571653	0.049024	0.880592
three	0.814568	0.277160	0.880316	0.431326
four	0.374020	0.899420	0.460304	0.100843
five	0.433270	0.125107	0.494675	0.961825
six	0.601648	0.478576	0.205690	0.560547

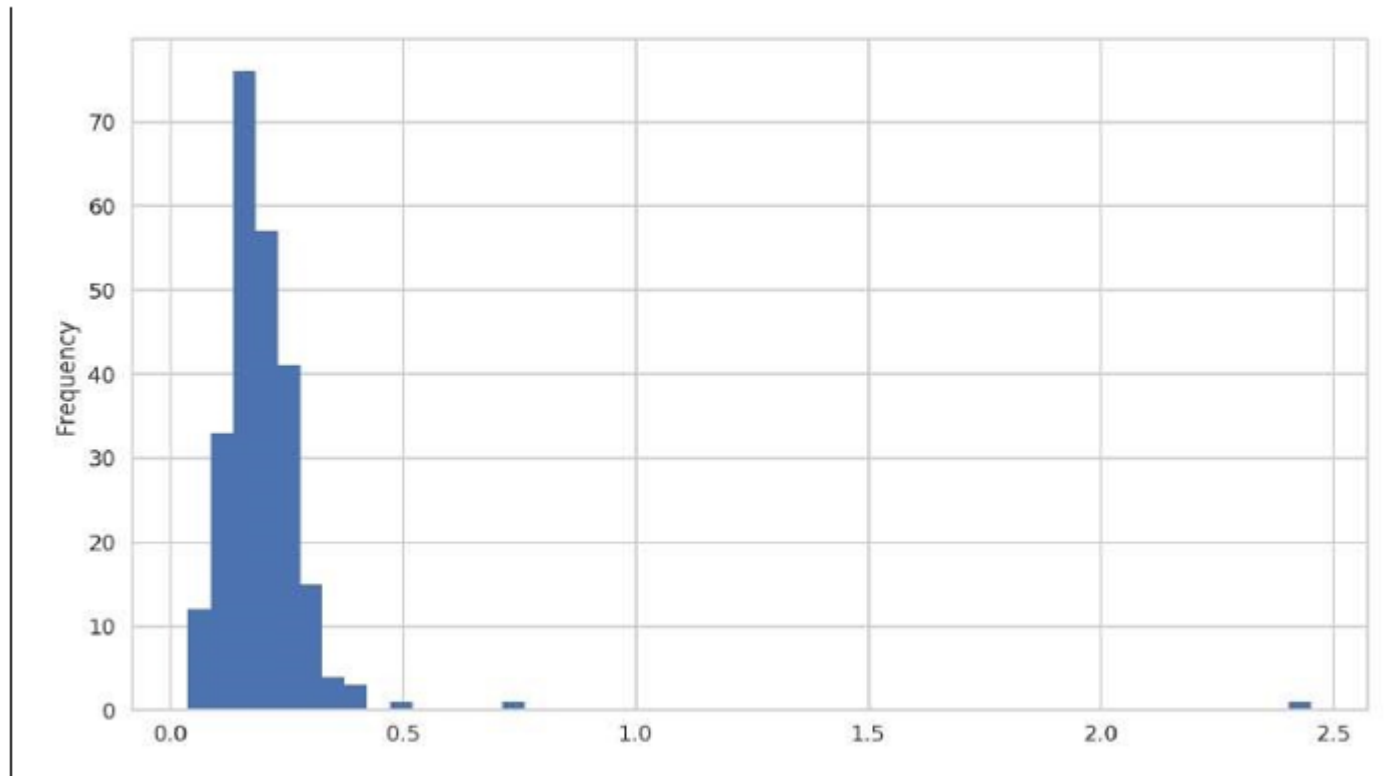
```
In [58]: df.plot.barh(stacked=True, alpha=0.5)
```



## 2.4. Histograms

- *Histogram* là một loại biểu đồ dạng thanh cung cấp sự hiển thị rời rạc về tần số giá trị. Các điểm dữ liệu được chia thành các ngăn rời rạc, cách đều nhau và số điểm dữ liệu trong mỗi ngăn được vẽ trên đồ thị. Sử dụng dữ liệu tiền boia trong các minh họa trước, có thể tạo biểu đồ tỷ lệ phần trăm tiền boia trên tổng hóa đơn bằng phương thức *plot.hist* trên *Series*:

```
In [72]: tips['tip_pct'].plot.hist(bins=50)
```

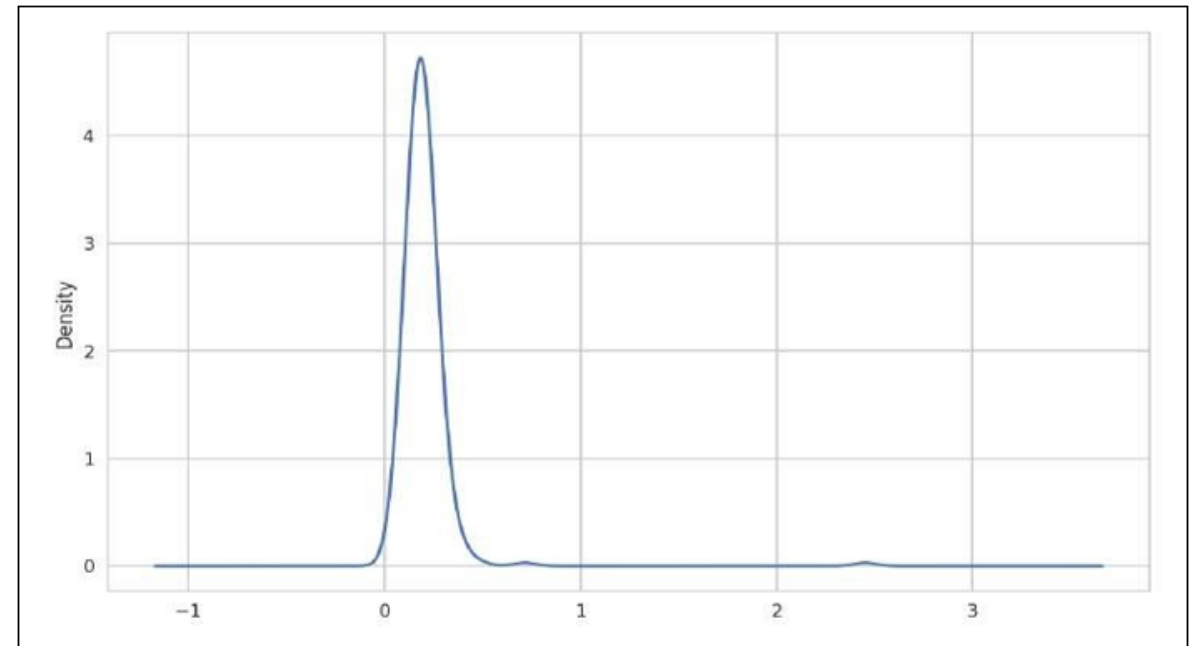


*Biểu đồ tỷ lệ phần trăm tiền boia*

## 2.5. Biểu đồ mật độ (*Density Plots*)

- Loại biểu đồ liên quan là biểu đồ mật độ (*density plot*), được hình thành bằng cách tính toán ước tính phân bố xác suất liên tục (*continuous probability distribution*) có thể đã tạo ra dữ liệu được quan sát. Quy trình thông thường là tính gần đúng phân bố này dưới dạng hỗn hợp các “hạt nhân” (*kernels*) - tức là các phân bố đơn giản hơn như phân bố chuẩn (*normal distribution*). Do đó, biểu đồ mật độ còn được gọi là biểu đồ ước tính mật độ hạt nhân (*kernel density estimate* - KDE). Việc sử dụng *plot.kde* tạo một biểu đồ mật độ bằng cách sử dụng ước tính hỗn hợp chuẩn thông thường

```
In [73]: tips['tip_pct'].plot.density()
```



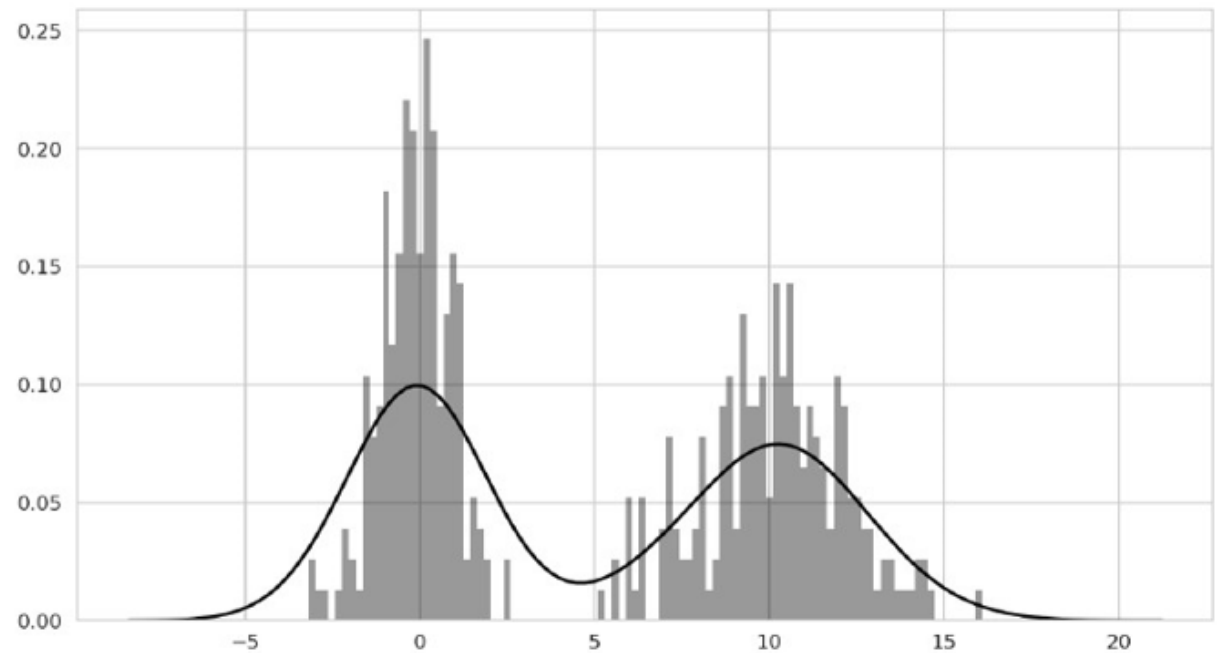
Biểu đồ mật độ phần trăm tiền boa



## 2.5. Biểu đồ mật độ (*Density Plots*)

- *Seaborn* làm cho 2 biểu đồ *histograms* và *density* trở nên dễ dàng hơn thông qua phương thức *distplot*, phương thức này có thể vẽ đồng thời cả biểu đồ *histograms* và biểu đồ ước tính mật độ liên tục (*continuous density estimate*). Ví dụ, xem xét một phân phối 2 phương thức (*bimodal distribution*) bao gồm các kết quả rút ra từ hai phân phối chuẩn hóa khác nhau:

```
In [74]: comp1 = np.random.normal(0, 1, size=200)
In [75]: comp2 = np.random.normal(10, 2, size=200)
In [76]: values = pd.Series(np.concatenate([comp1, comp2]))
In [77]: sns.distplot(values, bins=100, color='k')
```

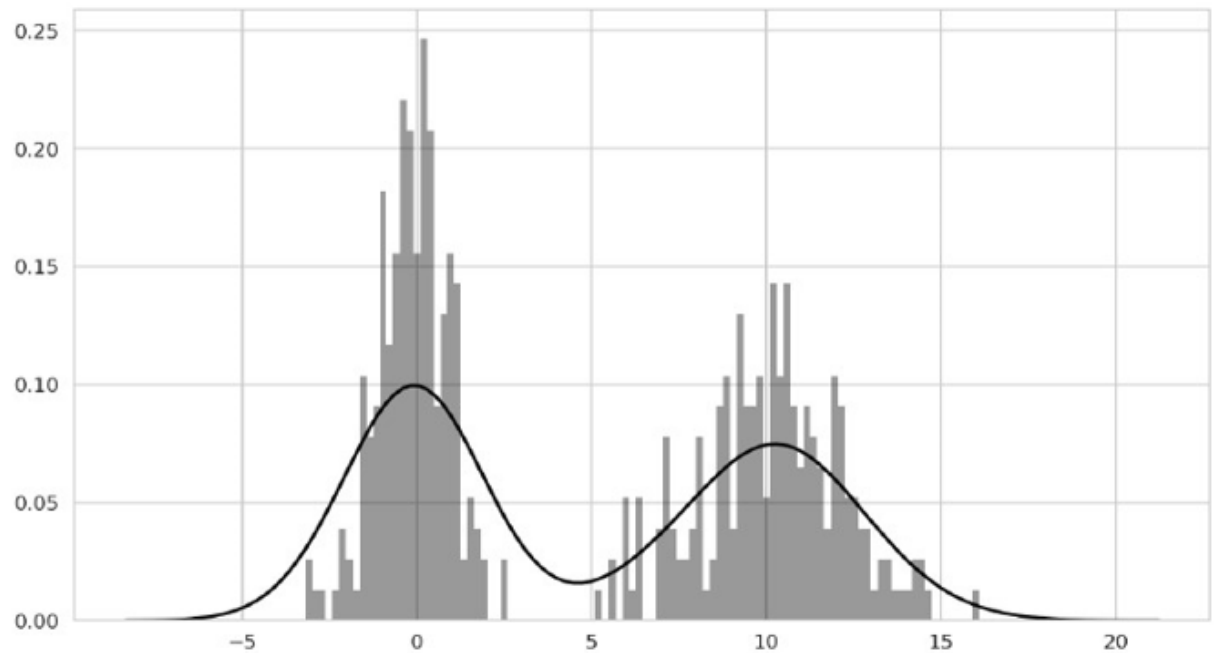


Histogram và density được vẽ đồng thời trên cùng 1 biểu đồ

## 2.5. Biểu đồ mật độ (*Density Plots*)

- *Seaborn* làm cho 2 biểu đồ *histograms* và *density* trở nên dễ dàng hơn thông qua phương thức *distplot*, phương thức này có thể vẽ đồng thời cả biểu đồ *histograms* và biểu đồ ước tính mật độ liên tục (*continuous density estimate*). Ví dụ, xem xét một phân phối 2 phương thức (*bimodal distribution*) bao gồm các kết quả rút ra từ hai phân phối chuẩn hóa khác nhau:

```
In [74]: comp1 = np.random.normal(0, 1, size=200)
In [75]: comp2 = np.random.normal(10, 2, size=200)
In [76]: values = pd.Series(np.concatenate([comp1, comp2]))
In [77]: sns.distplot(values, bins=100, color='k')
```



Histogram và density được vẽ đồng thời trên cùng 1 biểu đồ

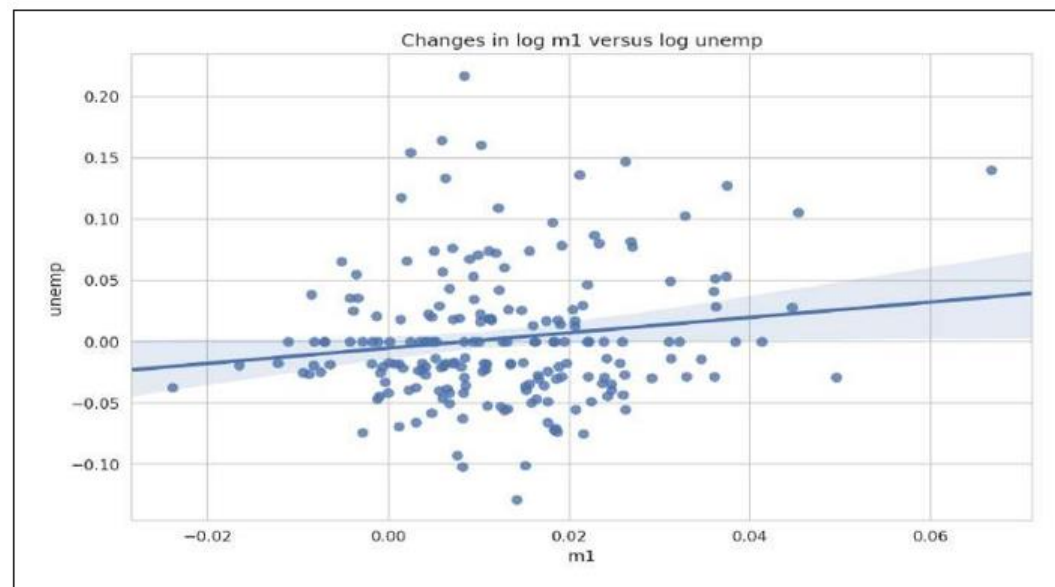
## 2.6. Biểu đồ phân tán (hoặc biểu đồ điểm - Scatter or Point Plots)

- Biểu đồ điểm hoặc biểu đồ phân tán có thể là một cách hữu ích để kiểm tra mối quan hệ giữa hai *Series* dữ liệu một chiều. Ví dụ sau tải tập dữ liệu `macrodata` từ dự án mô hình thống kê, chọn một vài biến, sau đó tính toán sự khác biệt:

```
In [78]: macro = pd.read_csv('examples/macrodata.csv')
In [79]: data = macro[['cpi', 'm1', 'tbilrate', 'unemp']]
In [80]: trans_data = np.log(data).diff().dropna()
In [81]: trans_data[-5:]
Out[81]:
```

	cpi	m1	tbilrate	unemp
198	-0.007904	0.045361	-0.396881	0.105361
199	-0.021979	0.066753	-2.277267	0.139762
200	0.002340	0.010286	0.606136	0.160343
201	0.008419	0.037461	-0.200671	0.127339
202	0.008894	0.012202	-0.405465	0.042560

```
In [82]: sns.regplot('m1', 'unemp', data=trans_data)
Out[82]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb613720be0>
In [83]: plt.title('Changes in log %s versus log %s' % ('m1', 'unemp'))
```

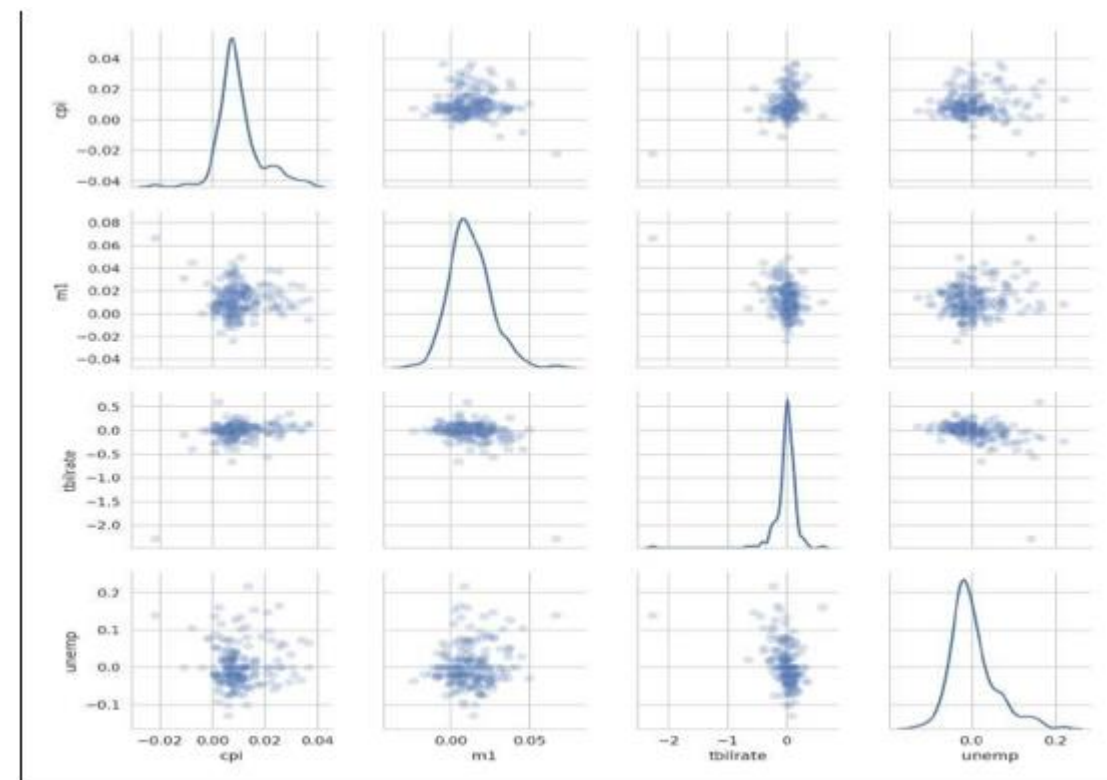


Biểu đồ hồi quy/phân tán của *seaborn*

## 2.6. Biểu đồ phân tán (hoặc biểu đồ điểm - *Scatter or Point Plots*)

- Khi khám phá việc phân tích dữ liệu, sẽ rất hữu ích khi có thể xem xét tất cả các biểu đồ phân tán giữa một nhóm biến; đây được gọi là ma trận biểu đồ cặp (*pairs plot* hoặc *scatter plot*). Tạo một biểu đồ như vậy từ đầu là một công việc hơi tốn công, vì vậy *seaborn* có hàm *pairplot* hỗ trợ đặt biểu đồ hoặc ước tính mật độ của từng biến dọc theo đường chéo – *diagonal*.
- Đối số *plot\_kws*: cho phép chuyển các tùy chọn cấu hình tới các lệnh gọi vẽ đồ thị riêng lẻ trên các phần tử ngoài đường chéo. Hãy xem tài liệu về *seaborn.pairplot* để biết thêm các tùy chọn cấu hình chi tiết.

```
In [84]: sns.pairplot(trans_data, diag_kind='kde',  
                    plot_kws={'alpha': 0.2})
```

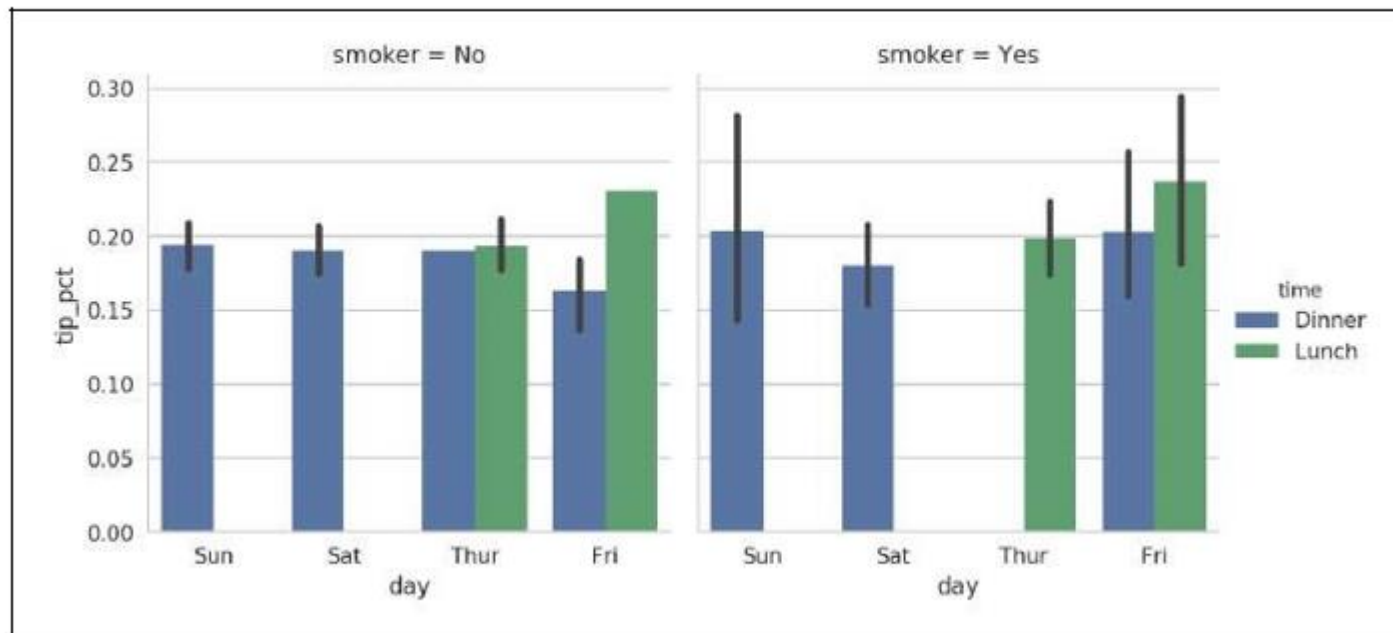


Ma trận biểu đồ cặp

## 2.7. Facet Grids và Categorical Data

- Với các tập dữ liệu có các thứ nguyên nhóm bổ sung (nhân tố thứ 3): một cách để trực quan hóa dữ liệu với nhiều biến phân loại là sử dụng *facet grid*. *Seaborn* có một hàm *factorplot* tích hợp sẵn giúp đơn giản hóa việc tạo nhiều loại biểu đồ *facet grid*.

```
In [85]: sns.factorplot(x='day', y='tip_pct', hue='time',  
                        col='smoker', kind='bar', data=tips[tips.tip_pct < 1])
```

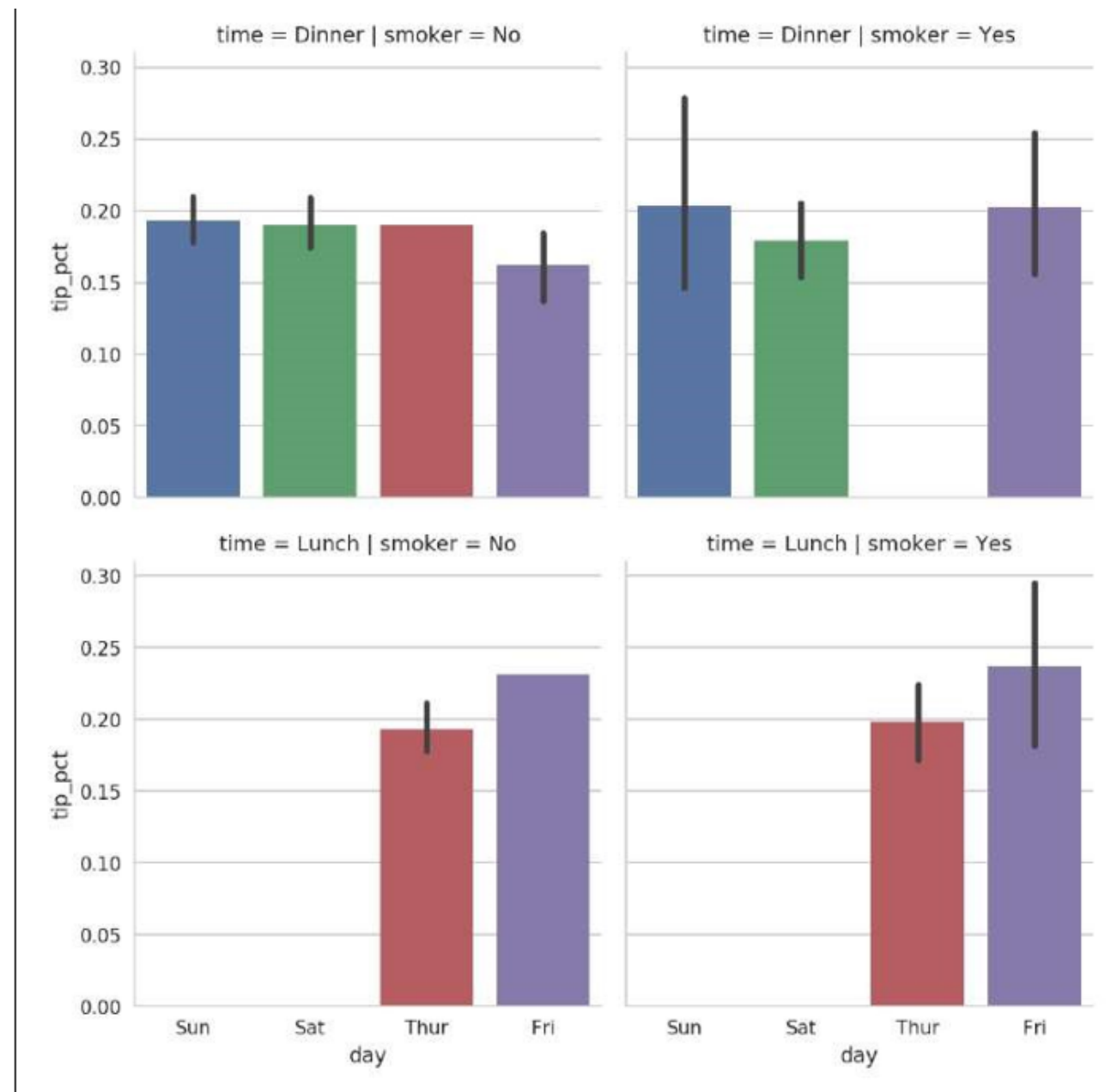


*Tỷ lệ tiền boa theo ngày/giờ/người hút thuốc*

## 2.7. Facet Grids và Categorical Data

- Với các tập dữ liệu có các thứ nguyên nhóm bổ sung (nhân tố thứ 3): một cách để trực quan hóa dữ liệu với nhiều biến phân loại là sử dụng *facet grid*. *Seaborn* có một hàm *factorplot* tích hợp sẵn giúp đơn giản hóa việc tạo nhiều loại biểu đồ *facet grid*.

```
In [86]: sns.factorplot(x='day', y='tip_pct',  
                        row='time', col='smoker',  
                        kind='bar',  
                        data=tips[tips.tip_pct < 1])
```

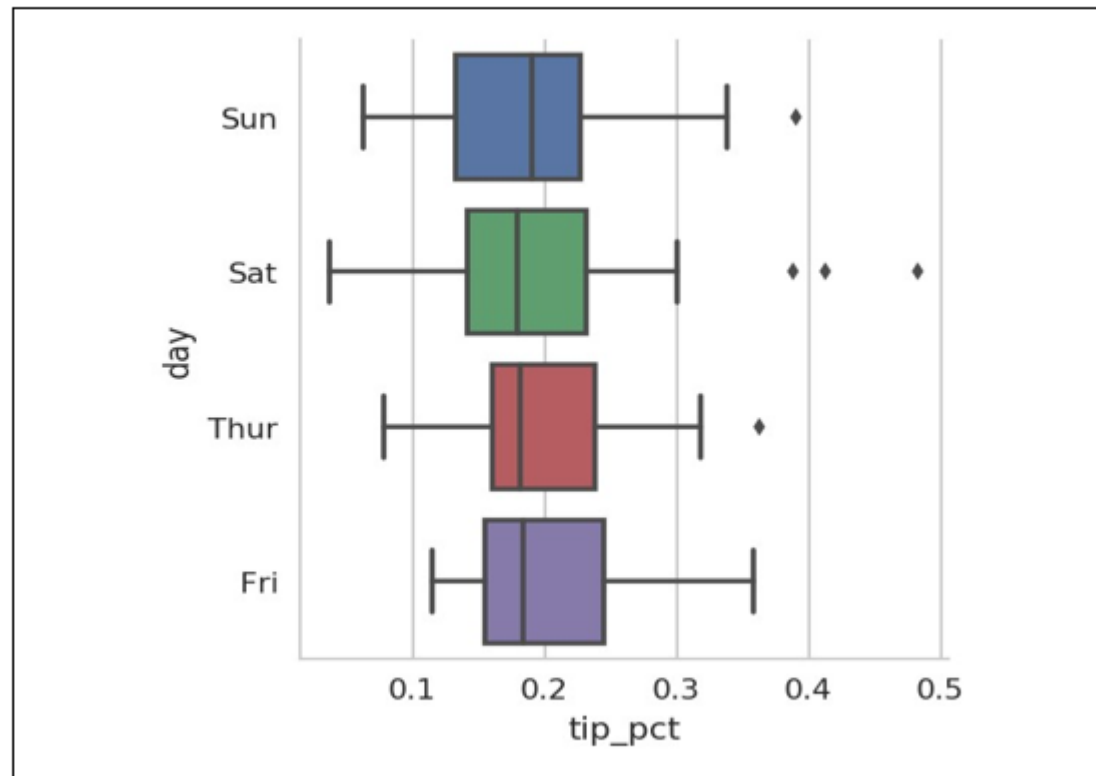


Biểu diễn *tip\_pct* theo ngày; xét theo các khía cạnh *time/smoker*

## 2.7. Facet Grids và Categorical Data

- *factorplot* hỗ trợ các loại biểu đồ khác có thể hữu ích tùy thuộc vào nội dung mà ta đang cố gắng hiển thị. Ví dụ: biểu đồ hình hộp (*boxplot* - hiển thị trung vị, tứ phân vị và ngoại lệ) có thể là một loại trực quan hóa hiệu quả

```
In [87]: sns.factorplot(x='tip_pct', y='day', kind='box',  
                        data=tips[tips.tip_pct < 0.5])
```



Biểu đồ hộp (Box plot) của *tip\_pct* theo ngày



