

1 Numpy

<https://numpy.org/doc/stable/user/quickstart.html>

<https://cs231n.github.io/python-numpy-tutorial/#numpy>

Numpy là một thư viện lõi phục vụ cho khoa học máy tính của Python, hỗ trợ cho việc tính toán các mảng nhiều chiều, có kích thước lớn với các hàm đã được tối ưu áp dụng lên các mảng nhiều chiều đó. Numpy đặc biệt hữu ích khi thực hiện các hàm liên quan tới Đại Số Tuyến Tính.

Cài đặt bằng `pip install numpy`

1.1 Arrays

```
import numpy as np

a = np.array([1, 2, 3])    # Create a rank 1 array
print(type(a))            # Prints "<class 'numpy.ndarray'>"
print(a.shape)            # Prints "(3,)"
print(a[0], a[1], a[2])   # Prints "1 2 3"
a[0] = 5                  # Change an element of the array
print(a)                  # Prints "[5, 2, 3]"

b = np.array([[1,2,3],[4,5,6]]) # Create a rank 2 array
print(b.shape)            # Prints "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0]) # Prints "1 2 4"
```

Numpy cũng cung cấp rất nhiều hàm để khởi tạo arrays

```
import numpy as np

a = np.zeros((2,2))    # Tạo một numpy array với tất cả phần tử là 0
print(a)               # "[[ 0.  0.]
                      #  [ 0.  0.]]"

b = np.ones((1,2))     # Tạo một numpy array với tất cả phần tử là 1
print(b)               # "[[ 1.  1.]]"

c = np.full((2,2), 7)  # Tạo một mảng hằng
print(c)               # "[[ 7.  7.]
                      #  [ 7.  7.]]"
```

```

d = np.eye(2)          # Tạo một ma trận đơn vị 2 x 2
print(d)               # "[[ 1.  0.]
                       #  [ 0.  1.]]"

e = np.random.random((2,2)) # Tạo một mảng với các giá trị ngẫu nhiên
print(e)               # Có thể là "[[ 0.91940167  0.08143941]
                       #  [ 0.68744134  0.87236687]]"

f = np.arange(10) # Tạo 1 numpy array với các phần tử từ 0 đến 9
print(f)          # "[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]"

```

1.2 Array indexing

Numpy cung cấp một số cách để truy xuất phần tử trong mảng

Slicing: Tương tự như list trong python, numpy arrays cũng có thể được cắt.
`import numpy as np`

```

# Khởi tạo numpy array có shape = (3, 4) có giá trị như sau:
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Sử dụng slicing để tạo numpy array b bằng cách lấy 2 hàng đầu tiên
# và cột 1, 2. Như vậy b sẽ có shape = (2, 2):
# [[2 3]
#  [6 7]]
b = a[:2, 1:3]

# Một array tạo ra từ slicing sẽ có cùng địa chỉ với array gốc.
# Nếu thay đổi 1 trong 2 thì array còn lại cũng thay đổi.
print(a[0, 1]) # Prints "2"
b[0, 0] = 77   # b[0, 0] ở đây tương đương với a[0, 1]
print(a[0, 1]) # Prints "77"

```

Bạn cũng có thể kết hợp việc dùng slicing và dùng chỉ số. Tuy nhiên, cách làm đó sẽ cho ra một mảng mới có rank thấp hơn mảng gốc.

```
import numpy as np
```

```
# Tạo một numpy array có shape (3, 4) với giá trị như sau:
```

```

# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Hai cách truy cập dữ liệu ở hàng giữa của mảng
# Dùng kết hợp chỉ số và slice -> được array mới có rank thấp hơn,
# Nếu chỉ dùng slice ta sẽ có 1 array mới có cùng rank
# với array gốc
row_r1 = a[1, :]    # Rank 1, hàng thứ 2 của a
row_r2 = a[1:2, :]  # Rank 2, vẫn là hàng thứ 2 của a
print(row_r1, row_r1.shape) # Prints "[5 6 7 8] (4,)"
print(row_r2, row_r2.shape) # Prints "[[5 6 7 8]] (1, 4)"

# Chúng ta có thể làm tương tự với cột của numpy array:
col_r1 = a[:, 1]
col_r2 = a[:, 1:2]
print(col_r1, col_r1.shape) # Prints "[ 2  6 10] (3,)"
print(col_r2, col_r2.shape) # Prints "[[ 2]
#                               [ 6]
#                               [10]] (3, 1)"

```

Integer array indexing: Khi bạn truy xuất mảng dùng slicing, kết quả trả về sẽ là mảng con của mảng ban đầu, tuy nhiên sử dụng chỉ số mảng cho phép bạn xây dựng mảng tùy ý từ một mảng khác

```

import numpy as np

a = np.array([[1,2], [3, 4], [5, 6]])

# Truy xuất mảng dùng chỉ số.
# Kết quả thu được là 1 mảng có shape (3,)
print(a[[0, 1, 2], [0, 1, 0]]) # Prints "[1 4 5]"

# Sẽ thu được kết quả tương đương như trên với cách này:
print(np.array([a[0, 0], a[1, 1], a[2, 0]])) # Prints "[1 4 5]"

# Bạn được phép sử dụng chỉ số mảng để
# truy xuất tới 1 phần tử
# của mảng gốc nhiều hơn 1 lần
print(a[[0, 0], [1, 1]]) # Prints "[2 2]"

# Một cách làm khác tương đương:
print(np.array([a[0, 1], a[0, 1]])) # Prints "[2 2]"

```

Một mẹo hữu ích dùng chỉ số mảng để chọn và thay đổi phần tử từ mỗi hàng của ma trận

```
import numpy as np

# Tạo một mảng mới từ đó ta sẽ chọn các phần tử
a = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])

print(a) # prints "array([[ 1,  2,  3],
#                [ 4,  5,  6],
#                [ 7,  8,  9],
#                [10, 11, 12]])"

# Tạo một mảng các chỉ số
b = np.array([0, 2, 0, 1])

# Lấy 1 phần tử từ mỗi hàng của a dùng với chỉ số ở mảng b
print(a[np.arange(4), b]) # Prints "[ 1  6  7 11]"

# Thay đổi một phần tử từ mỗi hàng của a dùng với chỉ số ở mảng b
a[np.arange(4), b] += 10

print(a) # prints "array([[11,  2,  3],
#                [ 4,  5, 16],
#                [17,  8,  9],
#                [10, 21, 12]])"
```

Boolean array indexing: Cho phép bạn lựa chọn ra các phần tử tùy ý của một mảng, thường dùng để chọn ra các phần tử thỏa mãn điều kiện nào đó

```
import numpy as np

a = np.array([[1,2], [3, 4], [5, 6]])

bool_idx = (a > 2) # Tìm các phần tử lớn hơn 2;
                  # Trả về 1 numpy array of Booleans có shape như mảng
a
                  # và giá trị tại mỗi phần tử là
                  # True nếu phần tử của a tại đó > 2,
                  # False cho trường hợp ngược lại.

print(bool_idx) # Prints "[[False False]
#               [ True  True]
#               [ True  True]]"
```

```
# Sử dụng một boolean array indexing để lấy
# các phần tử thỏa mãn điều kiện nhất định trong a
# Ví dụ ở đây in ra các giá trị của a > 2
# sử dụng array bool_idx đã tạo
print(a[bool_idx]) # Prints "[3 4 5 6]"

# Một cách ngắn gọn hơn:
print(a[a > 2])    # Prints "[3 4 5 6]"
```

1.3 Datatypes

Mỗi numpy array là một lưới các phần tử cùng kiểu dữ liệu. Numpy cung cấp một tập hợp lớn các kiểu dữ liệu số mà bạn có thể sử dụng để xây dựng các mảng. Numpy cố gắng đoán một kiểu dữ liệu khi bạn tạo một mảng, nhưng các hàm xây dựng các mảng thường cũng bao gồm một đối số tùy chọn để chỉ định rõ ràng kiểu dữ liệu

```
import numpy as np

x = np.array([1, 2]) # Để numpy xác định kiểu dữ liệu
print(x.dtype)      # Prints "int64"

x = np.array([1.0, 2.0]) # Để numpy xác định kiểu dữ liệu
print(x.dtype)          # Prints "float64"

x = np.array([1, 2], dtype=np.int64) # Chỉ định kiểu dữ liệu cụ thể cho
mảng
print(x.dtype)                  # Prints "int64"
```

1.4 Array math

Ghép, cộng, nhân, hoán vị chỉ với một dòng code. Dưới đây là một số ví dụ về các phép toán số học và nhân khác nhau với các mảng Numpy

```
import numpy as np

x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)
```

```

# Tổng của 2 mảng, cả 2 cách cho cùng một kết quả
# [[ 6.0  8.0]
#  [10.0 12.0]]
print(x + y)
print(np.add(x, y))

# Hiệu 2 mảng
# [[-4.0 -4.0]
#  [-4.0 -4.0]]
print(x - y)
print(np.subtract(x, y))

# Tính tích từng phần tử của x nhân với từng phần tử của y
# [[ 5.0 12.0]
#  [21.0 32.0]]
print(x * y)
print(np.multiply(x, y))

# Tương tự thương của từng phần x chia với từng phần tử y
# [[ 0.2          0.33333333]
#  [ 0.42857143  0.5        ]]
print(x / y)
print(np.divide(x, y))

# Bình phương từng phần tử trong x
# [[ 1.          1.41421356]
#  [ 1.73205081  2.         ]]
print(np.sqrt(x))

```

Để nhân 2 ma trận hoặc nhân vector với ma trận trong numpy, chúng ta sử dụng hàm **dot**

```

import numpy as np

x = np.array([[1,2],[3,4]])
y = np.array([[5,6],[7,8]])

v = np.array([9,10])
w = np.array([11, 12])

# Tích trong của 2 vector; Cả 2 đều cho kết quả là 219
print(v.dot(w))
print(np.dot(v, w))

# Nhân ma trận với vector; cả 2 đều cho mảng rank 1: [29 67]

```

```
print(x.dot(v))
print(np.dot(x, v))

# Ma trận với ma trận; cả 2 đều cho mảng rank 2
# [[19 22]
#  [43 50]]
print(x.dot(y))
print(np.dot(x, y))
```

Numpy cung cấp nhiều hàm hữu ích để thực hiện tính toán trên mảng; một trong những hàm hữu ích nữa là **sum**

```
import numpy as np

x = np.array([[1,2],[3,4]])

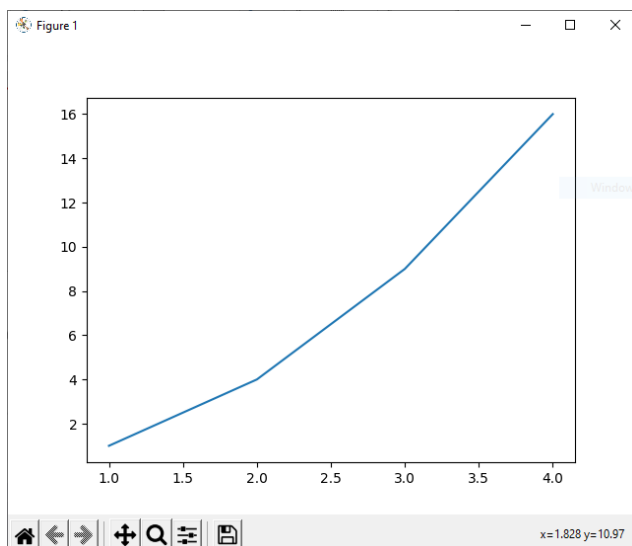
print(np.sum(x)) # Tổng các phần tử của mảng; prints "10"
print(np.sum(x, axis=0)) # Tính tổng theo từng cột; prints "[4 6]"
print(np.sum(x, axis=1)) # Tính tổng theo từng hàng; prints "[3 7]"
```

2 Matplotlib

2.1 Ví dụ 1

```
import matplotlib.pyplot as plt

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



Chỉ định kích thước: `figure()`, thêm tiêu đề và đặt tên trục x và trục y bằng cách sử dụng các phương thức `title()`, `xlabel()` và `ylabel()`.

```
plt.figure(figsize=(15, 5))
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
plt.title("First Plot")
plt.xlabel("X label")
plt.ylabel("Y label")
plt.show()
```

2.2 Ví dụ 2: Kết hợp Numpy

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

```
xpoints = np.array([0, 6])
ypoints = np.array([0, 250])
```

```
plt.plot(xpoints, ypoints, marker='o')
plt.show()
```

2.3 Ví dụ 3: Vẽ đường

'''

linestyle (ls): https://matplotlib.org/stable/gallery/lines_bars_and_markers/linestyles.html

- dotted: dấu chấm
- dashed: dấu gạch
- 'solid' (default): nét liền đặc
- dashdot: chấm gạch

color (c): https://matplotlib.org/stable/gallery/color/named_colors.html

- viết tắt tiếng anh
- #rrggbb (giá trị r,g,b từ 0 --> F) ví dụ: c = '#4CAF50'

Độ dày nét vẽ

```
linewidth = '20.5'
```

```
lw = '12'
```

```
'''
```

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

```
import random
```

```
import numpy as np
```

```
# Sinh số ngẫu nhiên từ 5 --> 50
```

```
n = random.randint(5, 10)
```

```
xpoints = []
```

```
ypoints = []
```

```
# Khởi tạo ngẫu nhiên n phần tử cho x/y
```

```
for i in range(0, n):
```

```
    xpoints.append(random.randint(1, 20))
```

```
    ypoints.append(random.randint(1, 20))
```

```
plt.plot(xpoints, ypoints, linestyle='dotted', color='r', lw='1')
```

```
y1 = np.array([3, 8, 1, 10])
```

```
y2 = np.array([6, 2, 7, 11])
```

```
plt.plot(y1)
```

```
plt.plot(y2)
```

```
# Setting cho đồ thị
```

```
plt.title("Đồ thị các điểm")
```

```
plt.xlabel("Trục x")
plt.ylabel("Trục y")

# Tùy chọn hiển thị lưới
# plt.grid() # Cả 2
plt.grid(axis='x') # Chỉ có trục x

plt.show() # Chỉ có 1
```

Bổ sung:

```
plot(x, y, format)
```

- Tham số x là danh sách các tọa độ trục x
- Tham số y là danh sách các tọa độ trục y
- format định dạng đồ thị

Một số định dạng khác như sau:

- `'r*-'` các điểm hình ngôi sao màu đỏ, đường nối các điểm dạng `-`.
- `'bD-.'` các điểm hình kim cương màu xanh dương, đường nối các điểm dạng `-`.
- `'g^--'` các điểm hình tam giác hướng lên màu xanh lá, đường nối các điểm dạng `-`.
- Nếu bạn không muốn các điểm nối với nhau, có thể bỏ định dạng đường thẳng đi, ví dụ `'go-'` sẽ thành `'go'`

2.4 Mẫu hàm vẽ đồ thị

```
def plotting_workflow(data):

    # 1. Manipulate data

    # 2. Create plot

    # 3. Plot data

    # 4. Customize plot
```

```
# 5. Save plot

# 6. Return plot
return plot
```

Ví dụ:

```
import matplotlib.pyplot as plt

# 1. Prepare data
x = [1, 2, 3, 4]
y = [11, 22, 33, 44]

# 2. Setup plot
fig, ax = plt.subplots(figsize=(10,10)) # figsize dimension is inches

# 3. Plot data
ax.plot(x, y)

# 4. Customize plot
ax.set(title="Sample Simple Plot", xlabel="x-axis", ylabel="y-axis")

# 5. Save & show
fig.savefig("images/simple-plot.png")
```

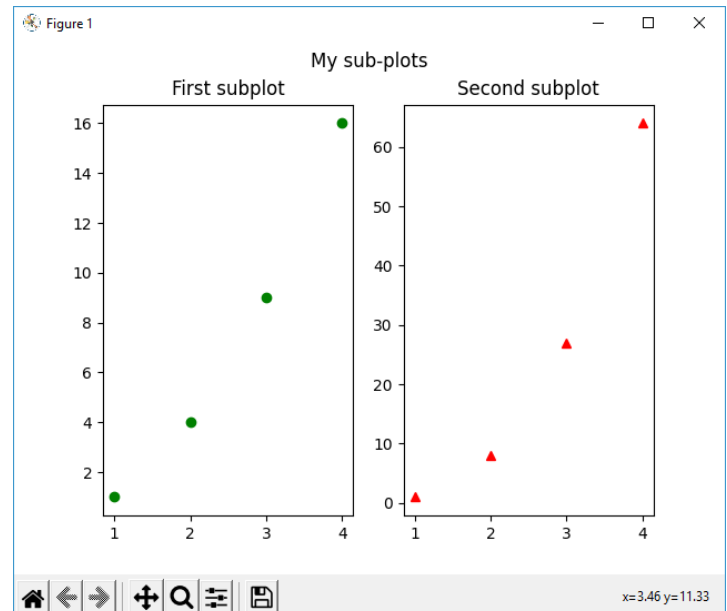
2.5 Vẽ nhiều đồ thị trên cùng một ảnh

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

```
plt.subplot(1, 2, 1)
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], "go")
plt.title("First subplot")
```

```
x = np.arange(1, 5)
y = x ** 3
plt.subplot(1, 2, 2)
plt.plot(x, y, "r^")
plt.title("Second subplot")
```

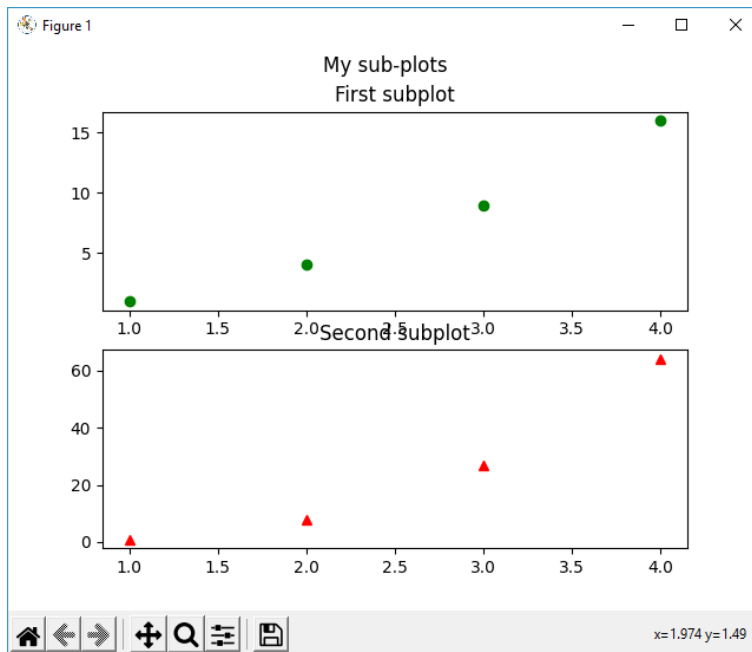
```
plt.suptitle("My sub-plots")
plt.show()
```



Nếu ta muốn các sub-plots thành hai hàng và một cột, chúng ta có thể truyền các đối số (2,1,1) và (2,1,2)

```
plt.subplot(2, 1, 1)
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], "go")
plt.title("First subplot")
```

```
plt.subplot(2, 1, 2)
plt.plot(x, y, "r^")
plt.title("Second subplot")
```



2.6 Bar chart

...

thuộc tính bar/barh:

- width: độ dày cột dành cho bar()
- height: độ dày cột dành cho barh()

...

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

```
import numpy as np
```

```
x = np.array(["A", "B", "C", "D"])
```

```
y = np.array([3, 8, 1, 10])
```

```
plt.subplot(2, 2, 1)
```

```
plt.bar(x, y, color="red", width=0.5)
```

```
plt.subplot(2, 2, 4)
```

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

```
plt.subplot(2, 2, 2)
```

```
index = np.arange(4)
```

```
y2 = np.array([3.3, 7.5, 1.9, 8.8])
```

```
plt.bar(index, y2, color='yellow', label="Fig 2")
```

```
plt.xlabel("Thực phẩm")
```

```
plt.ylabel("Doanh thu")
```

```
plt.show()
```

2.7 Pie chart

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

```
import numpy as np
```

```
y = np.array([19, 25, 32, 31])
```

```
data_labels = ["333", "Crystal Tiger", "Heineken", "Budweiser"]
```

```
# Độ hở
```

```
do_ho = [0.1, 0.05, 0.02, 0]
```

```
plt.pie(y, labels=data_labels, explode=do_ho, shadow=True)
```

```
plt.legend(title="GIÁ BÁN BIA")
plt.show()
```

2.8 Histogram

2.8.1 Ví dụ 1

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

```
# x = np.random.randn(100) # Tạo 100 phần tử ngẫu nhiên
x = np.random.normal(170, 10, 250)
print(x)
```

```
plt.hist(x, 11) # 11: số tầng
plt.show()
```

2.8.2 Ví dụ 2

```
# Make a Histogram plot
```

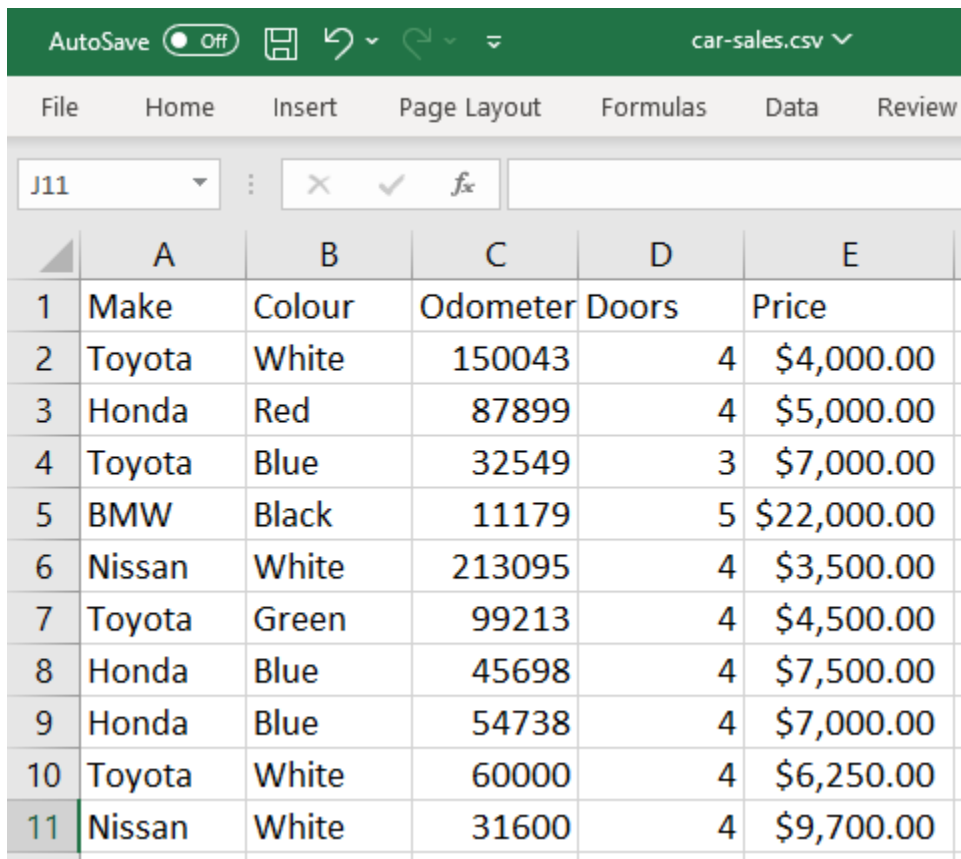
```
x = np.random.randn(1000) # Make some data from a normal
distribution
```

```
fig, ax = plt.subplots()
ax.hist(x);
```

```
x = np.random.random(1000) # random data from random distribution
fig, ax = plt.subplots()
ax.hist(x);
```

2.9 Ví dụ tổng hợp

Sử dụng file mẫu car-sales.csv như mẫu:



The screenshot shows the Microsoft Excel interface with a green title bar. The title bar includes 'AutoSave' (turned off), a save icon, undo and redo icons, and the file name 'car-sales.csv'. The ribbon at the top has tabs for 'File', 'Home', 'Insert', 'Page Layout', 'Formulas', 'Data', and 'Review'. The formula bar shows 'J11' and a formula icon. The main area displays a table with 6 columns: 'Make', 'Colour', 'Odometer', 'Doors', and 'Price'. The rows are numbered 1 to 11. The data is as follows:

	A	B	C	D	E
1	Make	Colour	Odometer	Doors	Price
2	Toyota	White	150043	4	\$4,000.00
3	Honda	Red	87899	4	\$5,000.00
4	Toyota	Blue	32549	3	\$7,000.00
5	BMW	Black	11179	5	\$22,000.00
6	Nissan	White	213095	4	\$3,500.00
7	Toyota	Green	99213	4	\$4,500.00
8	Honda	Blue	45698	4	\$7,500.00
9	Honda	Blue	54738	4	\$7,000.00
10	Toyota	White	60000	4	\$6,250.00
11	Nissan	White	31600	4	\$9,700.00

```
import pandas as pd
```

```
ts = pd.Series(np.random.randn(1000),  
               index=pd.date_range('1/1/2020', periods=1000))
```

```
ts = ts.cumsum() # Return cumulative sum over a DataFrame or  
Series
```

```
ts.plot();
```

```
# Make a dataframe
```

```
car_sales = pd.read_csv("data/car-sales.csv")
```

```

# Remove price column symbols
car_sales["Price"] = car_sales["Price"].str.replace('[\$\,\.]',
 '')
type(car_sales["Price"][0])

# Remove last two zeros from price
# 4 0 0 0 0 0
# [-6][-5][-4][-3][-2][-1]
car_sales["Price"] = car_sales["Price"].str[:-2]

car_sales["Sale Date"] = pd.date_range("1/1/2020",
periods=len(car_sales))
type(car_sales["Price"][0])

car_sales["Total Sales"] =
car_sales["Price"].astype(int).cumsum()

car_sales.plot(x="Sale Date", y="Total Sales");
car_sales["Price"] = car_sales["Price"].astype(int) # Reassign
price column to int

# Plot scatter plot with price column as numeric
car_sales.plot(x="Odometer (KM)", y="Price", kind="scatter");

# How about a bar graph?
x = np.random.rand(10, 4)
df = pd.DataFrame(x, columns=['a', 'b', 'c', 'd'])
df.plot.bar();
df.plot(kind='bar'); # Can do the same thing with 'kind' keyword

```



```
car_sales.plot(x='Make', y='Odometer (KM)', kind='bar');
```

```
# How about Histograms?
```

```
car_sales["Odometer (KM)"].plot.hist();
```

```
# bins=10 default , bin width =
```

```
25, car_sales["Price"].plot.hist(bins=10);
```

```
car_sales["Odometer (KM)"].plot(kind="hist");
```

```
# Default number of bins is 10, bin width = 12,500
```

```
car_sales["Odometer (KM)"].plot.hist(bins=20);
```

```
# Let's try with another dataset
```

```
heart_disease = pd.read_csv("data/heart-disease.csv")
```

```
# Create a histogram of age
```

```
heart_disease["age"].plot.hist(bins=50);
```

```
heart_disease.plot.hist(figsize=(10, 30), subplots=True);
```

```
over_50 = heart_disease[heart_disease["age"] > 50]
```

```
# Pyplot method
```

```
# c: change colour of plot base on target value [0, 1]
```

```
over_50.plot(kind='scatter',  
             x='age',  
             y='chol',  
             c='target',  
             figsize=(10, 6));
```

```
# OO method
```

```
fig, ax = plt.subplots(figsize=(10, 6))
```

```
over_50.plot(kind='scatter',
             x="age",
             y="chol",
             c='target',
             ax=ax);
ax.set_xlim([45, 100]);
over_50.target.values
over_50.target.unique()

# Make a bit more of a complicated plot

# Create the plot
fig, ax = plt.subplots(figsize=(10, 6))

# Plot the data
scatter = ax.scatter(over_50["age"],
                    over_50["chol"],
                    c=over_50["target"])

# Customize the plot
ax.set(title="Heart Disease and Cholesterol Levels",
      xlabel="Age",
      ylabel="Cholesterol");

# Add a legend
ax.legend(*scatter.legend_elements(), title="Target");

# Add a horizontal line
```

```

ax.axhline(over_50["chol"].mean(), linestyle="--");

# Setup plot (2 rows, 1 column)
fig, (ax0, ax1) = plt.subplots(nrows=2, # 2 rows
                               ncols=1,
                               sharex=True,
                               figsize=(10, 8))

# Add data for ax0
scatter = ax0.scatter(x=over_50["age"],
                      y=over_50["chol"],
                      c=over_50["target"])

# Customize ax0
ax0.set(title="Heart Disease and Cholesterol Levels",
        #       xlabel="Age",
        ylabel="Cholesterol")
ax0.legend(*scatter.legend_elements(), title="Target")

# Setup a mean line
ax0.axhline(y=over_50["chol"].mean(),
            color='b',
            linestyle='--',
            label="Average")

# Add data for ax1
scatter = ax1.scatter(over_50["age"],
                      over_50["thalach"],
                      c=over_50["target"])

```

```
# Customize ax1
ax1.set(title="Heart Disease and Max Heart Rate Levels",
        xlabel="Age",
        ylabel="Max Heart Rate")
ax1.legend(*scatter.legend_elements(), title="Target")

# Setup a mean line
ax1.axhline(y=over_50["thalach"].mean(),
            color='b',
            linestyle='--',
            label="Average")

# Title the figure
fig.suptitle('Heart Disease Analysis', fontsize=16,
            fontweight='bold');
```