02_numpy_filled

October 18, 2020

1 NumPy

NumPy (oder Numpy) ist eine lineare Algebra-Bibliothek für Python. Sehr wichtig, da fast alle Bibliotheken im PyData-Ökosystem auf NumPy als einen ihrer Hauptbausteine vertrauen.

Numpy ist außerdem unglaublich schnell, da es Bindungen an C-Bibliotheken hat. **Warum Numpy anstelle von Python-Arrays -> gerne selbst recherchieren! ;)**

Hier nur eine Einführung in die Grundlagen von Numpy. - Schritt Nr. 1 -> Installieren

via poetry: `poetry add numpy`

direkt via pip: `pip install numpy`

1.1 Verwendung von Numpy

Numpy muss zuerst importiert werden! Typischerweise importieren wir NumPy als **np**

[36]:
```
import numpy as np
```

Numpy ist sehr mächtig. Wir werden hier allerdings nur die für uns wichtigen Konzepte behandeln: - Vektoren - Arrays - Matrizen - Zufallszahlengenerierung

Numpy Arrays sind für uns am wichtigsten. Numpy Arrays gibt es im Wesentlichen in zwei Varianten: **Vektoren und Matrizen.** Vektoren sind eindimensionale Arrays und Matrizen sind zweidimensionale Arrays (eine Matrix kann dennoch immer noch nur eine Zeile oder eine Spalte haben!).

Numpy Arrays erstellen Wir könnne eine Python-Liste nehmen und sie in ein Numpy Array umwandeln.

[37]:
```
liste = [1,2,3,4,5]
```

[38]:
```
liste
```

[38]:
```
[1, 2, 3, 4, 5]
```

[39]:
```
np.array(liste)
```

[39]:
```
array([1, 2, 3, 4, 5])
```

[40]:
```
matrix = [[1,2,3],[4,5,6],[7,8,9]]
matrix
```

[40]:
```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

[41]:
```
np.array(matrix)
```

[41]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

Es gibt viele verschiedene Möglichkeiten ein Array automatisch erstellen zu lassen mit bereits vorhandenen Methoden in Numpy!

arange Liefert gleichmäßig verteilte Werte (d.h. Stützstellen) innerhalb eines bestimmten **halboffenen** Intervalls **[start, stop)** mit einem in **steps** bestimmten Abstand und folgt dabei dem Syntax: np.arrange(start, stop, steps)

Achtung in Matlab und R: abgeschlossenes Intervall

[42]:
```
np.arange(0,10)
```

[42]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

[43]:
```
np.arange(2,11,2)
```

[43]:
```
array([ 2,  4,  6,  8, 10])
```

zeros and ones Erzeugt Arrays von Nullen oder Einsen!

[44]:
```
np.zeros(10)
```

[44]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

[45]:
```
np.zeros((5,5))
```

[45]:
```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

[46]:
```
np.ones(3)
```

[46]:
```
array([1., 1., 1.])
```

[47]:
```
np.ones((3,3))
```

[47]:
```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

linspace Liefert gleichmäßig verteilte Zahlen (d.h. äquidistante Stützstellen) über ein bestimmtes Intervall. `linspace` legt den Fokus auf die Anzahl der Stützstellen, `arange` dahingegen auf die Abstände.

[48]:
```
np.linspace(0,10,10)
```

[48]:
```
array([ 0.        ,  1.11111111,  2.22222222,  3.33333333,  4.44444444,
        5.55555556,  6.66666667,  7.77777778,  8.88888889, 10.        ])
```

eye Erstellt eine Identitätsmatrix mit 1 auf der Diagonalen und 0 sonst.

[49]:
```
np.eye(4)
```

[49]:
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

1.1.1 Random

Numpy bietet viele verschiedene Möglichkeiten ein Array mit Zufallszahlen zu erzeugen. Es handelt sich dabei im **Pseudozufallszahlen**, die von einem Zufallszahlengenerator `np.random.default_rng` erzeugt werden.

random Erstellt ein Array der gegebenen Form und füllet es mit Stichproben aus einer gleichmäßigen Verteilung über[0, 1).

[50]:
```
rng = np.random.default_rng()
rng.random(2)
```

[50]:
```
array([0.15076596, 0.66170719])
```

[51]:
```
rng.random((3,5))
```

[51]:
```
array([[0.25250385, 0.4993074 , 0.56469033, 0.52779184, 0.41616422],
       [0.892422  , 0.17956504, 0.55816816, 0.24803367, 0.30034867],
       [0.86812971, 0.16861169, 0.24815972, 0.12686045, 0.29908154]])
```

randn Stichprobe von der Normalverteilung

[52]:
```
rng.normal(2)
```

[52]:
```
2.191854215797941
```

[53]:
```
rng.normal(loc=10, scale=2, size=5)
```

[53]:
```
array([ 7.64953892,  9.09113209,  9.51894756,  9.56655299, 10.41388169])
```

randint Liefert zufällige Ganzzahlen von niedrig (inklusive) bis hoch (exklusiv).

[54]:
```
rng.integers(1,100)
```

[54]:
```
6
```

[55]:
```
rng.integers(low=1, high=100, size=10)
```

[55]:
```
array([33, 81, 37, 11,  2, 43, 21, 91, 30, 46])
```

Weitere Verteilungsbeispiele * beta(a, b, size) * binomial(n, p, size) * chisquare(degree_of_freedom, size) * poisson(lamda, size)

1.1.2 Array Attribute und Methoden

[56]:
```
arr = np.arange(25)
ranarr = rng.integers(0,50,10)
```

[57]:
```
arr
```

[57]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24])
```

[58]:
```
ranarr
```

[58]:
```
array([37, 11, 25, 36, 13, 23, 10,  0,  5, 11])
```

Reshape Ändert die Form eines Arrays

numpy.reshape(a, newshape, order='C') * **a**: array to be reshabed * **newshape**: int (then 1-D array) or tuple of integers; eindimensional kann durch -1 erreicht werden * **order**: * **'C'** letzter Index wechselst am schnellsten bis zum ersten Index wechselt am langsamsten * **'F'** erster Index wechselt am schnelslten bis zum letzten Index wechselt am langsamsten

[59]:
```
arr.reshape(5,5)
```

[59]:
```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
```

max, min, argmax, argmin Dies sind nützliche Methoden zum Auffinden von Maximal- oder Minimalwerten. Oder um ihre Indexpositionen mit argmin oder argmax zu finden.

[60]:
```
ranarr.max()
```

[60]:
```
37
```

[61]:
```
ranarr.argmax()
```

[61]:
```
0
```

[62]:
```
ranarr.min()
```

[62]:
```
0
```

[63]:
```
ranarr.argmin()
```

[63]:
```
7
```

Form (Shape) Die Form ist ein Attribut oder Eigenschaft eines Arrays und gibt als Tuple die Dimensionen zurück

[64]:
```
arr.shape
```

[64]:
```
(25,)
```

[65]:
```
arr.reshape(1,25)
```

[65]:
```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15,
        16, 17, 18, 19, 20, 21, 22, 23, 24]])
```

[66]:
```
arr.reshape(1,25).shape
```

[66]:
```
(1, 25)
```

[67]:
```
arr.reshape(25,1)
```

[67]:
```
array([[ 0],
       [ 1],
       [ 2],
```

```
       [ 3],
       [ 4],
       [ 5],
       [ 6],
       [ 7],
       [ 8],
       [ 9],
       [10],
       [11],
       [12],
       [13],
       [14],
       [15],
       [16],
       [17],
       [18],
       [19],
       [20],
       [21],
       [22],
       [23],
       [24]])
```

[68]:
```
arr.reshape(25,1).shape
```

[68]:
```
(25, 1)
```

[69]:
```
# Als Vektor
arr.reshape(-1)
```

[69]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24])
```

[70]:
```
arr.reshape(-1).shape
```

[70]:
```
(25,)
```

dtype Gibt an welcher Datentyp sich im Array befindet

[71]:
```
arr.dtype
```

[71]:
```
dtype('int64')
```

1.2 NumPy Indexing und Selection

[72]:
```
arr[4]
```

[72]:
```
4
```

[73]:
```
arr[1:5]
```

[73]:
```
array([1, 2, 3, 4])
```

1.2.1 Broadcasting

Numpy Arrays unterscheiden sich von einer normalen Python-Liste durch ihre Fähigkeit zum "broadcasten". Dabei wird das kleinere Array entlang des größeren "gebroadcasted", sodass die Arraygrößen passend sind.

[74]:
```
arr[0:6]=100
```

[75]:
```
arr
```

[75]:
```
array([100, 100, 100, 100, 100, 100,   6,   7,   8,   9,  10,  11,  12,
        13,  14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24])
```

[76]:
```
# Neu initialisieren um den Effekt zu veranschaulichen
arr = np.arange(0,11)

arr
```

[76]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

[77]:
```
slice_of_arr = arr[0:6]
slice_of_arr
```

[77]:
```
array([0, 1, 2, 3, 4, 5])
```

[78]:
```
#Ausschnitt ändern
slice_of_arr[:]=66

slice_of_arr
```

[78]:
```
array([66, 66, 66, 66, 66, 66])
```

Änderung auch in dem Orginal Array zu erkennen -> Numpy macht keine Kopien sondern arbeitet an dem Original (Vermeidung von Speicher-Problemen)

[79]:
```
arr
```

[79]:
```
array([66, 66, 66, 66, 66, 66,  6,  7,  8,  9, 10])
```

[80]:
```
# So kann man eine Kopie erstellen
arr_copy = arr.copy()

arr_copy
```

[80]:
```
array([66, 66, 66, 66, 66, 66,  6,  7,  8,  9, 10])
```

1.2.2 Indexing zweidimensionales Array

Array[row][col] oder Array[row,col]

[81]:
```
arr_2d = np.array(([5,10,15],[20,25,30],[35,40,45]))
```

[82]:
```
arr_2d[1]
```

[82]:
```
array([20, 25, 30])
```

[83]:
```
arr_2d[1][0]
```

[83]:
```
20
```

[84]:
```
arr_2d[1,0]
```

[84]:
```
20
```

[85]:
```
#Matrix
arr2d = np.zeros((10,10))
```

[86]:
```
#Mit Daten befüllen

for i in range(len(arr2d)):
    arr2d[i] = i

arr2d
```

[86]:
```
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [2., 2., 2., 2., 2., 2., 2., 2., 2., 2.],
       [3., 3., 3., 3., 3., 3., 3., 3., 3., 3.],
       [4., 4., 4., 4., 4., 4., 4., 4., 4., 4.],
       [5., 5., 5., 5., 5., 5., 5., 5., 5., 5.],
       [6., 6., 6., 6., 6., 6., 6., 6., 6., 6.],
       [7., 7., 7., 7., 7., 7., 7., 7., 7., 7.],
       [8., 8., 8., 8., 8., 8., 8., 8., 8., 8.],
```

```
       [9., 9., 9., 9., 9., 9., 9., 9., 9., 9.]])
```

[87]:
```
#Auch nur ein Teil daraus (bestimmte Reihen)
arr2d[[1,2,6,8]]
```

[87]:
```
array([[1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [2., 2., 2., 2., 2., 2., 2., 2., 2., 2.],
       [6., 6., 6., 6., 6., 6., 6., 6., 6., 6.],
       [8., 8., 8., 8., 8., 8., 8., 8., 8., 8.]])
```

1.2.3 Selection

[88]:
```
arr = np.arange(1,11)
arr
```

[88]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

[89]:
```
arr > 4
```

[89]:
```
array([False, False, False, False,  True,  True,  True,  True,  True,
       True])
```

[90]:
```
bool_arr = arr>4
```

[91]:
```
bool_arr
```

[91]:
```
array([False, False, False, False,  True,  True,  True,  True,  True,
       True])
```

[92]:
```
arr[bool_arr]
```

[92]:
```
array([ 5,  6,  7,  8,  9, 10])
```

[93]:
```
arr[arr>2]
```

[93]:
```
array([ 3,  4,  5,  6,  7,  8,  9, 10])
```

[94]:
```
x = 2
arr[arr>x]
```

[94]:
```
array([ 3,  4,  5,  6,  7,  8,  9, 10])
```

1.3 Numpy Operationen

Arithmetik

[95]:
```
arr = np.arange(0,10)
```

[96]:
```
arr
```

[96]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

[97]:
```
arr + arr
```

[97]:
```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
```

[98]:
```
# Warnung division by zero. Aber kein Error!
arr/arr
```

[98]:
```
array([nan,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.])
```

[99]:
```
# Auch nur eine Warnung kein Error (infinity)
1/arr
```

[99]:
```
array([       inf, 1.        , 0.5       , 0.33333333, 0.25      ,
       0.2       , 0.16666667, 0.14285714, 0.125     , 0.11111111])
```

[100]:
```
arr**4
```

[100]:
```
array([   0,    1,   16,   81,  256,  625, 1296, 2401, 4096, 6561])
```

Andere Numpy Operationen

[101]:
```
# Wurzel ziehen
np.sqrt(arr)
```

[101]:
```
array([0.        , 1.        , 1.41421356, 1.73205081, 2.        ,
       2.23606798, 2.44948974, 2.64575131, 2.82842712, 3.        ])
```

[102]:
```
# e^ berechnen
np.exp(arr)
```

[102]:
```
array([1.00000000e+00, 2.71828183e+00, 7.38905610e+00, 2.00855369e+01,
       5.45981500e+01, 1.48413159e+02, 4.03428793e+02, 1.09663316e+03,
       2.98095799e+03, 8.10308393e+03])
```

[103]:
```
#Logarithmus
np.log(arr)
```

[103]:
```
array([      -inf, 0.        , 0.69314718, 1.09861229, 1.38629436,
       1.60943791, 1.79175947, 1.94591015, 2.07944154, 2.19722458])
```