Wizualizacja Danych - Biblioteka NumPy

Ostatnia aktualizacja: 2022-04-02 17:09:57

NumPy

NumPy jest biblioteką Pythona służącą do obliczeń naukowych.

Zastosowania:

- algebra liniowa
- zaawansowane obliczenia matematyczne (numeryczne)
- całkowania
- rozwiązywanie równań
- ...

Import biblioteki NumPy

```
import numpy as np
```

Podstawowym bytem w bibliotece NumPy jest N-wymiarowa tablica zwana `ndarray`. Każdy element na tablicy traktowany jest jako typ `dtype`.

```
numpy.array(object, dtype=None, copy=True,
            order='K', subok=False, ndmin=0)
```

- object - to co ma być wrzucone do tablicy
- dtype - typ
- copy - czy obiekty mają być skopiowane, domyślne `True`
- order - sposób układania: C (rzędy), F (kolumny), A, K
- subok - realizowane przez podklasy (jeśli `True`), domyślnie `False`
- ndmin - minimalny rozmiar (wymiar) tablicy

```
import numpy as np

a = np.array([1, 2, 3])
print(a)
b = np.array([1, 2, 3.0])
print(b)
c = np.array([[1, 2], [3, 4]])
print(c)
d = np.array([1, 2, 3], ndmin=2)
print(d)
e = np.array([1, 2, 3], dtype=complex)
print(e)
```

```
f = np.array(np.mat('1 2; 3 4'))
print(f)
g = np.array(np.mat('1 2; 3 4'), subok=True)
print(g)
```

```
## [1 2 3]
## [1. 2. 3.]
## [[1 2]
##  [3 4]]
## [[1 2 3]]
## [1.+0.j 2.+0.j 3.+0.j]
## [[1 2]
##  [3 4]]
## [[1 2]
##  [3 4]]
```

Lista a tablica

```
import numpy as np
import time

start_time = time.time()
my_arr = np.arange(1000000)
my_list = list(range(1000000))
start_time = time.time()
my_arr2 = my_arr * 2
print("--- %s seconds ---" % (time.time() - start_time))
start_time = time.time()
my_list2 = [x * 2 for x in my_list]
print("--- %s seconds ---" % (time.time() - start_time))
```

```
## --- 0.0158383846282959 seconds ---
## --- 0.08654999732971191 seconds ---
```

Atrybuty tablic `ndarray`

Atrybut	Opis
`shape`	krotka z informacją liczbę elementów dla każdego z wymiarów
`size`	liczba elementów w tablicy (łączna)
`ndim`	liczba wymiarów tablicy
`nbytes`	liczba bajtów jaką tablica zajmuje w pamięci
`dtype`	typ danych

https://numpy.org/doc/stable/reference/arrays.ndarray.html#array-attributes

```
import numpy as np

tab1 = np.array([2, -3, 4, -8, 1])
print(type(tab1))
print(tab1.shape)
print(tab1.size)
print(tab1.ndim)
print(tab1.nbytes)
print(tab1.dtype)
```

```
## <class 'numpy.ndarray'>
## (5,)
## 5
## 1
## 20
## int32
```

```
import numpy as np

tab2 = np.array([[2, -3], [4, -8]])
print(type(tab2))
print(tab2.shape)
print(tab2.size)
print(tab2.ndim)
print(tab2.nbytes)
print(tab2.dtype)
tab3 = np.array([[2, -3], [4, -8, 5], [3]])
print(type(tab3))
print(tab3.shape)
print(tab3.size)
print(tab3.ndim)
print(tab3.nbytes)
print(tab3.dtype)
```

```
## <class 'numpy.ndarray'>
## (2, 2)
## 4
## 2
## 16
## int32
## <string>:1: VisibleDeprecationWarning: Creating an ndarray from ragged nested
## <class 'numpy.ndarray'>
## (3,)
## 3
## 1
## 24
## object
```

Typy danych

https://numpy.org/doc/stable/reference/arrays.scalars.html

https://numpy.org/doc/stable/reference/arrays.dtypes.html#arrays-dtypes-constructing

Typy całkowitoliczbowe	`int` , `int8` , `int16` , `int32` , `int64`
Typy całkowitoliczbowe (bez znaku)	`uint` , `uint8` , `uint16` , `uint32` , `uint64`
Typ logiczny	`bool`
Typy zmiennoprzecinkowe	`float` , `float16` , `float32` , `float64` , `float128`
Typy zmiennoprzecinkowe zespolone	`complex` , `complex64` , `complex128` , `complex256`
Napis	`str`

```
import numpy as np

tab = np.array([[2, -3], [4, -8]])
print(tab)
tab2 = np.array([[2, -3], [4, -8]], dtype=int)
print(tab2)
tab3 = np.array([[2, -3], [4, -8]], dtype=float)
print(tab3)
tab4 = np.array([[2, -3], [4, -8]], dtype=complex)
print(tab4)
```

```
## [[ 2 -3]
##  [ 4 -8]]
## [[ 2 -3]
##  [ 4 -8]]
## [[ 2. -3.]
##  [ 4. -8.]]
## [[ 2.+0.j -3.+0.j]
##  [ 4.+0.j -8.+0.j]]
```

Tworzenie tablic

`np.array` - argumenty rzutowany na tablicę (coś po czym można iterować) - warto sprawdzić rozmiar/kształt

```
import numpy as np

tab = np.array([2, -3, 4])
print(tab)
tab2 = np.array((4, -3, 3, 2))
```

```
print(tab2)
tab3 = np.array({3, 3, 2, 5, 2})
print(tab3)
tab4 = np.array({"pl": 344, "en": 22})
print(tab4)
```

```
## [ 2 -3  4]
## [ 4 -3  3  2]
## {2, 3, 5}
## {'pl': 344, 'en': 22}
```

`np.zeros` - tworzy tablicę wypełnioną zerami

```
import numpy as np

tab = np.zeros(4)
print(tab)
print(tab.shape)
tab2 = np.zeros([2, 3])
print(tab2)
print(tab2.shape)
tab3 = np.zeros([2, 3, 4])
print(tab3)
print(tab3.shape)
```

```
## [0. 0. 0. 0.]
## (4,)
## [[0. 0. 0.]
##  [0. 0. 0.]]
## (2, 3)
## [[[0. 0. 0. 0.]
##   [0. 0. 0. 0.]
##   [0. 0. 0. 0.]]
##
##  [[0. 0. 0. 0.]
##   [0. 0. 0. 0.]
##   [0. 0. 0. 0.]]]
## (2, 3, 4)
```

`np.ones` - tworzy tablicę wypełnioną jedynkami (to nie odpowiednik macierzy jednostkowej!)

```
import numpy as np

tab = np.ones(4)
print(tab)
print(tab.shape)
tab2 = np.ones([2, 3])
print(tab2)
print(tab2.shape)
```

```
tab3 = np.ones([2, 3, 4])
print(tab3)
print(tab3.shape)
```

```
## [1. 1. 1. 1.]
## (4,)
## [[1. 1. 1.]
##  [1. 1. 1.]]
## (2, 3)
## [[[1. 1. 1. 1.]
##   [1. 1. 1. 1.]
##   [1. 1. 1. 1.]]
##
##  [[1. 1. 1. 1.]
##   [1. 1. 1. 1.]
##   [1. 1. 1. 1.]]]
## (2, 3, 4)
```

`np.diag` - tworzy tablicę odpowiadającą macierzy diagonalnej

```
import numpy as np

tab0 = np.diag([3, 4, 5])
print(tab0)
tab1 = np.array([[2, 3, 4], [3, -4, 5], [3, 4, -5]])
print(tab1)
tab2 = np.diag(tab1)
print(tab2)
tab3 = np.diag(tab1, k=1)
print(tab3)
tab4 = np.diag(tab1, k=-2)
print(tab4)
tab5 = np.diag(np.diag(tab1))
print(tab5)
```

```
## [[3 0 0]
##  [0 4 0]
##  [0 0 5]]
## [[ 2  3  4]
##  [ 3 -4  5]
##  [ 3  4 -5]]
## [ 2 -4 -5]
## [3 5]
## [3]
## [[ 2  0  0]
##  [ 0 -4  0]
##  [ 0  0 -5]]
```

`np.arange` - tablica wypełniona równomiernymi wartościami

Składnia: `numpy.arange([start, ]stop, [step, ]dtype=None)`

```
import numpy as np

a = np.arange(3)
print(a)
b = np.arange(3.0)
print(b)
c = np.arange(3, 7)
print(c)
d = np.arange(3, 11, 2)
print(d)
e = np.arange(0, 1, 0.1)
print(e)
f = np.arange(3, 11, 2, dtype=float)
print(f)
g = np.arange(3, 10, 2)
print(g)
```

```
## [0 1 2]
## [0. 1. 2.]
## [3 4 5 6]
## [3 5 7 9]
## [0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9]
## [3. 5. 7. 9.]
## [3 5 7 9]
```

`np.linspace` - tablica wypełniona równomiernymi wartościami wg skali liniowej

```
import numpy as np

a = np.linspace(2.0, 3.0, num=5)
print(a)
b = np.linspace(2.0, 3.0, num=5, endpoint=False)
print(b)
c = np.linspace(2.0, 3.0, num=5, retstep=True)
print(c)
```

```
## [2.   2.25 2.5  2.75 3.  ]
## [2.  2.2 2.4 2.6 2.8]
## (array([2.  , 2.25, 2.5 , 2.75, 3.  ]), 0.25)
```

`np.logspace` - tablica wypełniona wartościami wg skali logarytmicznej

Składnia:
`numpy.logspace(start, stop, num=50, endpoint=True, base=10.0, dtype=None, axis=0)`

```
import numpy as np
```

```
a = np.logspace(2.0, 3.0, num=4)
print(a)
b = np.logspace(2.0, 3.0, num=4, endpoint=False)
print(b)
c = np.logspace(2.0, 3.0, num=4, base=2.0)
print(c)
```

```
## [ 100.          215.443469    464.15888336 1000.        ]
## [100.         177.827941   316.22776602 562.34132519]
## [4.         5.0396842  6.34960421 8.        ]
```

`np.empty` - pusta (niezaincjowana) tablica

```
import numpy as np

a = np.empty(3)
print(a)
b = np.empty(3, dtype=int)
print(b)
```

```
## [0. 1. 2.]
## [0 1 2]
```

`np.identity` - tablica przypominająca macierz jednostkową

`np.eye` - tablica z jedynkami na przekątnej (pozostałe zera)

```
import numpy as np

a = np.identity(4)
print(a)
b = np.eye(4, k=1)
print(b)
c = np.eye(4, k=2)
print(c)
d = np.eye(4, k=-1)
print(d)
```

```
## [[1. 0. 0. 0.]
##  [0. 1. 0. 0.]
##  [0. 0. 1. 0.]
##  [0. 0. 0. 1.]]
## [[0. 1. 0. 0.]
##  [0. 0. 1. 0.]
##  [0. 0. 0. 1.]
##  [0. 0. 0. 0.]]
## [[0. 0. 1. 0.]
##  [0. 0. 0. 1.]
##  [0. 0. 0. 0.]
```

```
##  [0. 0. 0. 0.]]
## [[0. 0. 0. 0.]
##  [1. 0. 0. 0.]
##  [0. 1. 0. 0.]
##  [0. 0. 1. 0.]]
```

Indeksowanie, “krojenie”

```
import numpy as np

a = np.array([2, 5, -2, 4, -7, 8, 9, 11, -23, -4, -7, 8, 1])
print(a[5])
print(a[-2])
print(a[3:6])
print(a[:])
print(a[0:-1])
print(a[:5])
print(a[4:])
print(a[4:-1])
print(a[4:10:2])
print(a[::-1])
print(a[::2])
print(a[::-2])
```

```
## 8
## 8
## [ 4 -7  8]
## [  2   5  -2   4  -7   8   9  11 -23  -4  -7   8   1]
## [  2   5  -2   4  -7   8   9  11 -23  -4  -7   8]
## [ 2  5 -2  4 -7]
## [ -7   8   9  11 -23  -4  -7   8   1]
## [ -7   8   9  11 -23  -4  -7   8]
## [ -7   9 -23]
## [  1   8  -7  -4 -23  11   9   8  -7   4  -2   5   2]
## [  2  -2  -7   9 -23  -7   1]
## [  1  -7 -23   9  -7  -2   2]
```

```
import numpy as np

a = np.array([[3, 4, 5], [-3, 4, 8], [3, 2, 9]])
b = a[:2, 1:]
print(b)
print(np.shape(b))
c = a[1]
print(c)
print(np.shape(c))
d = a[1, :]
print(d)
print(np.shape(d))
e = a[1:2, :]
```

```
print(e)
print(np.shape(e))
f = a[:, :2]
print(f)
print(np.shape(f))
g = a[1, :2]
print(g)
print(np.shape(g))
h = a[1:2, :2]
print(h)
print(np.shape(h))
```

```
## [[4 5]
##  [4 8]]
## (2, 2)
## [-3  4  8]
## (3,)
## [-3  4  8]
## (3,)
## [[-3  4  8]]
## (1, 3)
## [[ 3  4]
##  [-3  4]
##  [ 3  2]]
## (3, 2)
## [-3  4]
## (2,)
## [[-3  4]]
## (1, 2)
```

**Uwaga - takie “krojenie” to tzw “widok”.

```
import numpy as np

a = np.array([[3, 4, 5], [-3, 4, 8], [3, 2, 9]])
b = a[1:2, 1:]
print(b)
a[1][1] = 9
print(a)
print(b)
b[0][0] = -11
print(a)
print(b)
```

```
## [[4 8]]
## [[ 3  4  5]
##  [-3  9  8]
##  [ 3  2  9]]
## [[9 8]]
## [[  3   4   5]
##  [ -3 -11   8]
```

```
##  [  3   2   9]]
## [[-11   8]]
```

Naprawa:

```
import numpy as np

a = np.array([[3, 4, 5], [-3, 4, 8], [3, 2, 9]])
b = a[1:2, 1:].copy()
print(b)
a[1][1] = 9
print(a)
print(b)
b[0][0] = -11
print(a)
print(b)
```

```
## [[4 8]]
## [[ 3  4  5]
##  [-3  9  8]
##  [ 3  2  9]]
## [[4 8]]
## [[ 3  4  5]
##  [-3  9  8]
##  [ 3  2  9]]
## [[-11   8]]
```

Indeksowanie logiczne (fancy indexing)

```
import numpy as np

a = np.array([2, 5, -2, 4, -7, 8, 9, 11, -23, -4, -7, 8, 1])
b = a[np.array([1, 3, 7])]
print(b)
c = a[[1, 3, 7]]
print(c)
```

```
## [ 5  4 11]
## [ 5  4 11]
```

```
import numpy as np

a = np.array([2, 5, -2, 4, -7, 8, 9, 11, -23, -4, -7, 8, 1])
b = a > 0
print(b)
c = a[a > 0]
print(c)
```

```
## [ True  True False  True False  True  True  True False False False  True
##   True]
## [ 2  5  4  8  9 11  8  1]
```

```
import numpy as np

a = np.array([2, 5, -2, 4, -7, 8, 9, 11, -23, -4, -7, 8, 1])
b = a[a > 0]
print(b)
b[0] = -5
print(a)
print(b)
a[1] = 20
print(a)
print(b)
```

```
## [ 2  5  4  8  9 11  8  1]
## [  2   5  -2   4  -7   8   9  11 -23  -4  -7   8   1]
## [-5  5  4  8  9 11  8  1]
## [  2  20  -2   4  -7   8   9  11 -23  -4  -7   8   1]
## [-5  5  4  8  9 11  8  1]
```

Modyfikacja kształtu i rozmiaru

https://numpy.org/doc/stable/reference/routines.array-manipulation.html

```
import numpy as np

a = np.array([[3, 4, 5], [-3, 4, 8], [3, 2, 9]])
b = np.reshape(a, (1, 9))
print(b)
c = a.reshape(9)
print(c)
d = a.flatten()
print(d)
e = a.ravel()
print(e)
f = np.ravel(a)
print(f)
g = [[1, 3, 4]]
h = np.squeeze(g)
print(h)
i = a.T
print(i)
j = np.transpose(a)
print(j)
k = np.hstack((h, h, h))
print(k)
l = np.vstack((h, h, h))
print(l)
```

```
m = np.dstack((h, h, h))
print(m)
```

```
## [[ 3  4  5 -3  4  8  3  2  9]]
## [ 3  4  5 -3  4  8  3  2  9]
## [ 3  4  5 -3  4  8  3  2  9]
## [ 3  4  5 -3  4  8  3  2  9]
## [ 3  4  5 -3  4  8  3  2  9]
## [1 3 4]
## [[ 3 -3  3]
##  [ 4  4  2]
##  [ 5  8  9]]
## [[ 3 -3  3]
##  [ 4  4  2]
##  [ 5  8  9]]
## [1 3 4 1 3 4 1 3 4]
## [[1 3 4]
##  [1 3 4]
##  [1 3 4]]
## [[[1 1 1]
##   [3 3 3]
##   [4 4 4]]]
```

```
import numpy as np

a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6]])
r1 = np.concatenate((a, b))
print(r1)
r2 = np.concatenate((a, b), axis=0)
print(r2)
r3 = np.concatenate((a, b.T), axis=1)
print(r3)
r4 = np.concatenate((a, b), axis=None)
print(r4)
```

```
## [[1 2]
##  [3 4]
##  [5 6]]
## [[1 2]
##  [3 4]
##  [5 6]]
## [[1 2 5]
##  [3 4 6]]
## [1 2 3 4 5 6]
```

```
import numpy as np

a = np.array([[1, 2], [3, 4]])
r1 = np.resize(a, (2, 3))
```

```
print(r1)
r2 = np.resize(a, (1, 4))
print(r2)
r3 = np.resize(a, (2, 4))
print(r3)
```

```
## [[1 2 3]
##  [4 1 2]]
## [[1 2 3 4]]
## [[1 2 3 4]
##  [1 2 3 4]]
```

```
import numpy as np

a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6]])
r1 = np.append(a, b)
print(r1)
r2 = np.append(a, b, axis=0)
print(r2)
```

```
## [1 2 3 4 5 6]
## [[1 2]
##  [3 4]
##  [5 6]]
```

```
import numpy as np

a = np.array([[1, 2], [3, 7]])
r1 = np.insert(a, 1, 4)
print(r1)
r2 = np.insert(a, 2, 4)
print(r2)
r3 = np.insert(a, 1, 4, axis=0)
print(r3)
r4 = np.insert(a, 1, 4, axis=1)
print(r4)
```

```
## [1 4 2 3 7]
## [1 2 4 3 7]
## [[1 2]
##  [4 4]
##  [3 7]]
## [[1 4 2]
##  [3 4 7]]
```

```
import numpy as np
```

```
a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
r1 = np.delete(a, 1, axis=1)
print(r1)
r2 = np.delete(a, 2, axis=0)
print(r2)
```

```
## [[ 1  3  4]
##  [ 5  7  8]
##  [ 9 11 12]]
## [[1 2 3 4]
##  [5 6 7 8]]
```

Broadcasting

Wariant 1 - skalar-tablica - wykonanie operacji na każdym elemencie tablicy

```
import numpy as np

a = np.array([[1, 2], [5, 6], [9, 10]])
b = a + 4
print(b)
c = 2 ** a
print(c)
```

```
## [[ 5  6]
##  [ 9 10]
##  [13 14]]
## [[   2    4]
##  [  32   64]
##  [ 512 1024]]
```

Wariant 2 - dwie tablice - “gdy jedna z tablic może być rozszerzona” (oba wymiary są równe lub jeden z nich jest równy 1)

https://numpy.org/doc/stable/user/basics.broadcasting.html

```
import numpy as np

a = np.array([[1, 2], [5, 6]])
b = np.array([9, 2])
r1 = a + b
print(r1)
r2 = a / b
print(r2)
c = np.array([[4], [-2]])
r3 = a + c
print(r3)
r4 = c / a
print(r4)
```

```
## [[10  4]
##  [14  8]]
## [[0.11111111 1.        ]
##  [0.55555556 3.        ]]
## [[5 6]
##  [3 4]]
## [[ 4.          2.        ]
##  [-0.4        -0.33333333]]
```

Funkcje uniwersalne

https://numpy.org/doc/stable/reference/ufuncs.html#methods

Statystyka i agregacja

Funkcja	Opis
np.mean	Średnia wszystkich wartości w tablicy.
np.std	Odchylenie standardowe.
np.var	Wariancja.
np.sum	Suma wszystkich elementów.
np.prod	Iloczyn wszystkich elementów.
np.cumsum	Skumulowana suma wszystkich elementów.
np.cumprod	Skumulowany iloczyn wszystkich elementów.
np.min,np.max	Minimalna/maksymalna wartość w tablicy.
np.argmin, np.argmax	Indeks minimalnej/maksymalnej wartości w tablicy.
np.all	Sprawdza czy wszystki elementy są różne od zera.
np.any	Sprawdza czy co najmniej jeden z elementów jest różny od zera.

Wyrażenia warunkowe

https://numpy.org/doc/stable/reference/generated/numpy.where
https://numpy.org/doc/stable/reference/generated/numpy.choose
https://numpy.org/doc/stable/reference/generated/numpy.select
https://numpy.org/doc/stable/reference/generated/numpy.nonzero

Działania na zbiorach

https://numpy.org/doc/stable/reference/routines.set.html

Operacje tablicowe

https://numpy.org/doc/stable/reference/generated/numpy.transpose

https://numpy.org/doc/stable/reference/generated/numpy.flip
https://numpy.org/doc/stable/reference/generated/numpy.fliplr
https://numpy.org/doc/stable/reference/generated/numpy.flipud

https://numpy.org/doc/stable/reference/generated/numpy.sort

Alegbra liniowa

https://numpy.org/doc/stable/reference/routines.linalg.html

Funkcja na stringach

https://numpy.org/doc/stable/reference/routines.char.html

Data i czas

https://numpy.org/doc/stable/reference/arrays.datetime.html

Bibliografia:

- Dokumentacja biblioteki, https://numpy.org/doc/stable/, dostęp online 5.03.2021.
- Robert Jahansson, Matematyczny Python. Obliczenia naukowe i analiza danych z użyciem NumPy, SciPy i Matplotlib, Wyd. Helion, 2021.
- https://www.tutorialspoint.com/numpy/index.htm, dostęp online 20.03.2019.