1. Первинний аналіз даних з Pandas ¶

1.1 Робота з векторами в бібліотеці NumPy

Numpy - це бібліотека Python для обчислювально ефективних операцій з багатовимірними масивами, призначена в основному для наукових обчислень.

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

In [ ]:
```

```

In [ ]:
```
a = np.array([0, 1, 2, 3])
a
```

Out[2]:
```
array([0, 1, 2, 3])
```

Такий масив може містити:

- значення фізичних величин в різні моменти часу при моделюванні
- значення сигналу, виміряного пристроєм
- інтенсивності пікселів
- 3D координати об'єктів, отримані, наприклад, при МРТ
- ...

Навіщо NumPy: Ефективність базових операцій

In [ ]:
```
L = range(1000)
```

In [ ]:
```
%timeit [i**2 for i in L]
```

```
1000 loops, best of 3: 255 µs per loop
```

In [ ]:
```
a = np.arange(1000)
```

In [ ]:
```
%timeit a**2
```

```
The slowest run took 45.94 times longer than the fastest. This could mean that
an intermediate result is being cached.
1000000 loops, best of 3: 1.41 µs per loop
```

Інтерактивна справка

In [ ]:
```
?np.array
```

пошук в документації

In [ ]:
```
np.lookfor('create array')
```

```
Search results for 'create array'
---------------------------------
numpy.array
    Create an array.
numpy.memmap
    Create a memory-map to an array stored in a *binary* file on disk.
numpy.diagflat
    Create a two-dimensional array with the flattened input as a diagonal.
numpy.fromiter
    Create a new 1-dimensional array from an iterable object.
numpy.partition
    Return a partitioned copy of an array.
numpy.ctypeslib.as_array
    Create a numpy array from a ctypes array or POINTER.
numpy.ma.diagflat
    Create a two-dimensional array with the flattened input as a diagonal.
numpy.ma.make_mask
    Create a boolean mask from an array.
numpy.lib.Arrayterator
```

In [ ]:
```
np.con*?
```

Бібліотеку принято імпортувати так

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

Створення масивів

- **1-D**:

In [ ]:
```
a = np.array([0, 1, 2, 3])
a
```

Out[12]:
```
array([0, 1, 2, 3])
```

In [ ]:
```
a.ndim
```

Out[13]:
```
1
```

In [ ]:
```
a.shape
```

Out[14]:
```
(4,)
```

In [ ]:
```
len(a)
```

Out[15]:
```
4
```

- **2-D, 3-D, ...**:

In [ ]:
```
b = np.array([[0, 1, 2], [3, 4, 5]])    # 2 x 3 array
b
```

Out[16]:
```
array([[0, 1, 2],
       [3, 4, 5]])
```

In [ ]:
```
b.ndim
```

Out[17]:
```
2
```

In [ ]:
```
b.shape
```

Out[18]:
```
(2, 3)
```

In [ ]:
```
len(b)     # returns the size of the first dimension
```

Out[19]:
```
2
```

In [ ]:
```
c = np.array([[[1], [2]], [[3], [4]]])
c
```

Out[20]:
```
array([[[1],
        [2]],

       [[3],
        [4]]])
```

In [ ]:
```
c.shape
```

Out[21]:
```
(2, 2, 1)
```

Методи для створення масивів

На практиці ми рідко додаємо елементи по одному

- Рівномірно розподілені елементи:

In [ ]:
```
a = np.arange(10) # 0 .. n-1  (!)
a
```

Out[22]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [ ]:
```
b = np.arange(1, 9, 2) # start, end (exclusive), step
b
```

Out[23]:
```
array([1, 3, 5, 7])
```

- за числом елементів:

In [ ]:
```
c = np.linspace(0, 1, 6)   # start, end, num-points
c
```

Out[24]:
```
array([0. , 0.2, 0.4, 0.6, 0.8, 1. ])
```

In [ ]:
```
d = np.linspace(0, 1, 5, endpoint=False)
d
```

Out[25]:
```
array([0. , 0.2, 0.4, 0.6, 0.8])
```

- Масиви, що часто зустрічаються:

In [ ]:
```
a = np.ones((3, 3))  # reminder: (3, 3) is a tuple
a
```

Out[26]:
```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

In [ ]:
```
b = np.zeros((2, 2))
b
```

Out[27]:
```
array([[0., 0.],
       [0., 0.]])
```

In [ ]:
```
c = np.eye(3)
c
```

Out[28]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [ ]:
```
d = np.diag(np.array([1, 2, 3, 4]))
d
```

Out[29]:
```
array([[1, 0, 0, 0],
       [0, 2, 0, 0],
       [0, 0, 3, 0],
       [0, 0, 0, 4]])
```

- `np.random` генерація випадкових чисел (Mersenne Twister PRNG):

In [ ]:
```
a = np.random.rand(4)       # uniform in [0, 1]
a
```

Out[30]:
```
array([0.50755507, 0.0211933 , 0.43352176, 0.44631306])
```

In [ ]:
```
b = np.random.randn(4)      # Gaussian
b
```

Out[31]:
```
array([ 0.65034618, -0.51433646,  0.53942869,  1.52676162])
```

In [ ]:
```
np.random.seed(1234)        # Setting the random seed
```

Основні типи даних NumPy

Точка після числа означає, що це тип даних `float64`

In [ ]:
```
a = np.array([1, 2, 3])
a.dtype
```

Out[33]:
```
dtype('int64')
```

In [ ]:
```
b = np.array([1., 2., 3.])
b.dtype
```

Out[34]:
```
dtype('float64')
```

Можна задати тип даних явно. За замовчуванням- `float64`

In [ ]:
```
c = np.array([1, 2, 3], dtype=float)
c.dtype
```

Out[35]:
```
dtype('float64')
```

In [ ]:
```
a = np.ones((3, 3))
a.dtype
```

Out[36]:
```
dtype('float64')
```

Інші типи даних:

- Комплексні числа

In [ ]:
```
d = np.array([1+2j, 3+4j, 5+6*1j])
d.dtype
```

Out[37]:
```
dtype('complex128')
```

- Bool

In [ ]:
```
e = np.array([True, False, False, True])
e.dtype
```

Out[38]:
```
dtype('bool')
```

- Рядки

На рядки пам'ять виділяється "жадібно" - за максимальною кількістю літер в рядку. В цьому прикладі на кожен рядок виділяється по 7 літер, і тип даних - 'S7'

In [ ]:
```
f = np.array(['Bonjour', 'Hello', 'Hallo',])
f.dtype     # <--- strings containing max. 7 letters
```

Out[39]:
```
dtype('<U7')
```

Основи візуалізації

$ ipython notebook --pylab=inline

Або з notebook:

In [ ]:
```
%pylab inline
```

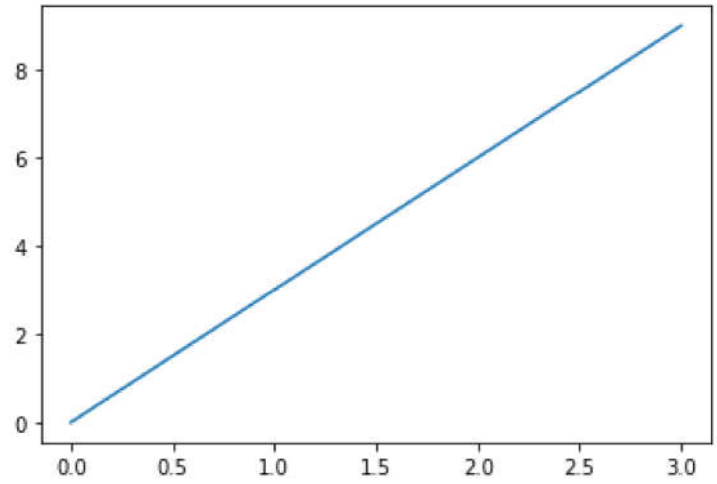
```
Populating the interactive namespace from numpy and matplotlib

/usr/local/lib/python3.6/dist-packages/IPython/core/magics/pylab.py:161: UserWa
rning: pylab import has clobbered these variables: ['e', 'f']
`%matplotlib` prevents importing * from pylab and numpy
  "\n`%matplotlib` prevents importing * from pylab and numpy"
```

Параметр `inline` говорить серверу IPython про те, що результати будуть відображатися в самій зошиті, а не в новому вікні.

Імпортуємо *Matplotlib*

In [ ]:
```
import matplotlib.pyplot as plt  # the tidy way
```

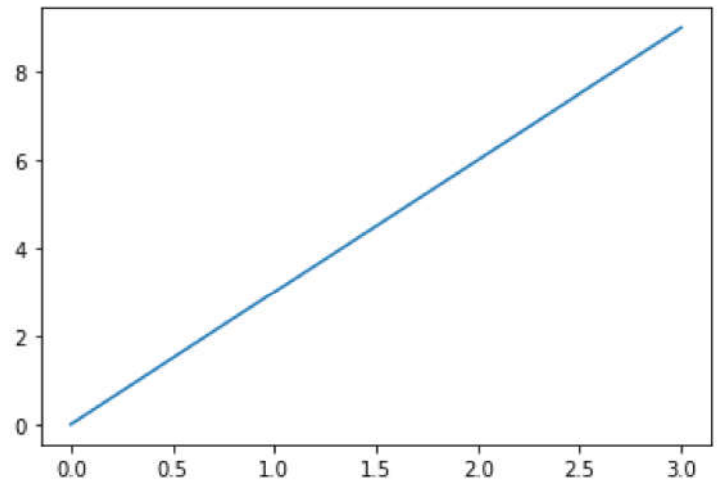
In [ ]:
```
x = np.linspace(0, 3, 20)
y = np.linspace(0, 9, 20)
plt.plot(x, y)       # line plot
plt.show()           # <-- shows the plot (not needed with pylab)
```



Або з використанням *pylab*:

In [ ]:
```
plot(x, y)       # line plot
```

Out[43]:
```
[<matplotlib.lines.Line2D at 0x7fa164027f60>]
```

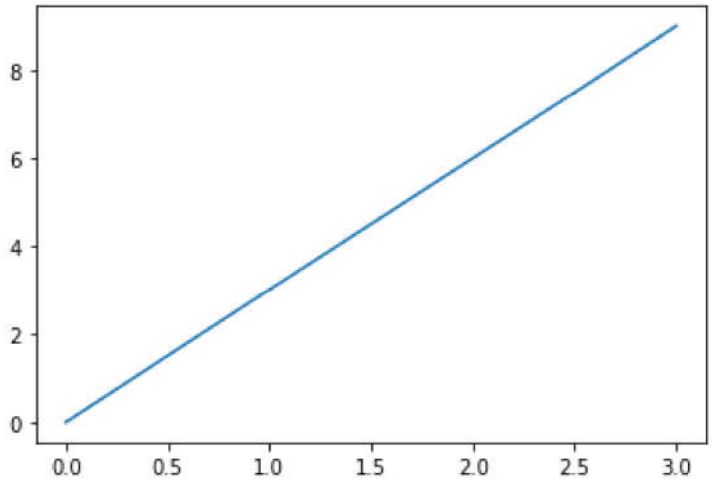


Використання `import matplotlib.pyplot as plt` рекомендується для скриптів, а `pylab` - в зошитах IPython.

- Відображення одновимірних масивів:

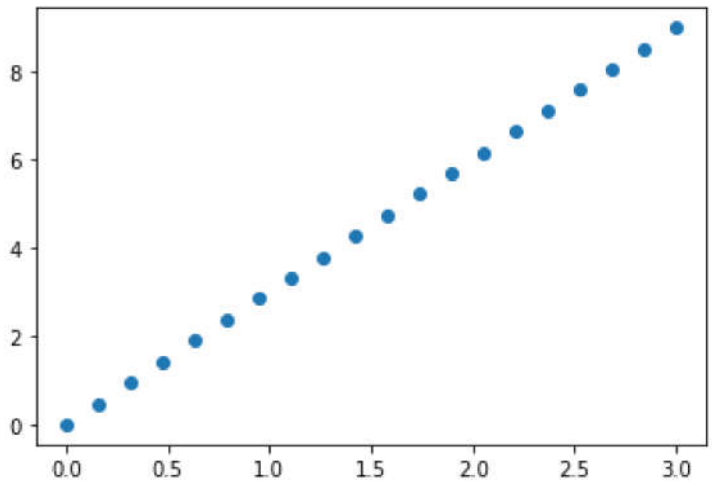
In [ ]:
```
x = np.linspace(0, 3, 20)
y = np.linspace(0, 9, 20)
plt.plot(x, y)       # line plot
```

Out[44]:
```
[<matplotlib.lines.Line2D at 0x7fa163c839b0>]
```



In [ ]:
```
plt.plot(x, y, 'o')  # dot plot
```

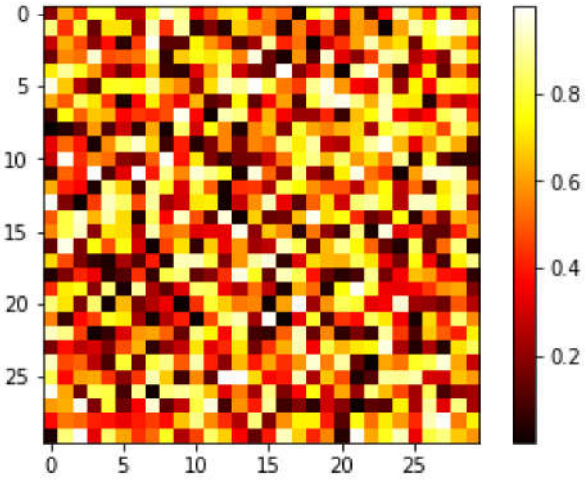
Out[45]:
```
[<matplotlib.lines.Line2D at 0x7fa163d380b8>]
```



- Відображення двохвимірних масивів (наприклад, зображень):

In [ ]:
```
image = np.random.rand(30, 30)
plt.imshow(image, cmap=plt.cm.hot)
plt.colorbar()
```

Out[46]:
```
<matplotlib.colorbar.Colorbar at 0x7fa16377eac8>
```



Індексування масивів і зрізи

В цілому так само, як з вбудованими послідовностями Python (наприклад, як зі списками).

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

Out[47]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [ ]:
```
a[0], a[2], a[-1]
```

Out[48]:
```
(0, 2, 9)
```

Працює і популярний в Python спосіб відображення масиву:

In [ ]:
```
a[::-1]
```

Out[49]:
```
array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
```

Для багатовимірних масивів індекси - це кортежі цілих чисел

In [ ]:
```
a = np.diag(np.arange(3))
a
```

Out[50]:
```
array([[0, 0, 0],
       [0, 1, 0],
       [0, 0, 2]])
```

In [ ]:
```
a[1, 1]
```

Out[51]:
```
1
```

In [ ]:
```
a[2, 1] = 10 # third line, second column
a
```

Out[52]:
```
array([[ 0,  0,  0],
       [ 0,  1,  0],
       [ 0, 10,  2]])
```

In [ ]:
```
a[1]
```

Out[53]:
```
array([0, 1, 0])
```

Зрізи

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

Out[54]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [ ]:
```
a[2:9:3] # [start:end:step]
```

Out[54]:
```
array([2, 5, 8])
```

Останній індекс не включається

In [ ]:
```
a[:4]
```

Out[55]:
```
array([0, 1, 2, 3])
```

За замовчуванням `start` - 0, `end` - індекс останнього елемента, `step` - 1:

In [ ]:
```
a[1:3]
```

Out[56]:
```
array([1, 2])
```

In [ ]:
```
a[::2]
```

Out[57]:
```
array([0, 2, 4, 6, 8])
```

In [ ]:
```
a[3:]
```

Out[58]:
```
array([3, 4, 5, 6, 7, 8, 9])
```

Можна поєднувати присвоювання і сріз:

In [ ]:
```
a = np.arange(10)
a[5:] = 10
a
```

Out[59]:
```
array([ 0,  1,  2,  3,  4, 10, 10, 10, 10, 10])
```

In [ ]:
```
b = np.arange(5)
a[5:] = b[::-1]
a
```

Out[60]:
```
array([0, 1, 2, 3, 4, 4, 3, 2, 1, 0])
```

Індексація масками

In [ ]:
```
np.random.seed(3)
a = np.random.random_integers(0, 20, 15)  #low, high, size
a
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:2: DeprecationWarn
ing: This function is deprecated. Please call randint(0, 20 + 1) instead
```

Out[61]:
```
array([10,  3,  8,  0, 19, 10, 11,  9, 10,  6,  0, 20, 12,  7, 14])
```

In [ ]:
```
(a % 3 == 0)
```

Out[62]:
```
array([False,  True, False,  True, False, False, False,  True, False,
        True,  True, False,  True, False, False])
```

In [ ]:
```
mask = (a % 3 == 0)
extract_from_a = a[mask] # or,  a[a%3==0]
extract_from_a           # extract a sub-array with the mask
```

Out[63]:
```
array([ 3,  0,  9,  6,  0, 12])
```

Індексація маскою може бути дуже корисною для присвоювання значень частині елементів масиву:

In [ ]:
```
a[a % 3 == 0] = -1
a
```

Out[64]:
```
array([10, -1,  8, -1, 19, 10, 11, -1, 10, -1, -1, 20, -1,  7, 14])
```

Індексація масивом цілих чисел

In [ ]:
```
a = np.arange(0, 100, 10)
a
```

Out[65]:
```
array([ 0, 10, 20, 30, 40, 50, 60, 70, 80, 90])
```

In [ ]:
```
a[[2, 3, 2, 4, 2]]  # note: [2, 3, 2, 4, 2] is a Python list
```

Out[66]:
```
array([20, 30, 20, 40, 20])
```

In [ ]:
```
a[[9, 7]] = -100
a
```

Out[67]:
```
array([   0,   10,   20,   30,   40,   50,   60, -100,   80, -100])
```

In [ ]:
```
a = np.arange(10)
idx = np.array([[3, 4], [9, 7]])
idx.shape
```

Out[68]:
```
(2, 2)
```

In [ ]:
```
a[idx]
```

Out[69]:
```
array([[3, 4],
       [9, 7]])
```