Python 的 50+ 練習： 資料科學學習手冊

> *資料科學模組 NumPy 入門*

數據交點 | 郭耀仁 https://linktr.ee/yaojenkuo

這個章節會登場的模組

`numpy` 模組。

關於 NumPy

什麼是 NumPy

> *NumPy 是 Numeric Python 的簡稱，是 Python 最重要的資料科學模組之一。NumPy 創造了 `ndarray` 的資料結構類別以及大量的通用函數與聚合函數，讓 Python 使用者能夠對進行快速的數值計算、使用統計函數、進行線性代數運算以及操作隨機的模擬任務等。*

來源：https://numpy.org/

（沒什麼用的冷知識）NumPy 的前身為兩個模組

- 1990 年代中期誕生的 `Numeric` 與 `Numarray` 模組。
- NumPy 於 2005 集兩者之大成問世。

來源：https://www.nature.com/articles/s41586-020-2649-2

根據說明文件的範例載入

來源：https://numpy.org/doc/stable/user/absolute_beginners.html#how-to-import-numpy

In [1]:

```
import numpy as np
```

如果環境中沒有安裝 `numpy`，載入時會遭遇 `ModuleNotFoundError`

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'numpy'
```

如果遭遇 `ModuleNotFoundError` 可以在終端機使用 `pip install numpy` 或者 `conda install numpy` 指令安裝

若要指定模組版本可以加上 `==MAJOR.MINOR.PATCH` 課程使用的模組版本為 1.21

```
pip install numpy==1.21
```

或者

```
conda install numpy==1.21
```

可以透過兩個屬性檢查版本與安裝路徑

- `__version__` 屬性檢查版本號。
- `__file__` 屬性檢查安裝路徑。

In [2]:

```
print(np.__version__)
print(np.__file__)
```

```
1.21.0
/srv/conda/envs/notebook/lib/python3.9/site-packages/numpy/__init__.py
```

NumPy 的核心功能

1. 使用 `ndarray` 來進行數值操作。
2. 使用模組定義的函數對 `ndarray` 進行數值運算。
3. `ndarray` 是其他資料科學模組 Pandas、Matplotlib 與 Scikit-Learn 的基石。

如何創造 `ndarray`

從 `list` 創造 `ndarray`

In [3]:

```
prime_list = [2, 3, 5, 7, 11]
prime_array = np.array(prime_list)
print(prime_array)
print(type(prime_array))
```

```
[ 2  3  5  7 11]
<class 'numpy.ndarray'>
```

利用 NumPy 函數創造內容元素相同的 `ndarray`

- `np.zeros()`
- `np.ones()`
- `np.full()`

In [4]:

```
print(np.zeros(5, dtype=int))
print(np.ones(5, dtype=float))
print(np.full(5, 6))
```

```
[0 0 0 0 0]
[1. 1. 1. 1. 1.]
[6 6 6 6 6]
```

利用 NumPy 函數創造數列型態的 `ndarray`

- `np.arange(start, stop, step)`
- `np.linspace(start, stop, num)` 值得注意的是 `stop` 參數預設為包含。

In [5]:

```
print(np.arange(1, 11, 2))
print(np.linspace(1, 9, 5, dtype=int))
```

```
[1 3 5 7 9]
[1 3 5 7 9]
```

利用 `np.random` 中的函數創造隨機性的 `ndarray`

In [6]:

```
m = 10000
uniform_array = np.random.random(m) # module.function()
normal_array = np.random.normal(0, 1, m)
randint_array = np.random.randint(1, 7, size=m)
print(uniform_array)
print(normal_array)
print(randint_array)
```

```
[0.56090609 0.84319229 0.10549993 ... 0.5769678  0.4578
3439 0.61463416]
[-0.40019236  0.79275922 -1.04887322 ...  0.28967493
0.26388343
 -0.05786187]
[2 3 3 ... 1 2 4]
```

常用的 `ndarray` 屬性

- `ndarray.ndim` 維度數。
- `ndarray.shape` 外型。
- `ndarray.size` 元素個數。
- `ndarray.dtype` 資料類別。

In [7]:

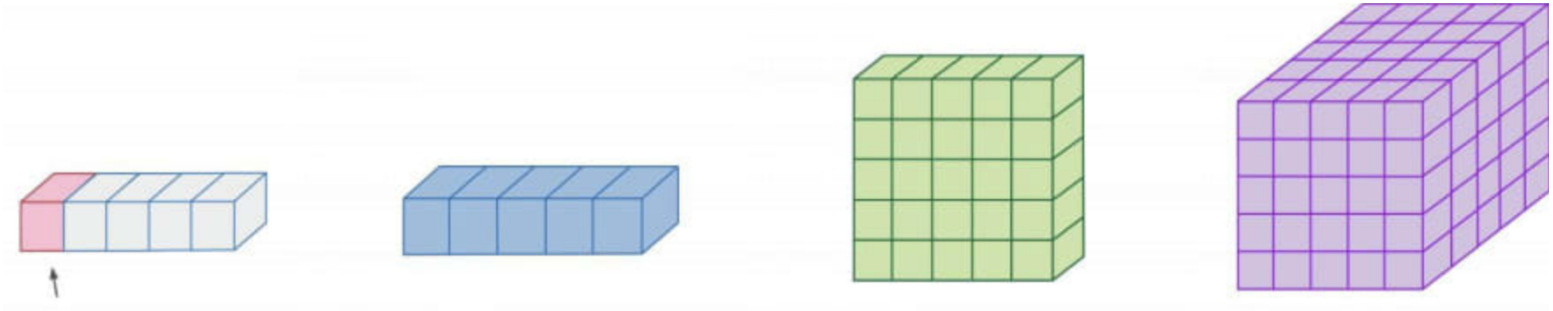
```
prime_list = [2, 3, 5, 7, 11]
prime_array = np.array(prime_list)
print(prime_array.ndim)
print(prime_array.shape)
print(prime_array.size)
print(prime_array.dtype)
```

```
1
(5,)
5
int64
```

不同維度數的 `ndarray` 有不同的暱稱

- 零維 `ndarray`：純量（Scalar）。
- 一維 `ndarray`：向量（Vector）。
- 二維 `ndarray`：矩陣（Matrix）。
- 三維或者 n 維 `ndarray`：張量（Tensor）。

純量、向量、矩陣與張量外型示意圖



來源：https://dev.to/juancarlospaco/tensors-for-busy-people-315k

In [8]:

```
scalar = np.array(5566)
print(scalar)
print(scalar.ndim)
print(scalar.shape)
```

```
5566
0
()
```

In [9]:

```
vector = np.array([5, 5, 6, 6])
print(vector)
print(vector.ndim)
print(vector.shape)
```

```
[5 5 6 6]
1
(4,)
```

In [10]:

```
matrix = np.array([[5, 5],
                   [6, 6],
                   [55, 66]])
print(matrix)
print(matrix.ndim)
print(matrix.shape) # (m, n)
```

```
[[ 5  5]
 [ 6  6]
 [55 66]]
2
(3, 2)
```

In [11]:

```
tensor = np.array([[[5, 5],
                    [6, 6],
                    [55, 66]],
                   [[5, 5],
                    [6, 6],
                    [55, 66]],
                   [[5, 5],
                    [6, 6],
                    [55, 66]],
                   [[5, 5],
                    [6, 6],
                    [55, 66]]])
print(tensor)
print(tensor.ndim)
print(tensor.shape) # (1, m, n)
```

```
[[[ 5  5]
  [ 6  6]
  [55 66]]

 [[ 5  5]
  [ 6  6]
  [55 66]]

 [[ 5  5]
  [ 6  6]
  [55 66]]

 [[ 5  5]
  [ 6  6]
  [55 66]]]
```

```
3
(4, 3, 2)
```

`ndarray` 與 `list` 的異同

`ndarray` 與 `list` 相同的地方

- indexing/slicing 的語法。
- 能夠以 indexing 更新。

indexing 的語法

`ndarray` 採用兩個方向的索引機制：

1. 由左至右：「從 0 開始」的索引機制。
2. 由右至左：「從 -1 開始」的索引機制。

In [12]:

```
primes_array = np.array([2, 3, 5, 7, 11])
print("From start to stop:")
print(primes_array[0])
print(primes_array[1])
print(primes_array[2])
print(primes_array[3])
print(primes_array[primes_array.size - 1])
print("From stop to start:")
print(primes_array[-1])
print(primes_array[-2])
print(primes_array[-3])
print(primes_array[-4])
print(primes_array[-primes_array.size])
```

```
From start to stop:
2
3
5
7
11
From stop to start:
11
7
5
3
2
```

slicing `[start:stop:step]`

除了可以取出特定位置的單個資料值，`ndarray` 也支援擷取特定片段，藉此獲得一個較短長度 `ndarray` 的語法。

- `start` 起始位置（包含）。
- `stop` 終止位置（排除）。
- `step` 間隔。

In [13]:

```
print(primes_array[::])   # default
print(primes_array[::2])  # step=2
print(primes_array[:3])   # stop=5, exclusive
print(primes_array[3:])   # start=5, inclusive
print(primes_array[::-1]) # step=-1, reverse
```

```
[ 2  3  5  7 11]
[ 2  5 11]
[2 3 5]
[ 7 11]
[11  7  5  3  2]
```

能夠以 indexing 更新

In [14]:

```
primes_array = np.array([2, 3, 5, 7, 11])
print(primes_array)   # before update
primes_array[-1] = 13 # update
print(primes_array)   # after update
```

```
[ 2  3  5  7 11]
[ 2  3  5  7 13]
```

`ndarray` 與 `list` 相異的地方

- indexing 二維以上的 `ndarray` 可以用更便捷的語法 `[i, j, k, ...]`
- 同質性資料結構類別。
- 支援元素操作（Elementwise）運算。
- 支援特殊的 indexing 語法。

indexing 二維以上的 `ndarray` 可以用 `ndarray` 便捷的語法 `[i, j, k, ...]`

In [15]:

```
matrix = np.array([[5, 5],
                   [6, 6],
                   [55, 66]])
print(matrix)
print(matrix[2, 1]) # 66 locates at [2, 1]
```

```
[[ 5  5]
 [ 6  6]
 [55 66]]
66
```

In [16]:

```
print(matrix[:, 1])
print(matrix[:, [1]]) # keep dimension
```

```
[ 5  6 66]
[[ 5]
 [ 6]
 [66]]
```

同質性資料結構類別

In [17]:

```
heterogeneous_list = [False, True, 5566, 55.66, 'Luke Skywalker']
for element in heterogeneous_list:
    print(type(element))
```

```
<class 'bool'>
<class 'bool'>
<class 'int'>
<class 'float'>
<class 'str'>
```

In [18]:

```
homogeneous_array = np.array(heterogeneous_list)
for element in homogeneous_array:
    print(type(element))
```

```
<class 'numpy.str_'>
<class 'numpy.str_'>
<class 'numpy.str_'>
<class 'numpy.str_'>
<class 'numpy.str_'>
```

支援元素操作（Elementwise）運算

In [19]:

```
# list does not support elementwise
primes_list = [2, 3, 5, 7, 11]
try:
    primes_list**2
except TypeError as error_message:
    print(error_message)
```

```
unsupported operand type(s) for ** or pow(): 'list' and
'int'
```

In [20]:

```
# ndarray supports elementwise
primes_array = np.array(primes_list)
primes_array**2
```

Out[20]:

```
array([  4,   9,  25,  49, 121])
```

支援特殊的 indexing 語法

- Fancy indexing
- Boolean indexing

什麼是 Fancy indexing

對應 `ndarray` 時中括號允許傳入 `list`，藉此可以更有彈性地取出 `ndarray` 中的元素。

In [21]:

```
primes_list = [2, 3, 5, 7, 11]
try:
    primes_list[[0, 1, 4]]
except TypeError as error_message:
    print(error_message)
```

```
list indices must be integers or slices, not list
```

In [22]:

```
primes_array = np.array([2, 3, 5, 7, 11])
print(primes_array)
print(primes_array[[0, 1, 4]])
```

```
[ 2  3  5  7 11]
[ 2  3 11]
```

什麼是 Boolean indexing

對應 `ndarray` 時中括號允許傳入由 `bool` 組成的相同長度 `list` 或 `ndarray`，藉此可以更有彈性地取出 `ndarray` 中的元素。

In [23]:

```
primes_list = [2, 3, 5, 7, 11]
try:
    primes_list[[False, True, True, True, True]]
except TypeError as error_message:
    print(error_message)
```

```
list indices must be integers or slices, not list
```

In [24]:

```
primes_array = np.array([2, 3, 5, 7, 11])
print(primes_array)
print(primes_array[[False, True, True, True, True]])
print(primes_array % 2 == 1)
print(primes_array[primes_array % 2 == 1])
```

```
[ 2  3  5  7 11]
[ 3  5  7 11]
[False  True  True  True  True]
[ 3  5  7 11]
```

如何操作 `ndarray`

常用的 `ndarray` 操作

- 調整外型。
- 複製。
- 合併。
- 分割。

調整外型

- `ndarray.reshape()`：調整為指定外型 `(..., m, n)`
- `ndarray.ravel()`：調整為一維 `(m,)`

In [25]:

```
array_range = np.arange(1, 13)
print(array_range)
print(array_range.shape)
print(array_range.reshape(3, 4))
print(array_range.reshape(3, 4).shape)
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12]
(12,)
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
(3, 4)
```

在其他維度已經決定時可以方便地指定 `-1` 給最後一個維度

In [26]:

```
print(array_range.reshape(3, -1))
print(array_range.reshape(-1, 3))
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

使用 `ndarray.ravel()` 調整成一維

In [27]:

```
array_range = np.arange(1, 13).reshape(3, -1)
print(array_range.shape)
print(array_range.ndim)
print(array_range.ravel().shape)
print(array_range.ravel().ndim)
```

```
(3, 4)
2
(12,)
1
```

複製

- 透過物件命名參照並不會真的複製，會讓兩個物件名稱共享一個 `ndarray` 的資料值，但是卻能有不同的外型。
- 使用 `ndarray.copy()` 明確地複製。

In [28]:

```
vector = np.arange(1, 10)
matrix = vector.reshape(3, 3)
matrix[1, 1] = 5566
print(vector)
print(matrix)
```

```
[   1    2    3    4 5566    6    7    8    9]
[[   1    2    3]
 [   4 5566    6]
 [   7    8    9]]
```

使用 `ndarray.copy()` 明確地複製

In [29]:

```
vector = np.arange(1, 10)
matrix = vector.copy().reshape(3, 3)
matrix[1, 1] = 5566
print(vector)
print(matrix)
```

```
[1 2 3 4 5 6 7 8 9]
[[   1    2    3]
 [   4 5566    6]
 [   7    8    9]]
```

合併

使用 `np.concatenate()` 函數合併。

- 指定參數 `axis=0` 垂直合併（預設值）。
- 指定參數 `axis=1` 水平合併。

In [30]:

```
array_a = np.arange(1, 5).reshape(2, 2)
array_b = np.arange(5, 9).reshape(2, 2)
print(np.concatenate((array_a, array_b)))         # default, axis=0
print(np.concatenate((array_a, array_b), axis=1)) # axis=1
```

```
[[1 2]
 [3 4]
 [5 6]
 [7 8]]
[[1 2 5 6]
 [3 4 7 8]]
```

分割

使用 `np.split()` 函數分割。

- 指定參數 `axis=0` 垂直分割（預設值）。
- 指定參數 `axis=1` 水平分割。

In [31]:

```
array_range = np.arange(20).reshape(-1, 2)
upper_array, lower_array = np.split(array_range, 2) # split to 2 ndarrays
print(upper_array)
print(lower_array)
```

```
[[0 1]
 [2 3]
 [4 5]
 [6 7]
 [8 9]]
[[10 11]
 [12 13]
 [14 15]
 [16 17]
 [18 19]]
```

如果以 `list` 傳入參數則表示分割的索引值

In [7]:

```
array_range = np.arange(20).reshape(-1, 2)
upper_array, lower_array = np.split(array_range, [2]) # split on index 2
print(upper_array)
print(lower_array)
```

```
[[0  1]
 [2  3]]
[[ 4  5]
 [ 6  7]
 [ 8  9]
 [10 11]
 [12 13]
 [14 15]
 [16 17]
 [18 19]]
```

In [32]:

```
array_range = np.arange(20).reshape(-1, 2)
left_array, right_array = np.split(array_range, 2, axis=1)
print(left_array)
print(right_array)
```

```
[[ 0]
 [ 2]
 [ 4]
 [ 6]
 [ 8]
 [10]
 [12]
 [14]
 [16]
 [18]]
[[ 1]
 [ 3]
 [ 5]
 [ 7]
 [ 9]
 [11]
 [13]
 [15]
 [17]
 [19]]
```

NumPy 函數

NumPy 提供非常豐富的數值運算函數

- 通用函數（Universal functions）。
- 聚合函數（Aggregate functions）。

什麼是通用函數

通用函數是具備向量化（Vectorized）特性的函數，接受固定數量、外型的輸入並對應相同數量、外型的輸出。

In [33]:

```
array_range = np.arange(10)
print(array_range)
print(np.power(array_range, 2)) # np.power() is a universal function
print(np.exp(array_range))      # np.exp() is a universal function
```

```
[0 1 2 3 4 5 6 7 8 9]
[ 0  1  4  9 16 25 36 49 64 81]
[1.00000000e+00 2.71828183e+00 7.38905610e+00 2.0085536
9e+01
 5.45981500e+01 1.48413159e+02 4.03428793e+02 1.0966331
6e+03
 2.98095799e+03 8.10308393e+03]
```

轉換純量函數為通用函數

使用 `np.vectorize()` 函數將只能作用在單一資料值上的函數轉換為通用函數。

In [34]:

```
def is_prime(x):
    number_of_divisors = 0
    for integer in range(1, x + 1):
        if x % integer == 0:
            number_of_divisors += 1
    return number_of_divisors == 2

try:
    is_prime(np.arange(10))
except TypeError as error_message:
    print(error_message)
```

```
only integer scalar arrays can be converted to a scalar
index
```

In [35]:

```
vectorized_is_prime = np.vectorize(is_prime)
vectorized_is_prime(np.arange(10))
```

Out[35]:

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

什麼是聚合函數

聚合函數是能夠將多個資料值輸入摘要為單一值輸出的函數。

In [36]:

```
array_range = np.arange(1, 16).reshape(3, 5)
np.sum(array_range)
```

Out[36]:

```
120
```

NumPy 的聚合函數有兩個值得注意的特性

1. 可以沿指定的軸（axis）進行聚合。
2. 針對含有 `np.nan`（Not a Number）的 `ndarray` 有相對應名稱的聚合函數可以運算。

In [37]:

```
# Aggregate along specific axis
array_range = np.arange(1, 16, dtype=float).reshape(3, 5)
print(np.sum(array_range))
print(np.sum(array_range, axis=0))
print(np.sum(array_range, axis=1))
```

```
120.0
[18. 21. 24. 27. 30.]
[15. 40. 65.]
```

NumPy 除了定義類別、函數，亦有定義常數

Not a Number 可以表示「未定義」或「遺漏」的浮點數。

來源：https://numpy.org/doc/stable/reference/constants.html

In [38]:

```
print(np.nan)
print(type(np.nan))
```

```
nan
<class 'float'>
```

針對含有 `np.nan` 的 `ndarray` 有相對應名稱的聚合函數可以運算

- `np.sum()` vs. `np.nansum()`
- `np.mean()` vs. `np.nanmean()`
- ...等。

In [39]:

```
# Similar function names for array with np.nan
array_range = np.arange(1, 16, dtype=float).reshape(3, 5)
array_range[2, 4] = np.nan
print(array_range)
print(np.sum(array_range))
print(np.nansum(array_range))
```

```
[[ 1.  2.  3.  4.  5.]
 [ 6.  7.  8.  9. 10.]
 [11. 12. 13. 14. nan]]
nan
105.0
```

重點統整

- NumPy 是 Numeric Python 的簡稱，是 Python 最重要的資料科學模組之一。
- NumPy 的核心功能
    - 使用 `ndarray` 來進行數值操作。
    - 使用模組定義的函數對 `ndarray` 進行數值運算。
    - `ndarray` 是其他資料科學模組 Pandas、Matplotlib 與 Scikit-Learn 的基石。

重點統整（續）

- `ndarray` 與 `list` 相異的地方：
    - indexing 二維以上的 `ndarray` 可以用更便捷的語法 `[i, j, k, ...]`
    - 同質性資料結構類別。
    - 支援元素操作（Elementwise）運算。
    - 支援特殊的 indexing 語法：Fancy indexing/Boolean indexing
- 常用的 ndarray 操作
    - 調整外型。
    - 複製。
    - 合併。
    - 分割。