



Python for Data Science

Par. Dr CHAIBI Hasna

A.U 2024-2025

Introduction

- Nous vivons dans un monde noyé par les données. Les sites web suivent à la trace chaque clic de chaque utilisateur. Tous les jours, votre smartphone enregistre votre localisation et votre vitesse à la seconde près.
- Les voitures intelligentes collectent les habitudes de conduite, les maisons intelligentes collectent les habitudes de vie et les marqueteurs intelligents collectent les habitudes d'achat.
- ***La Data Science est la science des données. C'est la discipline qui permet d'explorer et d'analyser les données brutes pour les transformer en informations précieuses permettant de résoudre des problèmes et de prendre des décisions.***

Introduction

- La Data Science, ou science des données, désigne un mélange disciplinaire d'inférence de données, de développement d'algorithmes et de technologie. Ces disciplines ont pour but de **résoudre des problèmes analytiques complexes**.
- **La Data Science permet de découvrir des connaissances au sein des données.** En plongeant dans ces informations, l'utilisateur peut découvrir et comprendre des tendances et des comportements complexes. Il s'agit de faire remonter à la surface des informations pouvant aider les entreprises à prendre des décisions plus intelligentes.

Introduction

- Python possède plusieurs fonctionnalités qui le rendent particulièrement bien adapté pour apprendre (et pratiquer) la data science :
- il est gratuit ;
- il est assez facile à coder (et aussi à comprendre) ;
- il dispose de nombreuses bibliothèques en lien avec la data science.

Introduction

- Il existe plusieurs bibliothèques, frameworks, modules et boîtes à outils dédiés aux **data sciences** pour mettre en place les algorithmes et les techniques de data science les plus courants (et aussi les moins courants d'ailleurs).
- Si vous devenez data scientist, vous pénétrerez dans l'intimité de **NumPy**, **scikitlearn**, **pandas**, **seaborn**, **Matplotlib** et d'autres bibliothèques.

Prise en main de l'environnement de développement.

Anaconda, qu'est-ce que c'est ?



- **Anaconda** est une distribution libre et open source des langages de programmation Python et R appliquée au développement d'applications dédiées à la science des données et à l'apprentissage automatique (traitement de données à grande échelle, analyse prédictive, calcul scientifique), qui vise à simplifier la gestion des paquets et de déploiement.
- En installant Anaconda, vous installerez **Python**, **Jupyter Notebook** (que nous présenterons plus en détail aux prochains slides) et des dizaines de packages scientifiques, dont certains indispensables à la science de données

Prise en main de l'environnement de développement.

Anaconda, qu'est-ce que c'est ?

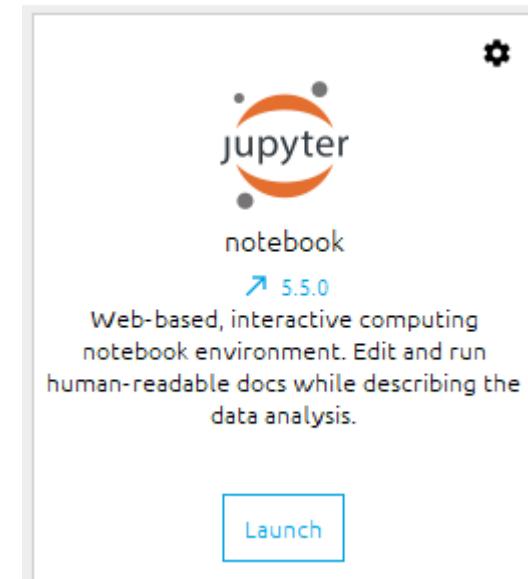
- Anaconda c'est :
 - Une solution open source tout-en-un
 - Des packages et des outils inclus
 - Un gestionnaire d'environnements open source
- Anaconda met à votre disposition une solution vous permettant d'installer tous l'écosystème data avec notamment :
 - une installation de python
 - les packages de data science tels que **numpy**, **pandas**, **scikit-learn**...
 - Des IDE de dernière génération tels que **Jupyter** ([JupyterLab](#) et Jupyter Notebooks) ou Spyder
 - l'outil **conda** pour gérer les environnements et les répertoires de packages



Prise en main de l'environnement de développement.

Anaconda, qu'est-ce que c'est ?

- **Jupyter Notebook** est une application web qui vous permet de stocker des lignes de code Python, les résultats de l'exécution de ces dernières (graphiques, tableaux, etc.) et du texte formaté.
- **Jupyter Notebook** est un outil puissant qui permet aux utilisateurs du langage Python de créer et de partager des documents interactifs contenant du code dynamique et exécutable, des visualisations de contenus, des textes de documentation et des équations.



Introduction

Définition :

- Un module est un fichier contenant des sous-programmes regroupés de façon cohérente
- Le nom du fichier aura comme suffixe .py

Structure d'un projet:

Plusieurs modules de fonctions et un module particulier, dit “**main**”, où l'on définit les variables du projet et où l'on utilise les fonctions des différents modules.

Trois formats pour importer

- **import NomDuModule**
- **from NomDuModule import ***
- **from NomDuModule import uneFonction**

Les modules

import ...

import nomModule

- Importe tout le contenu du fichier nomModule.py
- Pour faire référence à quelque chose dans le fichier il faut ajouter le nom du module « Exemple » avant le nom de l'entité utilisée.

Exemple :

```
import Exemple  
Exemple.doubler(n)  
print(Exemple.tripler(n))
```

Attention tout le contenu du module sera chargé en mémoire.

Les modules

from ... import ...

from nomModule import *

- Tout le contenu du fichier nomModule.py est importé et chargé en mémoire
- Tout le module est dans le namespace courant, il est donc inutile de faire préfixé toutes les entités présentes dans le fichier par le nom du module

Exemple:

```
from Exemple import *
tripler(n)
doubler(n)
```

from ... import ...

from nomModule import item

- Seul l'élément item est importé du fichier nomModule.py.
- Comme dans le cas précédent il n'est pas nécessaire de faire préfixer le nom de cet item par le nom du module
- Attention à l'écrasement d'une entité de même nom déjà présente dans le namespace courant.

Exemple :

```
from Exemple import tripler  
tripler(n)
```

Les modules

from ... import ...

- Exemple

```
# le module Exemple
def doubler(x):
    return 2*x
def tripler(x):
    return 3*x
```

```
import import_ipynb
```

```
from Exemple import doubler
print(doubler(12))
```

24

```
from exemple import *
print(tripler(12))
```

36

```
import import_ipynb
```

```
import Exemple
print(tripler(15))
```

45

Les modules de base pour Data science

Les modules de base pour Data science

Les modules de base pour Data science:

- import math
- import random
- import statistics

Les modules de base pour Data science

Le module math et statistics

Module math

```
import math
math.
    acos
    acosh
    asin
    asinh
    atan
    atan2
    atanh
    ceil
    copysign
    cos
```

```
import math
pi=math.pi
print(math.sin(pi/2))
```

1.0

#Module statistics

```
statistics.
    bisect_left
    bisect_right
    collections
    Decimal
    Fraction
    groupby
    harmonic_mean
    math
    mean
    median
```

```
import statistics
liste=[12, 14, 13, 17, 7, 8]
statistics.mean(liste)
```

11.83333333333334

```
statistics.variance(liste)
```

14.166666666666666

Les modules de base pour Data science

Le module random

- Il existe de nombreux phénomènes dus au hasard dans la réalité, et il peut être utile, parfois numériquement, de pouvoir modéliser ces phénomènes pour balayer les différents cas possibles. En Python, le module **random** contient plusieurs fonctions pour pouvoir générer des nombres ou des suites de nombres aléatoires.
- La fonction de base de génération de nombre aléatoire s'appelle **random()**. Elle va générer un **float aléatoire compris entre 0 et 1 non inclus**.

Les modules de base pour Data science

Le module random

Module random

```
#générer aléatoirement un nombre réel  
#compris entre 0 et 1  
print(random.random())
```

0.7579544029403025

```
#générer aléatoirement un nombre entier  
# compris entre 3 et 6  
print(random.randint(3,6))
```

5

```
# générer aléatoirement un float compris entre 3 et 6  
import random  
print(random.uniform(3,6))
```

5.357221706839371

```
#générer aléatoirement un entier  
#entre 0 et 20  
print(random.randrange(20))
```

16

Les modules de base pour Data science

Le module random

Module random

Choisissez aléatoirement dans une liste : sous-échantillonnage

Le module **random** propose une fonction permettant de faire la sélection directement sur la liste : la fonction **choice**

L'évolution de celle-ci est la fonction **choices**, permettant cette fois-ci de sélectionner un échantillon de la liste initiale,

```
import random
random.seed(0) # permet d'avoir toujours même aléatoire
liste=[12, 14, 13, 17, 7, 8]
print(random.choice(liste))
```

17

```
noms=["hasna","laila","ahmad","safae","karim"]
print(random.choices(noms,k=2))
print(random.choices(noms,k=3))
```

```
['karim', 'ahmad']
['laila', 'ahmad', 'laila']
```

générer éléatoirement k élémunent dans une liste

```
noms=["hasna","laila","ahmad","safae","karim"]
print(random.sample(noms,2))
print(random.sample(noms,3))
```

```
['ahmad', 'safae']
['laila', 'ahmad', 'karim']
```

Les modules de base pour Data science

Le module random

Module random

```
#générer une liste avec des nombres entre 0 et 50  
# et le nombre d'élément de cette liste est aussi  
#aléatoire (entre 0 et 10 éléments)
```

```
print(random.sample(range(50),random.randrange(10)))
```

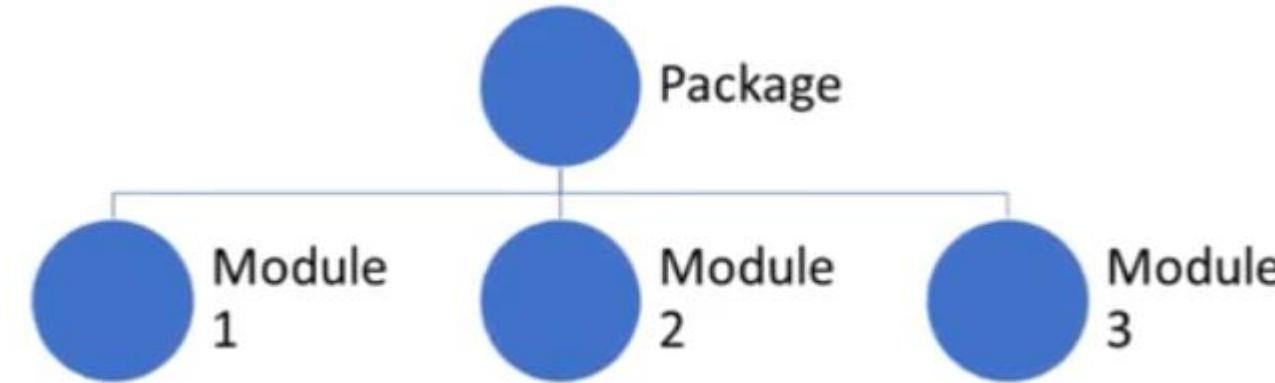
```
[9, 34, 28]
```

```
# Mélanger aléatoirement une Liste  
liste=[1, 3, 5, 6, 8, 9, 10, 12, 14,18]  
random.shuffle(liste)  
print(liste)
```

```
[5, 3, 9, 6, 10, 8, 18, 12, 14, 1]
```

Le package

- Un **package** est un regroupement de différents modules.



Les fonctions de base de python

Built-in Functions				
<code>abs()</code>	<code>delattr()</code>	<code>hash()</code>	<code>memoryview()</code>	<code>set()</code>
<code>all()</code>	<code>dict()</code>	<code>help()</code>	<code>min()</code>	<code>setattr()</code>
<code>any()</code>	<code>dir()</code>	<code>hex()</code>	<code>next()</code>	<code>slice()</code>
<code>ascii()</code>	<code>divmod()</code>	<code>id()</code>	<code>object()</code>	<code>sorted()</code>
<code>bin()</code>	<code>enumerate()</code>	<code>input()</code>	<code>oct()</code>	<code>staticmethod()</code>
<code>bool()</code>	<code>eval()</code>	<code>int()</code>	<code>open()</code>	<code>str()</code>
<code>breakpoint()</code>	<code>exec()</code>	<code>isinstance()</code>	<code>ord()</code>	<code>sum()</code>
<code>bytearray()</code>	<code>filter()</code>	<code>issubclass()</code>	<code>pow()</code>	<code>super()</code>
<code>bytes()</code>	<code>float()</code>	<code>iter()</code>	<code>print()</code>	<code>tuple()</code>
<code>callable()</code>	<code>format()</code>	<code>len()</code>	<code>property()</code>	<code>type()</code>
<code>chr()</code>	<code>frozenset()</code>	<code>list()</code>	<code>range()</code>	<code>vars()</code>
<code>classmethod()</code>	<code>getattr()</code>	<code>locals()</code>	<code>repr()</code>	<code>zip()</code>
<code>compile()</code>	<code>globals()</code>	<code>map()</code>	<code>reversed()</code>	<code>__import__()</code>
<code>complex()</code>	<code>hasattr()</code>	<code>max()</code>	<code>round()</code>	

Les fonctions de base pour Data Science

- Les fonctions les plus importantes pour Data Science

```
a=-4.67
```

```
abs(a) # La valeur absolue de a
```

```
4.67
```

```
round(a) # Renvoie le nombre arrondi à la précision
```

```
-5
```

```
Liste=[1,0,20,22]
```

```
max(Liste) # renvoie le max de la liste
```

```
22
```

```
min(Liste) # renvoie le min de la liste
```

```
0
```

```
sum(Liste) # renvoie la somme des éléments de la liste
```

```
43
```

Les fonctions de base pour Data Science

- Les fonctions les plus importantes pour Data Science

```
len(Liste) # renvoie la taille de la liste
```

```
4
```

```
Liste1=[True, False, True]
```

```
all(Liste1) #retourne True si tous les éléments de la liste sont égaux à True
```

```
False
```

```
any(Liste) #retourne True quand au moins un des éléments égale à True
```

```
True
```

```
Liste=[1,0,20,22]
```

```
any(Liste) # il y a au moins un nombre qui n'est pas égal à 0
```

```
True
```

Les fonctions de base pour Data Science

- Les fonctions de conversion:

1) Des variables:

str(), int(), float()

```
y=14  
type(y) # renvoie le type de la variable y
```

int

```
str(y) # convertit un nombre entier en chaîne de caractère
```

'14'

```
x='30'  
x=int(x) # convertit un string à un nombre entier  
type(x)
```

int

```
b=40  
b=float(b) # convertit un entier en float  
print(b)  
type(b)
```

40.0

Les fonctions de base pour Data Science

- Les fonctions de conversion:

2) Des structures: list(), tuple(), dict()

```
Liste_2=[30,0,12,39]
Tuple_2=tuple(Liste_2)
print(Tuple_2) #convertit une liste au tuple
(30, 0, 12, 39)

list(Tuple_2) # convertit un tuple à une liste
[30, 0, 12, 39]

Dic={"Ahmad":20, 'Rayane':10, 'Talal': 45}
list(Dic.keys()) #convertit les clés de dictionnaire à une liste
['Ahmad', 'Rayane', 'Talal']

Nom=['Ahmad', 'Zahira', 'Yan']
Age=[12, 13, 14]
dict(zip(Nom, Age))# convertit des listes au dictionnaire
{'Ahmad': 12, 'Zahira': 13, 'Yan': 14}
```

Le module NUMPY

Le module NUMPY

NDArray

- **NumPy** est la librairie Python dédiée au calcul scientifique fournissant des fonctions très performantes de calcul, mais aussi des **structures de données**, tout aussi performantes.
- En Data Science, il est essentiel d'avoir des structures adaptées pour stocker et manipuler de grandes quantités de données. C'est là qu'intervient **NumPy**, qui intègre une nouvelle structure de données en Python, les **ndarrays** (tableaux à N dimensions, en français, N représentant un chiffre), qui sont des tableaux multidimensionnels ou matrices. Il est important de noter que cette structure de données permet de stocker des données uniquement de même type

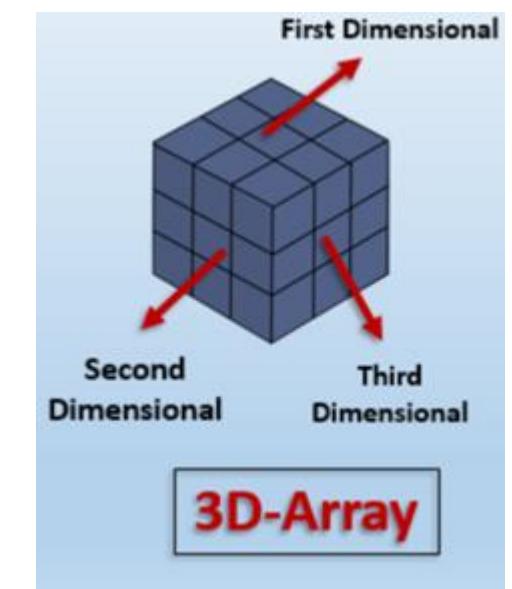
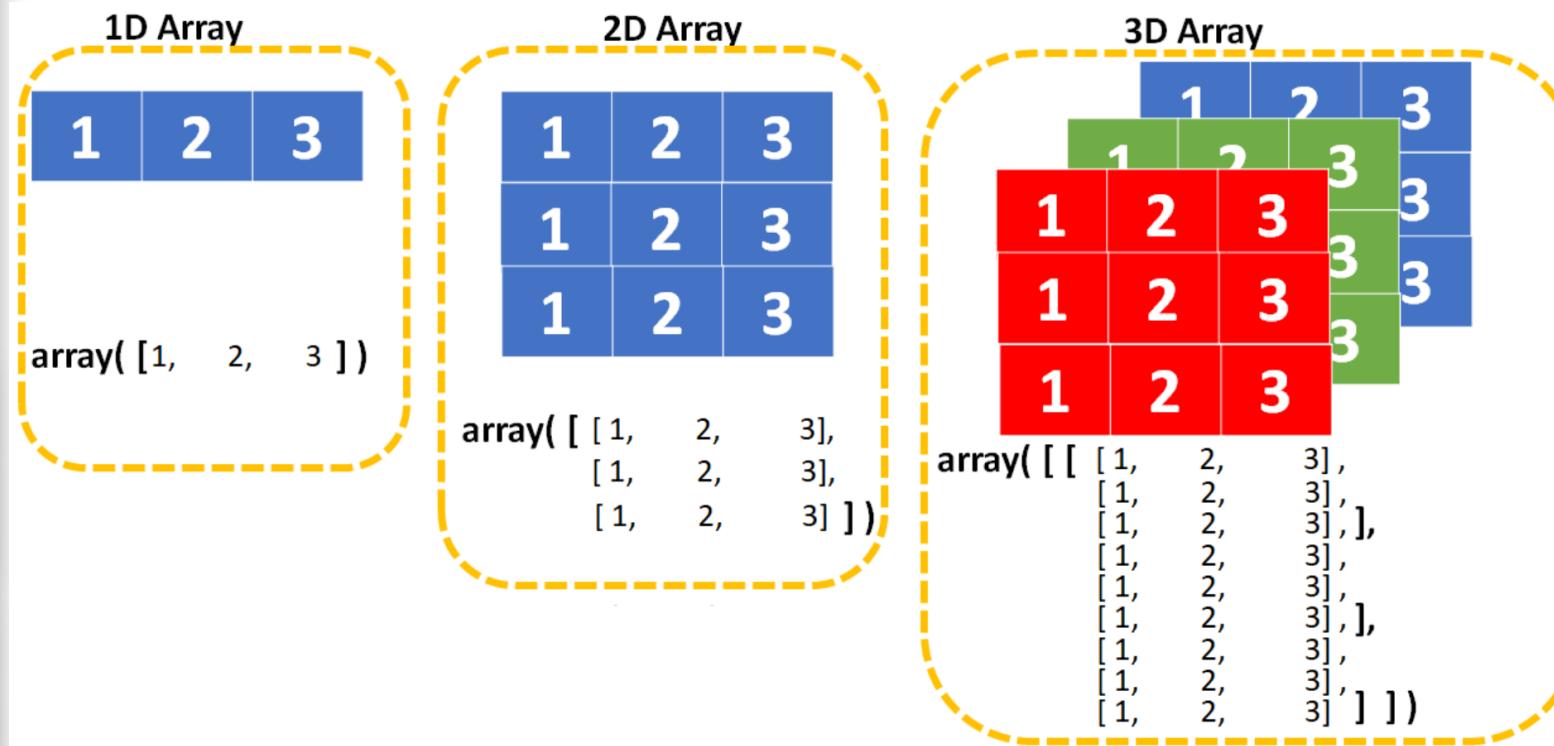
NDArray

- NumPy peut gérer de très gros tableaux et est très performante en temps de calcul sur ces tableaux : les ndarrays prennent en effet moins de place mémoire que d'autres objets Python, comme par exemple les listes.
- C'est pour cela que cette librairie a été développée et qu'elle est si utilisée : elle est très performante. C'est également pour cette raison que de nombreuses librairies ont été développées au-dessus de celle-ci, telle que Pandas, que nous verrons plus tard dans ce cours

Le module NUMPY

NDArray

Les **ndarrays** de NumPy peuvent être unidimensionnels (aussi appelés **1D array**, ce qu'on peut voir comme une liste), bidimensionnels (**2D array**) donc un tableau avec des lignes et des colonnes, ou encore des tableaux à plus de deux dimensions (**3D array**, **4D array**, **5D array**...),.

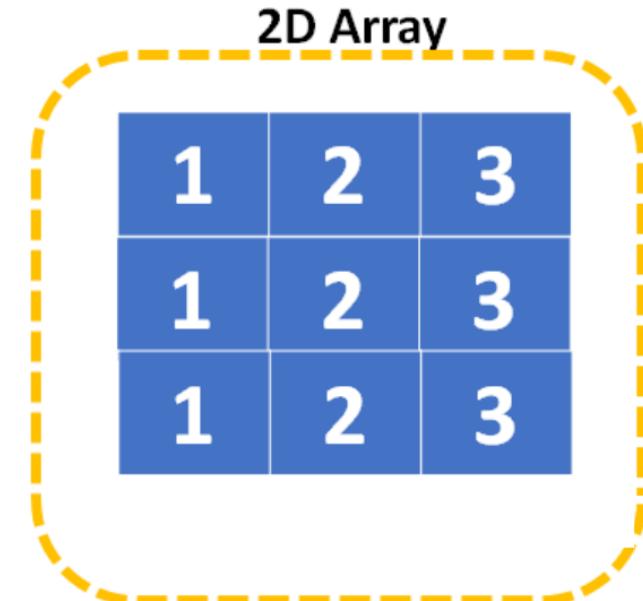


Le module NUMPY

NDArray

Lorsque vous souhaitez effectuer des opérations mathématiques ou logiques rapides sur un grand jeu de données, **NumPy** est votre allié. Pour pouvoir utiliser **NumPy** sous **Python**, il suffit de charger la librairie sous **Jupyter**, celle-ci étant déjà installée dans la distribution Anaconda

	A	B	C
1	age	salaire	credit
2	25	5000	1
3	35	7000	1
4	20	4000	1
5	50	3000	0
6	60	3500	0
7	62	4000	0
8	40	9000	1
9	42	2000	0
10	46	10000	1
11	37	8000	1
12	22	9000	1
13	30	2000	0
14	50	3000	0
15	56	12000	1
16	38	8000	1
17	33	2000	0
18	26	2200	0
19	23	2300	0



IMPORTANT

NDArray est un
objet qui comporte
une série d'attributs
et des méthodes

Le module NUMPY

NDArray – les attributs

Attributs	Description
T	Transposez la matrice. Lorsque le tableau est 1 D, le tableau original est retourné.
data	Un objet tampon Python qui pointe vers la position de départ des données dans le tableau.
dtype	Le type de données de l'élément contenu dans le ndarray.
flags	Informations sur la façon de stocker les données de ndarray en mémoire (disposition de la mémoire).
flat	Un itérateur qui convertit ndarray en un tableau unidimensionnel.
imag	La partie imaginaire des données du ndarray

Le module NUMPY

NDArray – les attributs

<code>real</code>	Partie réelle des données ndarray
<code>size</code>	Le nombre d'éléments contenus dans le ndarray.
<code>itemsize</code>	La taille de chaque élément en octets.
<code>nbytes</code>	La mémoire totale (en octets) occupée par le ndarray.
<code>ndim</code>	Le nombre de dimensions contenues dans le ndarray.
<code>shape</code>	La forme du ndarray (les résultats sont des tuples).
<code>strides</code>	Le nombre d'octets requis pour passer à l'élément adjacent suivant dans chaque direction de dimension est représenté par un tuple.
<code>ctypes</code>	Un itérateur qui est traité dans le module ctypes.
<code>base</code>	L'objet sur lequel ndarray est basé (quelle mémoire est référencée).

NDArray – les attributs

❑ L'attribut shape

- Shape trouve la forme d'un tableau. Par forme, nous voulons dire qu'il aide à trouver les dimensions d'un tableau. Il est un tuple car nous ne pouvons pas modifier un tuple tout comme nous ne pouvons pas modifier les dimensions d'un tableau.

NDArray – les attributs

□ L'attribut shape

```
import numpy as np  
A = np.array([])  
print(A.shape)
```

(0,)

```
A = np.array([89, 34, 56, 87, 90, 23, 11, 2, 65, 78, 82, 28, 78])
```

```
print(A.shape)
```

(13,)

NDArray – les attributs

□ L'attribut `shape`

```
A = np.array([[11, 12, 5], [15, 6, 10], [10, 8, 12], [12,15,8], [34, 78, 90]])  
print(A)  
print(A.shape)
```

```
[[11 12 5]  
 [15 6 10]  
 [10 8 12]  
 [12 15 8]  
 [34 78 90]]  
(5, 3)
```

NDArray – les attributs

□ L'attribut size

- La taille d'un tableau est le nombre total d'éléments dans le tableau. **L'attribut size** donne la taille d'un tableau.

```
A = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
print(A.size)
```

8

NDArray – les attributs

□ L'attribut T

- La transposé d'un tableau à deux est obtenue par **L'attribut T**

```
A = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
print(A.T)
```

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

NdArray – les attributs

□ L'attribut **ndim**

- Ndim donne le nombre de dimension contenues dans NdArray.

```
A = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
print(A.ndim)
```

2

NDArray – les constructeurs

- `np.zeros(shape)`
- `np.linspace(début, fin, quantité)`
- `np.empty(shape)`
- `np.arange(début, fin ,pas)`
- `np.ones(shape)`
- `np.eye()`
- `np.full(shape,value)`
- `np.random.randn(shape)`

NDArray – les constructeurs

□ zeros()

- np.**zeros()** est utilisé pour créer un tableau dont tous les éléments sont 0

```
A=np.zeros((4,6))
```

```
A
```

```
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.]])
```

NDArray – les constructeurs

empty()

- Dans certains cas, vous voulez seulement initialiser un tableau avec la forme spécifiée et vous ne vous souciez pas des données d'initialisation à l'intérieur. Vous pouvez utiliser **np.empty** pour obtenir une initialisation plus rapide, mais gardez à l'esprit que cela ne garantit pas que la valeur du tableau créé soit 0.

```
A=np.empty((4,6))
```

```
A
```

```
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.]])
```

NDArray – les constructeurs

❑ zeros_like()

- Supposons que nous ayons déjà un tableau et que nous voulions créer un tableau de zéros ayant la même forme. Nous pourrions utiliser la méthode traditionnelle pour créer ce nouveau tableau.

```
A=np.zeros((4,6))
```

```
A
```

```
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.]])
```

```
B=np.zeros_like(A)
```

```
B
```

```
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.]])
```

NDArray – les constructeurs

□ **ones()**

- Comme pour les **zeros**, nous pourrions aussi créer un tableau rempli de uns. La syntaxe et les paramètres de **np.ones()** sont identiques à ceux de **np.zeros()**.

```
A=np.ones((3,5))  
A  
  
array([[1., 1., 1., 1., 1.],  
       [1., 1., 1., 1., 1.],  
       [1., 1., 1., 1., 1.]])
```

NDArray – les constructeurs

□eye()

- Le tableau diagonal est le tableau 2-D avec des uns sur la diagonale et des zéros ailleurs. Il peut être créé avec la méthode np.eye()

```
A=np.eye(5)
A
array([[1.,  0.,  0.,  0.,  0.],
       [0.,  1.,  0.,  0.,  0.],
       [0.,  0.,  1.,  0.,  0.],
       [0.,  0.,  0.,  1.,  0.],
       [0.,  0.,  0.,  0.,  1.]])
```

NDArray – les constructeurs

□ **full()**

- Permet de créer un tableau rempli de par un nombre .

```
A=np.full((5,3),8)
```

```
A
```

```
array([[8, 8, 8],  
       [8, 8, 8],  
       [8, 8, 8],  
       [8, 8, 8],  
       [8, 8, 8]])
```

NDArray – les constructeurs

random.randn()

- `numpy.random.randn(10)` : array 1d de 10 nombres d'une distribution gaussienne standard (moyenne 0, écart-type 1).
- `numpy.random.randn(10, 10)` : array 2d de 10×10 nombres d'une distribution gaussienne standard.
- `numpy.random.randint(1, 5, 10)` : une array 1d de 10 nombres entiers entre 1 et 5, 5 exclus.
- `numpy.random.random_integers(1, 5, 10)` : une array 1d de 10 nombres entiers entre 1 et 5, 5 inclus.
- `numpy.random.random_sample(7)` : renvoie 7 valeurs aléatoires dans l'intervalle $[0,1[$.

NDArray – les constructeurs

random.randn()

```
A=np.random.randn(10)  
print(A)
```

```
[ 0.22137809 -0.25632089 -2.23820248  0.85703759 -0.67205079  0.92401736  
-0.91934082  0.0254295   -0.61792591 -0.28967177]
```

```
A=np.random.randn(3,4)  
print(A)
```

```
[[ -0.12426782 -1.25685292  0.39992658  0.6506183 ]  
[ -0.91312844 -0.03795624 -0.5671448   -0.37218876]  
[ -0.40302534  0.79840801 -1.00779    0.35101293]]
```

NDArray – les constructeurs

□ **linspace(début, fin ,q)**

- Renvoie des nombres uniformément espacés sur un intervalle [début et fin].

```
np.linspace(0,20,5)
```

```
array([ 0.,  5., 10., 15., 20.])
```

```
np.linspace(0,20,8)
```

```
array([ 0.          ,  2.85714286,  5.71428571,  8.57142857, 11.42857143,
       14.28571429, 17.14285714, 20.        ])
```

NDArray – les constructeurs

❑ arange(début, fin ,pas)

- Renvoie des nombres uniformément espacés sur un intervalle [début et fin].

```
np.arange(0,10)
```

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

```
np.arange(0,10,2)
```

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

```
np.arange(0,10,3)
```

```
array([0, 3, 6, 9])
```

NDArray – les constructeurs

□ Tableau de trois dimension

```
#Tableau de trois dimension  
T= np.zeros((3, 4, 2))
```

```
print(T)
```

```
[[[0. 0.]  
 [0. 0.]  
 [0. 0.]  
 [0. 0.]]]
```

```
[[[0. 0.]  
 [0. 0.]  
 [0. 0.]  
 [0. 0.]]]
```

```
[[[0. 0.]  
 [0. 0.]  
 [0. 0.]  
 [0. 0.]]]]
```

NDArray – type de données

□ **dtype**

- Le type de données - **dtype** dans NumPy est différent des types de données primitives dans Python, par exemple, **dtype** a le type avec une résolution plus élevée qui est utile dans le calcul des données.

Type de données	Description
<code>bool</code>	Booléen
<code>int8</code>	Entier signé 8 bits
<code>int16</code>	Entier signé 16 bits
<code>int32</code>	Entier signé 32 bits
<code>int64</code>	Entier signé 64 bits
<code>uint8</code>	Entier non signé 8 bits
<code>uint16</code>	Entier non signé 16 bits
<code>uint32</code>	Entier non signé 32 bits
<code>uint64</code>	Entier non signé 64 bits

NDArray – type de données

□ **dtype**

- Le type de données - **dtype** dans NumPy est différent des types de données primitives dans Python, par exemple, **dtype** a le type avec une résolution plus élevée qui est utile dans le calcul des données.

np.float16

$\pi = 3.14$

- Moins de précis
- Plus rapide

np.float64

$\pi = 3.141592653$
 589793

- Plus précis
- Moins rapide

Type de données	Description
float16	Nombre à virgule flottante 16 bits
float32	Nombre à virgule flottante 32 bits
float64	Nombre à virgule flottante 64 bits
complex64	Nombre complexe 64 bits
complex128	Nombre complexe de 128 bits

NDArray – type de données

□ **dtype**

- Lors de la création d'une nouvelle donnée **ndarray**, vous pouvez définir le type de données de l'élément par des constantes de type chaîne ou de données dans la bibliothèque NumPy.

```
# by string
test = np.array([4, 5, 6], dtype='int64')
print(test)

# by data type constant in numpy
test = np.array([7, 8, 8], dtype=np.int64)
print(test)
```

NDArray – append

Append

Numpy a aussi la fonction **append** pour ajouter des données à un tableau, tout comme l'opération `append` à `list` en Python. Mais dans certains cas, `append` dans NumPy est aussi un peu similaire à la méthode **extend** dans `list` en Python.

```
numpy.append(arr, values, axis = None)
```

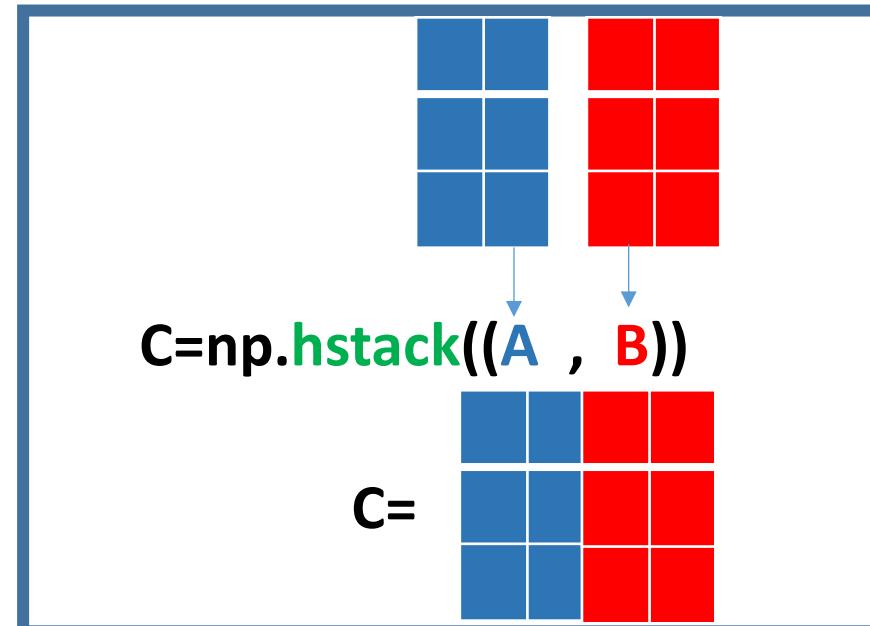
Paramètres d'entrée

nom du paramètre	type de données	Description
<code>arr</code>	comme un tableau	Un tableau pour ajouter un élément
<code>values</code>	comme un tableau	Tableau ajouté
<code>axis</code>	INT	L'axe le long duquel les <code>valeurs</code> sont ajoutées.

NDArray – hstack

□ hstack

- Empilez les tableaux en séquence horizontalement (par colonne).
- Cela équivaut à la concaténation le long du deuxième axe, sauf pour les tableaux 1D où il concatène le long du premier axe.



```
A=np.full((3,4),8)
B=np.zeros((3,2),dtype='int16')
print(A)
print(B)
```

```
[[8 8 8 8]
 [8 8 8 8]
 [8 8 8 8]]
 [[0 0]
 [0 0]
 [0 0]]
```

```
C=np.hstack((A,B))
print(C)
```

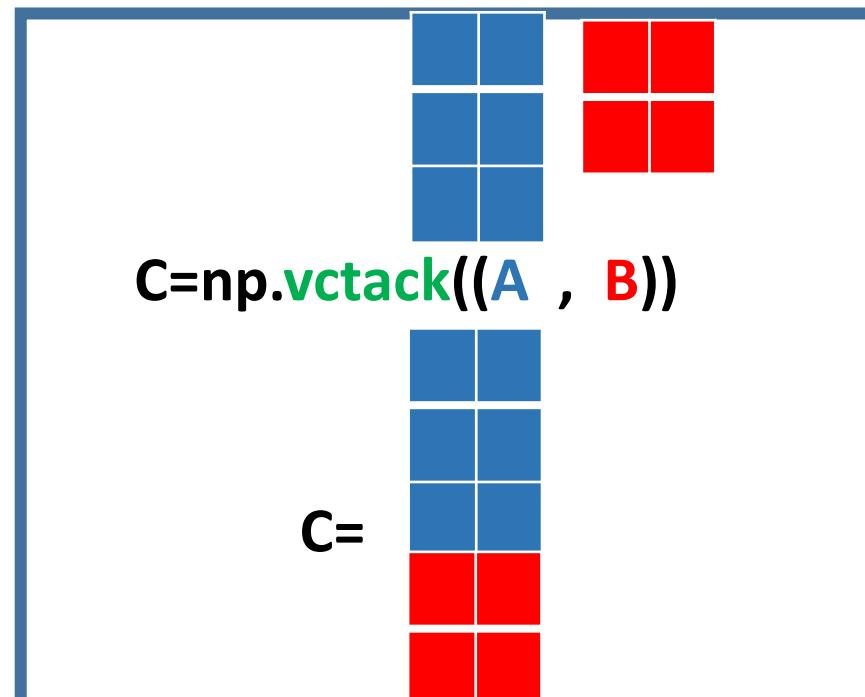
```
[[8 8 8 8 0 0]
 [8 8 8 8 0 0]
 [8 8 8 8 0 0]]
```

Le module NUMPY

NDArray – vstack

□ vstack

- Empilez les tableaux en séquence verticale (par rangée).
- Cela équivaut à la concaténation le long du premier axe après que les tableaux 1-D de forme ($N,$) ont été remodelés en $(1,N)$.



```
A=np.full((2,4),8)
B=np.zeros((3,4),dtype='int16')
print(A)
print(B)
```

```
[[8 8 8 8]
 [8 8 8 8]]
 [[0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]]
```

```
C=np.vstack((A,B))
print(C)
```

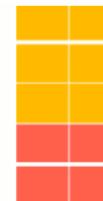
```
[[8 8 8 8]
 [8 8 8 8]
 [0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]]
```

NDArray – concatenate

concatenate

- concatène plusieurs tableaux sur un axe spécifié. Elle accepte une séquence de tableaux comme paramètre et les réunit en un seul tableau.

```
np.concatenate([Array A, Array B], axis = 0)
```



```
np.concatenate([Array A, Array B], axis = 1)
```



```
A=np.full((3,4),8)
B=np.zeros((3,2),dtype='int16')
print(A)
print(B)
```

```
[[8 8 8 8]
 [8 8 8 8]
 [8 8 8 8]]
 [[0 0]
 [0 0]
 [0 0]]
```

```
C=np.concatenate((A,B),axis=1)
print(C)
```

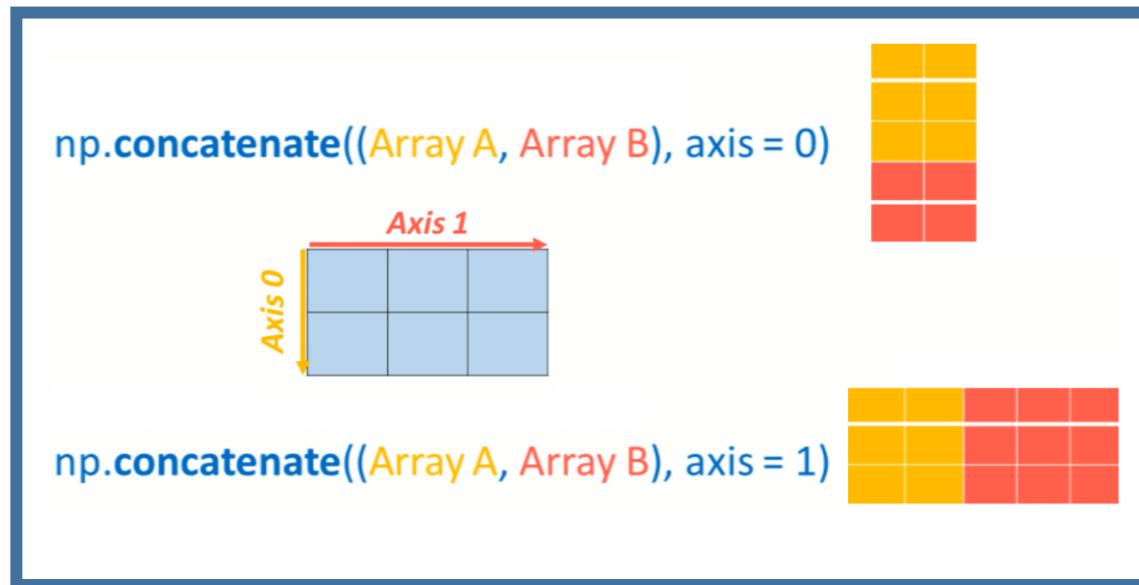
```
[[8 8 8 8 0 0]
 [8 8 8 8 0 0]
 [8 8 8 8 0 0]]
```

Le module NUMPY

NDArray – concatenate

concatenate

- concatène plusieurs tableaux sur un axe spécifié. Elle accepte une séquence de tableaux comme paramètre et les réunit en un seul tableau.



```
A=np.full((2,4),8)
B=np.zeros((3,4),dtype='int16')
print(A)
print(B)
```

```
[[8 8 8 8]
 [8 8 8 8]]
 [[0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]]
```

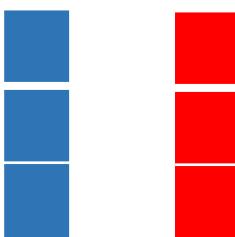
```
C=np.concatenate((A,B),axis=0)
print(C)
```

```
[[8 8 8 8]
 [8 8 8 8]
 [0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]]
```

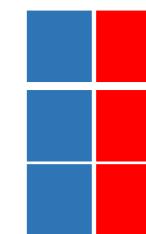
NDArray – vstack

□ Column_stack

- **numpy.column_stack()** La fonction est utilisée pour empiler des array 1-D sous forme de colonnes dans un array 2-D. Elle prend une séquence de array 1-D et les empile sous forme de colonnes pour créer un seul array 2-D.



C=np.column_stack((A , B))



```
import numpy as np
A= np.array([5, 7, 9, 1])
B= np.array([3, 8, 6, 2])
C=np.column_stack((A,B))
C
```



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

NDArray – redimensionnement

□ Reshape et resize

- NumPy a deux fonctions (et aussi des méthodes) pour changer les formes des tableaux - **reshape** et **resize**. Elles ont une différence significative.

Elle convertit un vecteur de 8 éléments en un tableau de la forme de (4, 2). Elle pourrait être exécutée avec succès parce que la quantité d'éléments avant et après le **reshape** est identique.

```
A = np.arange(8)  
print(A)
```

```
[0 1 2 3 4 5 6 7]
```

```
B=np.reshape(A, (2, 4))  
print(B)
```

```
[[0 1 2 3]  
 [4 5 6 7]]
```

NDArray – redimensionnement

❑ Reshape et resize

- NumPy a deux fonctions (et aussi des méthodes) pour changer les formes des tableaux - **reshape** et **resize**. Elles ont une différence significative.



Elle soulève l'erreur **ValueError** si les quantités sont différentes

```
A = np.arange(8)  
print(A)
```

```
[0 1 2 3 4 5 6 7]
```

```
B=np.reshape(arrayA, (3, 4))  
print(B)
```

ValueError: cannot reshape array of size 8 into shape (3,4)

NDArray – redimensionnement

□ Reshape et resize

resize est un peu similaire à **reshape** dans le sens de la conversion de forme. Mais il a quelques différences significatives.

- Si le nombre d'éléments du tableau cible n'est pas le même que celui du tableau original, il forcera à redimensionner mais ne provoquera pas d'erreurs.
- Si le nouveau tableau a plus de lignes, il répétera les données du tableau original mais ne lèvera pas l'erreur.
- Si le nombre d'éléments dans le nouveau tableau est plus petit, il récupère le nombre d'éléments dont il a besoin pour remplir le nouveau tableau dans l'ordre des lignes.

```
import numpy as np
```

```
A = np.arange(8)
```

```
B = np.resize(A, (2, 4))
```

```
print("A=", A)
```

```
print("B=\n", B)
```

```
A= [0 1 2 3 4 5 6 7]
```

```
B=
```

```
[[0 1 2 3]]
```

```
[4 5 6 7]]
```

```
D = np.resize(A, (4, 4))
```

```
print(D)
```

```
[[0 1 2 3]]
```

```
[4 5 6 7]]
```

```
[0 1 2 3]]
```

```
[4 5 6 7]]
```

```
D = np.resize(B, (3, 4))
```

```
print(D)
```

```
[[0 1 2 3]]
```

```
[4 5 6 7]]
```

```
[0 1 2 3]]
```

```
D = np.resize(B, (1, 4))
```

```
print(D)
```

```
[[0 1 2 3]]
```

NDArray – redimensionnement

□ Ravel

ravel utilisée pour convertir transformer des tableaux N-dimensionnels en tableaux à dimension unique.

```
A= np.array([[1,2,3],[4,5,6],[7,8,9]])
print("A=\n",A)
print(A.ravel())
```

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

NDArray – redimensionnement

□ Ravel

ravel utilisée pour convertir transformer des tableaux N-dimensionnels en tableaux à dimension unique.

```
A= np.array([[1,2,3],[4,5,6],[7,8,9]])
print("A=\n",A)
print(A.ravel())
```

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

NDArray – redimensionnement

□ Roll

utilisée pour dérouler les éléments du tableau le long d'un axe spécifié.
(axis=0: le long des lignes).

```
C
```

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

```
R=np.roll(C, -1, axis=0)
```

```
R
```

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

```
R=np.roll(C, -2, axis=0)
```

```
R
```

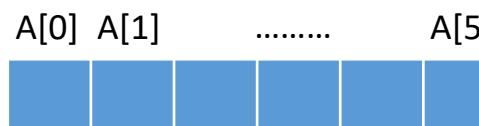
```
array([[9, 6],  
       [1, 2],  
       [5, 3],  
       [7, 8]])
```

NDArray – Accès aux éléments d'un tableau

□ Indexation et slicing

- Accéder à des éléments ou à des sous-tableaux va nous permettre de leur appliquer des fonctions vectorisées
- la manière d'accéder aux éléments d'un tableau numpy dépend de la forme du tableau (**shape**)

1- accès à un tableau de dimension 1 (Indexing)



```
A = np.arange(12)
print("A= ",A)
print("A[2]= ",A[2])
```

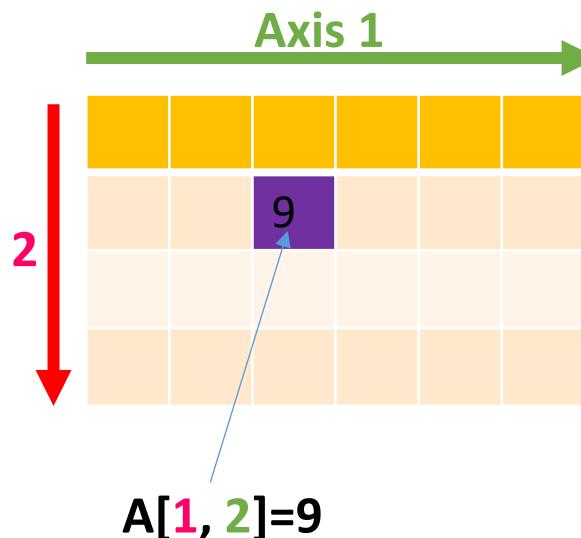
```
A= [ 0  1  2  3  4  5  6  7  8  9 10 11]
A[2]=  2
```

NDArray – Accès aux éléments d'un tableau

□ Indexation et slicing

1- accès à un tableau de deux dimension (Indexing)

A[ligne, colonne]



```
A = np.arange(12)
print("A= ",A)
print("A[2]= ",A[2])
```

```
A= [ 0  1  2  3  4  5  6  7  8  9 10 11]
A[2]=  2
```

Le module NUMPY

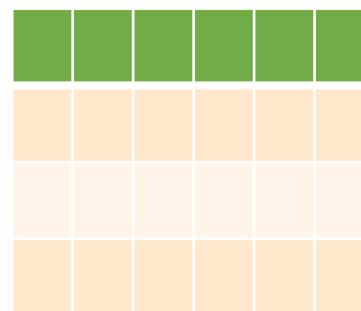
NDArray – Accès aux éléments d'un tableau

❑ slicing

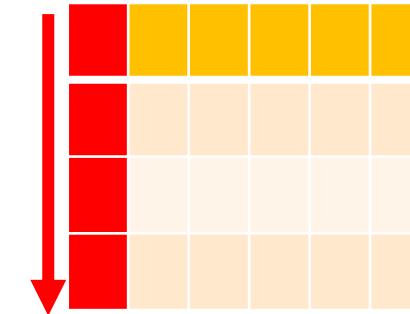
A[ligne, colonne]

A[Début : fin , Début : fin]

A[0 , :]



A[:, 0]



```
A = np.arange(12).reshape((3, 4))  
A
```

```
array([[ 0,  1,  2,  3],  
       [ 4,  5,  6,  7],  
       [ 8,  9, 10, 11]])
```

```
A[:,0]
```

```
array([0, 4, 8])
```

```
A[0,:]
```

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

Le module NUMPY

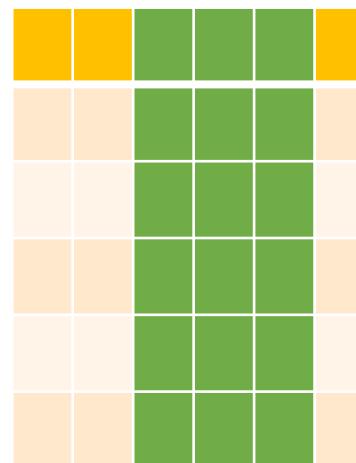
NDArray – Accès aux éléments d'un tableau

❑ slicing

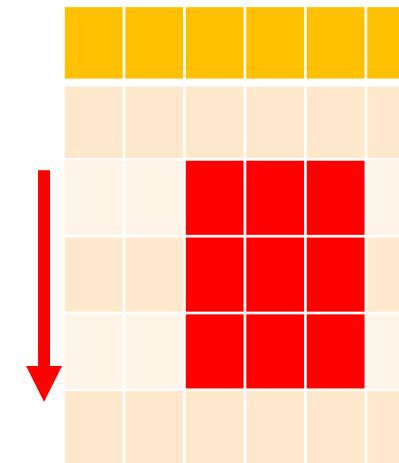
A[ligne, colonne]

A[Début : fin , Début : fin]

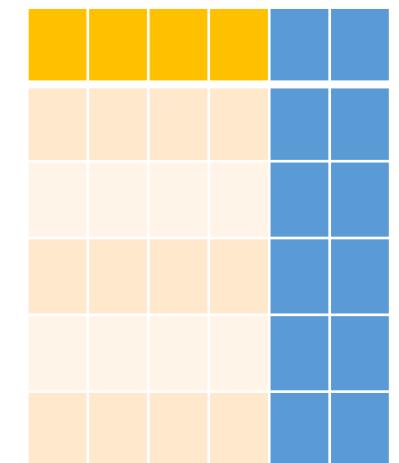
A[:, 2 : 5]



A[2:5 , 2: 5]



A[:, -2:]



```
A = np.arange(16).reshape((4, 4))
```

```
A
```

```
array([[ 0,  1,  2,  3],  
       [ 4,  5,  6,  7],  
       [ 8,  9, 10, 11],  
       [12, 13, 14, 15]])
```

```
A[1:3,1:4]
```

```
array([[ 5,  6,  7],  
       [ 9, 10, 11]])
```

```
A[ :, -2: ]
```

```
array([[ 2,  3],  
       [ 6,  7],  
       [10, 11],  
       [14, 15]])
```

Le module NUMPY

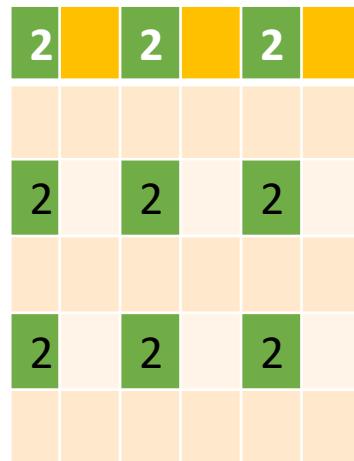
NDArray – Accès aux éléments d'un tableau

❑ slicing

A[ligne, colonne]

A[Début : fin : pas , Début : fin : pas]

A[::2, ::2]=2



```
A=np.zeros((6,6))
```

```
A
```

```
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., 0., 0., 0., 0., 0.],  
       [0., 0., 0., 0., 0., 0.]])
```

```
A[::2,::2]=2
```

```
A
```

```
array([[2., 0., 2., 0., 2., 0.],  
       [0., 0., 0., 0., 0., 0.],  
       [2., 0., 2., 0., 2., 0.],  
       [0., 0., 0., 0., 0., 0.],  
       [2., 0., 2., 0., 2., 0.],  
       [0., 0., 0., 0., 0., 0.]])
```

Le module NUMPY

NDArray – Accès aux éléments d'un tableau

❑ slicing

```
A=np.zeros((6,6))  
A
```

```
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., 0., 0., 0., 0., 0.],  
       [0., 0., 0., 0., 0., 0.]])
```

```
A[::2,::2]=2  
A
```

```
array([[2., 0., 2., 0., 2., 0.],  
       [0., 0., 0., 0., 0., 0.],  
       [2., 0., 2., 0., 2., 0.],  
       [0., 0., 0., 0., 0., 0.],  
       [2., 0., 2., 0., 2., 0.],  
       [0., 0., 0., 0., 0., 0.]])
```

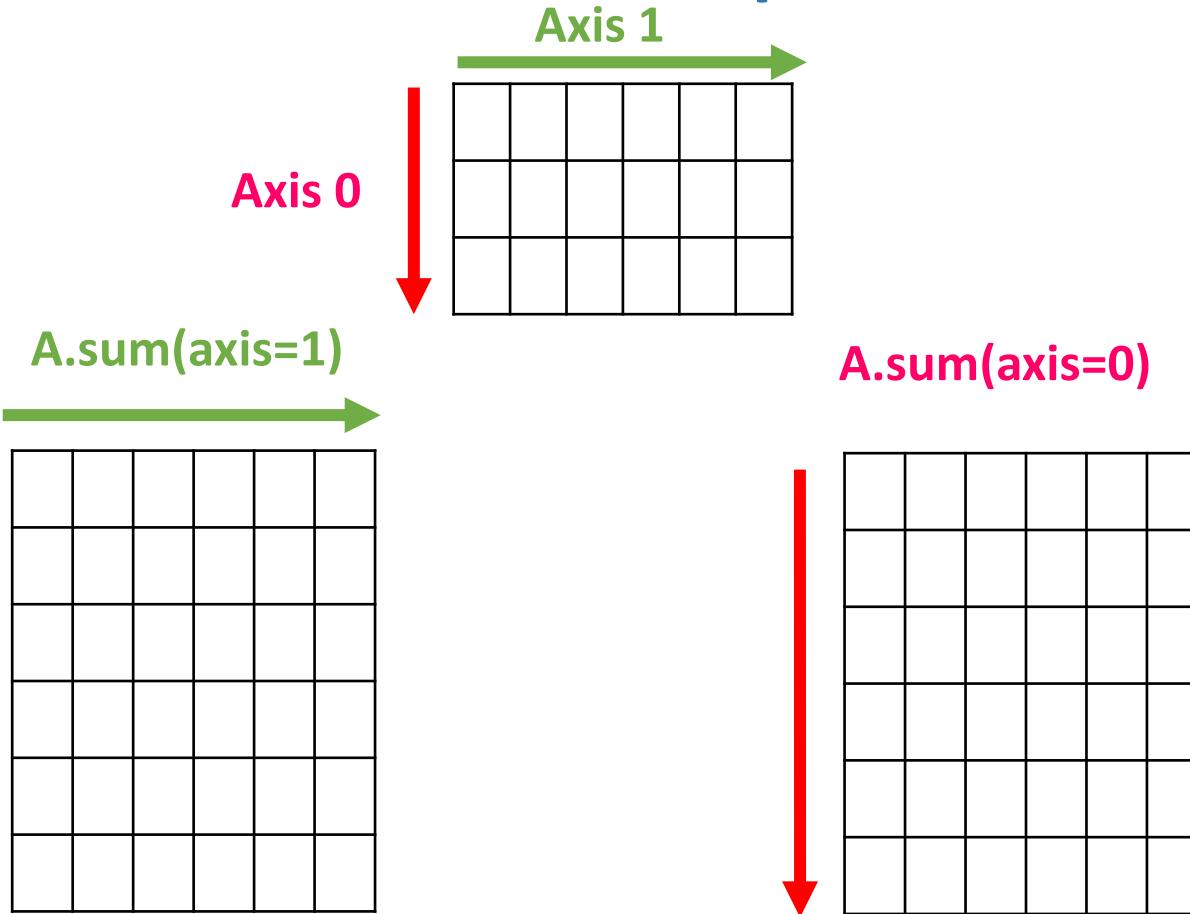
```
A[A<1]=47  
A
```

```
array([[ 2., 47.,  2., 47.,  2., 47.],  
       [47., 47., 47., 47., 47., 47.],  
       [ 2., 47.,  2., 47.,  2., 47.],  
       [47., 47., 47., 47., 47., 47.],  
       [ 2., 47.,  2., 47.,  2., 47.],  
       [47., 47., 47., 47., 47., 47.]])
```

Le module NUMPY

NDArray -

□ Calcul Mathématiques



```
A = np.arange(16).reshape((4, 4))  
A
```

```
array([[ 0,  1,  2,  3],  
       [ 4,  5,  6,  7],  
       [ 8,  9, 10, 11],  
       [12, 13, 14, 15]])
```

```
A.sum(axis=0)
```

```
array([24, 28, 32, 36])
```

```
A.sum(axis=1)
```

```
array([ 6, 22, 38, 54])
```

Le module NUMPY

NDArray –

□ Calcul Mathématiques

- A.sum(axis)
- A.cumsum(axis)
- A.prod(axis)
- A.cumprod(axis)
- A.min(axis)
- A.max(axis)
- A.sort(axis)
- A.argmin (axis)
- A.argmax (axis)
- A.argsort (axis)

```
A=np.random.randint(1,10,[3,4])
```

```
A
```

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

```
A.sum(axis=1)
```

```
array([21, 13, 20])
```

```
A.max(axis=1)
```

```
array([7, 8, 9])
```

```
A.argmin(axis=1)
```

```
array([3, 0, 2], dtype=int64)
```

```
A.prod(axis=1)
```

```
array([630, 32, 432])
```

```
A.sort(axis=1)
```

```
A
```

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

Le module NUMPY

NDArray –

□ Calcul Mathématiques

➤ A.argsort (axis)

The diagram illustrates the mapping of indices from a sorted array to an unsorted array. At the top, there is an unsorted array:

0	1	2	3	4	5	6
22	9	0	4	7	6	14

Below it is a sorted array:

2	3	5	4	1	6	0
0	4	6	7	9	14	22

Blue arrows show the correspondence between the indices of the sorted array and the values in the unsorted array. For example, the index 0 of the sorted array corresponds to the value 22 in the unsorted array, and the index 6 corresponds to 0.

Argsort() retourne les positions
dans l'ordre de tri du tableau A, sans modifier le tableau

```
A=np.array([22,9,0,4,7,6,14])  
A.sort()  
A
```

```
array([ 0,  4,  6,  7,  9, 14, 22])
```

```
C=np.array([22,9,0,4,7,6,14])  
print(C.argsort())  
print(C)
```

```
[2 3 5 4 1 6 0]  
[22 9 0 4 7 6 14]
```

NDArray –

□ Calcul Mathématiques

➤ Coefficient de corrélation

```
import numpy as np
A=np.random.randint(1,40,[3,4])
A
array([[23, 28, 25, 11],
       [13, 31, 36,  2],
       [28,  4, 19,  2]])
```

```
np.corrcoef(A)
```

```
array([[1.          , 0.86444504, 0.36089282],
       [0.86444504, 1.          , 0.15202286],
       [0.36089282, 0.15202286, 1.          ]])
```

```
np.corrcoef(A[1,:,:],A[2,:,:])
```

```
array([[1.          , 0.15202286],
       [0.15202286, 1.          ]])
```

```
np.corrcoef(A[0,:,:],A[1,:,:])
```

```
array([[1.          , 0.86444504],
       [0.86444504, 1.          ]])
```

Le module NUMPY

NDArray –

□ Calcul Mathématiques

- **Unique:** récupère toutes les valeurs uniques dans le tableau NumPy donné et trie ces valeurs uniques.
-

return_index	Booléen. Si <code>True</code> , retourne un tableau d'indices de la première occurrence de chaque valeur unique.
return_counts	Booléen. Si <code>True</code> , retourne un tableau du nombre de chaque valeur unique.
axis	trouver des lignes (<code>axis=0</code>) ou des colonnes (<code>axis=1</code>) uniques. Par défaut, les éléments uniques sont récupérés dans le tableau aplati.

```
A=np.random.randint(1,20,[2,3])
A
array([[ 6, 10,  3],
       [10,  3,  2]])
```

```
np.unique(A,return_counts=True)
(array([ 2,  3,  6, 10]), array([1, 2, 1, 2], dtype=int64))
```

```
A=np.array([[2,5,1],[3,4,8],[2,5,1],[9,3,6]])
A
array([[2, 5, 1],
       [3, 4, 8],
       [2, 5, 1],
       [9, 3, 6]])
```

```
np.unique(A,axis=0)
array([[2, 5, 1],
       [3, 4, 8],
       [9, 3, 6]])
```

NDArray –

□ Calcul Mathématiques

Nan: Le nan est une constante qui indique que la valeur donnée n'est pas légale - **Not a Number.**

- Notez que nan et NULL sont deux choses différentes. La valeur NULL indique quelque chose qui n'existe pas et qui est vide.
- En Python, nous traitons de telles valeurs très fréquemment dans différents objets. Il est donc nécessaire de détecter de telles constantes.
- En Python, nous avons la fonction **isnan()**, qui peut vérifier les valeurs **nan**. Et cette fonction est disponible en deux modules - NumPy et math. La fonction **isna()** du module pandas peut également vérifier les valeurs **nan**.

```
A=np.array([[2,5,1],[3,np.nan,8],[2,5,1],[9,3,np.nan]])  
A
```

```
array([[ 2.,  5.,  1.],  
       [ 3.,  nan,  8.],  
       [ 2.,  5.,  1.],  
       [ 9.,  3.,  nan]])
```

```
#vérifier l'existence des NaN  
np.isnan(A)
```

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

```
#calculer le nombre des NaN  
np.isnan(A).sum()
```

2

```
# remplacer les NaN par des 0  
A[np.isnan(A)]=0  
A
```

```
array([[2., 5., 1.],  
       [3., 0., 8.],  
       [2., 5., 1.],  
       [9., 3., 0.]])
```

Le module NUMPY

NDArray –

□ Calcul Mathématiques

- **Numpy.mean()**: calcule la moyenne du tableau donné le long de l'axe spécifié
- **Numpy.nanmean()** : calcule la moyenne du tableau donné le long de l'axe spécifié en ignorant les valeurs NaN
- **Numpy.std()** : calcule l'écart type du tableau donné le long de l'axe spécifié
- **Numpy.nanstd()** : calcule l'écart type du tableau donné le long de l'axe spécifié en ignorant les valeurs NaN
- **Numpy.var()** : calcule variance du tableau donné le long de l'axe spécifié
- **Numpy.nanvar()** : calcule variancedu tableau donné le long de l'axe spécifié en ignorant les valeurs NaN

```
A=np.array([[2,5,1],[3,4,8],[2,5,1],[9,3,6]])
```

```
A
```

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

```
np.mean(A)
```

```
4.083333333333333
```

```
np.mean(A, axis=0)
```

```
array([4. , 4.25, 4. ])
```

```
A=np.array([[2,5,1],[3,np.nan,8],[2,5,1],[9,3,np.nan]])
```

```
A
```

```
array([[ 2.,  5.,  1.],  
       [ 3.,  nan,  8.],  
       [ 2.,  5.,  1.],  
       [ 9.,  3.,  nan]])
```

```
np.nanmean(A)
```

```
4.083333333333333
```

Le module NUMPY

NDArray –

□ Calcul Mathématiques

```
A=np.array([[2,5,1],[3,1,0]])  
A
```

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

```
A+3
```

```
array([[5, 8, 4],  
       [6, 4, 3]])
```

```
A*3
```

```
array([[ 6, 15,  3],  
       [ 9,  3,  0]])
```

```
B=np.full((2,3),2)
```

```
B
```

```
array([[2, 2, 2],  
       [2, 2, 2]])
```

```
A=np.array([[2,5,1],[3,1,0]])
```

```
A
```

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

```
A+B
```

```
array([[4, 7, 3],  
       [5, 3, 2]])
```

Le module NUMPY

NDArray –

□ Calcul Mathématiques

- **Numpy.dot:** calcule le produit en **points** de deux tableaux d'entrée.

```
A=np.array([[2,0,1],[1,1,0]])  
B=np.array([[5,3,1],[2,1,0]])  
print('A=\n',A)  
print('B=\n',B)
```

```
A=  
[[2 0 1]  
 [1 1 0]]  
B=  
[[5 3 1]  
 [2 1 0]]
```

```
A*B
```

```
array([[10, 0, 1],  
 [ 2, 1, 0]])
```

```
A.dot(B.T)
```

```
array([[11, 4],  
 [ 8, 3]])
```

MATPLOTLIB

Matplotlib

Matplotlib Il s'agit de l'une des bibliothèques python les plus utilisées pour représenter des graphiques en 2D. Elle permet de produire une grande variété de graphiques et ils sont de grande qualité.

Le module **pyplot** de **matplotlib** est l'un de ses principaux modules. Il regroupe un grand nombre de fonctions qui servent à créer des graphiques et les personnaliser (travailler sur les axes, le type de graphique, sa forme et même rajouter du texte).

Graphiques simples

Important : Importer le module `matplotlib.pyplot`

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

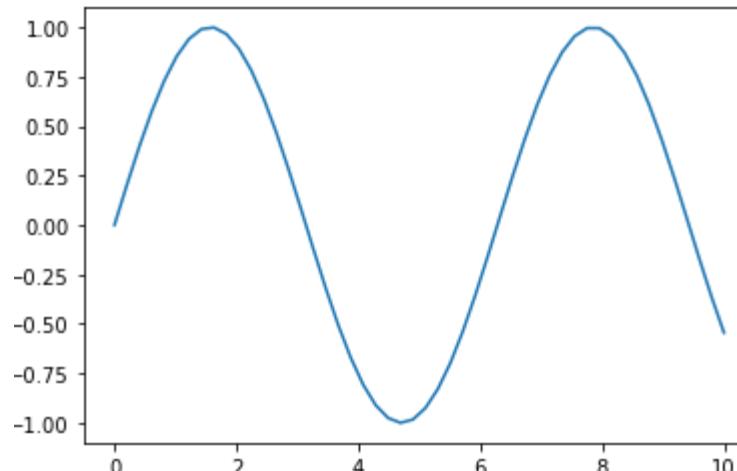
Il existe deux méthodes pour réaliser des graphes

Utiliser la fonction `plt.plot` ou bien OOP

```
import matplotlib.pyplot as plt  
x = np.linspace(0,20,100)  
y = np.sin(x)  
plt.plot(x, y)  
plt.show()
```

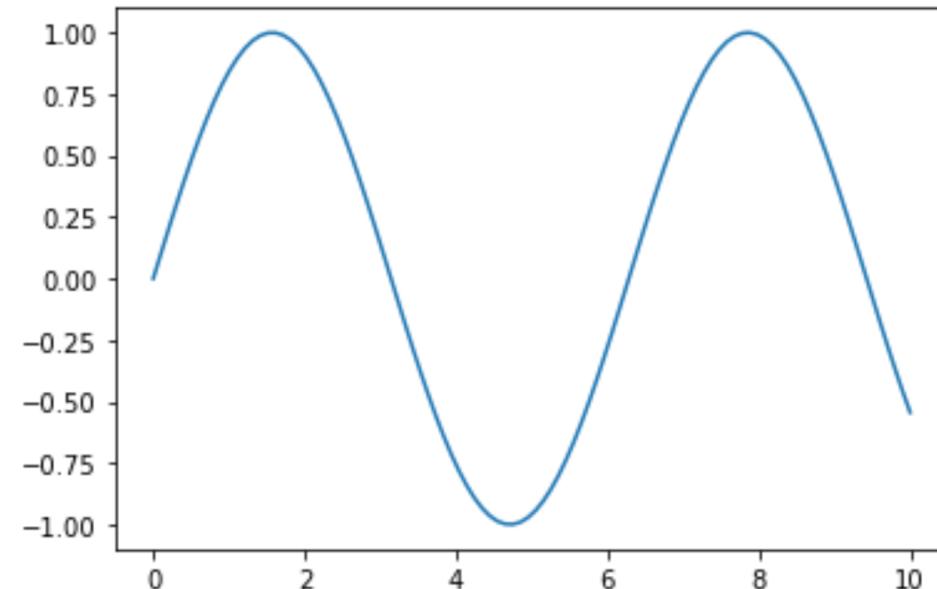
```
import matplotlib.pyplot as plt  
x = np.linspace(0,20,100)  
y = np.sin(x)  
fig,ax=plt.subplots()  
ax.plot(x,y)  
plt.show()
```

On obtient le même résultat



Graphiques simples

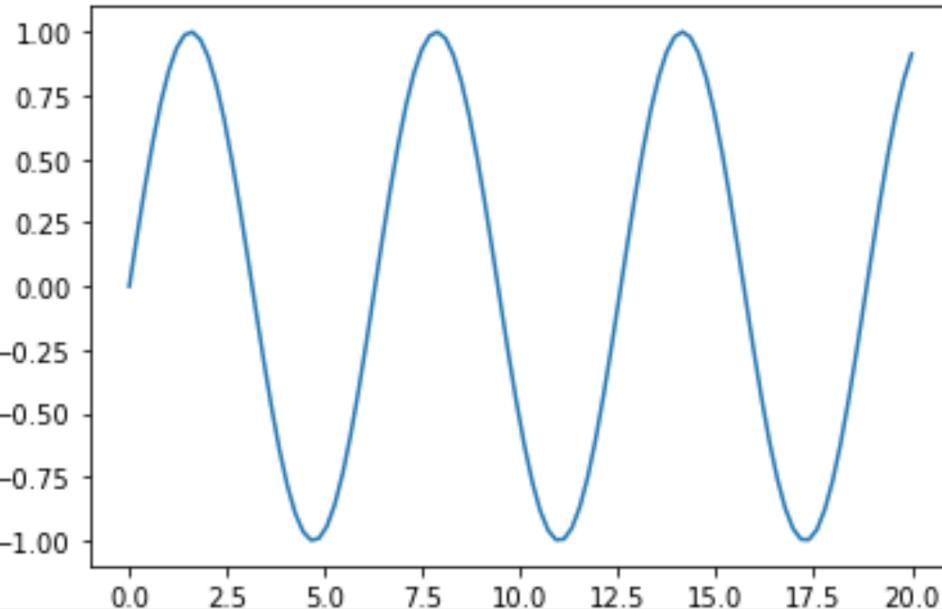
```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0,10,100)
y = np.sin(x)
plt.plot(x, y)
plt.show()
```



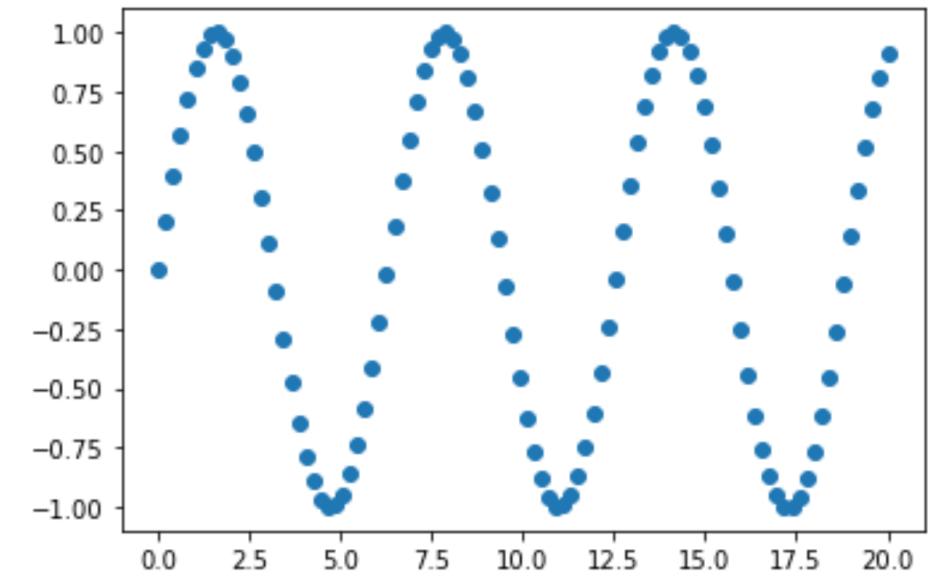
Attention: x et y doivent avoir la même dimension

Graphiques simples

```
import matplotlib.pyplot as plt  
x = np.linspace(0,20,100)  
y = np.sin(x)  
plt.plot(x, y)  
plt.show()
```



```
import matplotlib.pyplot as plt  
x = np.linspace(0,20,100)  
y = np.sin(x)  
plt.scatter(x, y)  
plt.show()
```



Attention: x et y doivent avoir la même dimension

Styles Graphiques

Il existe beaucoup de styles à ajouter aux graphiques.

Voici les plus importants à retenir :

label permet de légendier un graphique, c'est-à-dire d'attribuer un nom à une courbe. Il suffit alors de choisir d'afficher la légende avec la commande `legend` qui est à placer juste avant `show`.

c : couleur de la ligne

lw : (linewidth) épaisseur de la ligne (pour les graphiques `plot`)

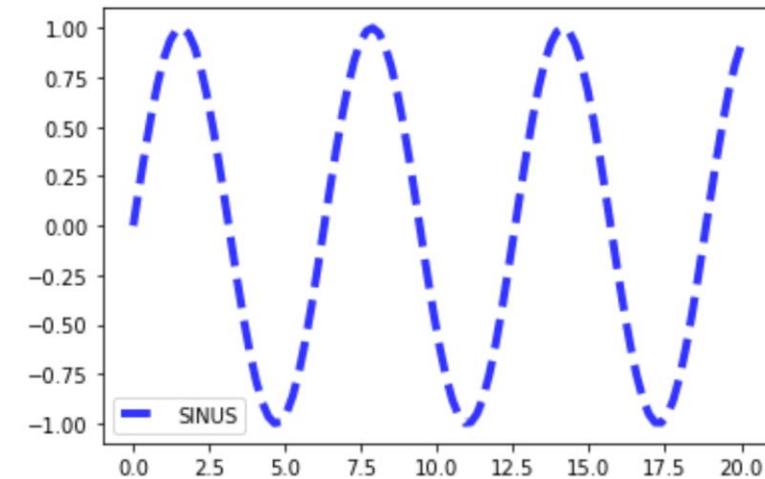
ls : (linestyle) style de la ligne (pour les graphiques `plot`)

size : taille du point (pour les graphiques `scatter`)

marker : style de points (pour les graphiques `scatter`)

alpha : transparence du graphique

```
plt.plot(x, y, label='SINUS', c='blue', lw=4, ls='--', alpha=0.8)
plt.legend()
plt.show()
```



```
from matplotlib import lines
print(lines.lineStyles.keys())

dict_keys(['-', '--', '-.', ':', 'None', ' ', ''])

print(lines.lineStyles.values())

dict_values(['_draw_solid', '_draw_dashed', '_draw_dash_dot', '_draw_dotted', '_draw_nothing', '_draw_nothing', '_draw_nothing'])
```

Cycle de vie d'une figure

Pour créer des figures proprement, on doit suivre le cycle de vie suivant :

plt.figure(figsize())

plt.plot()

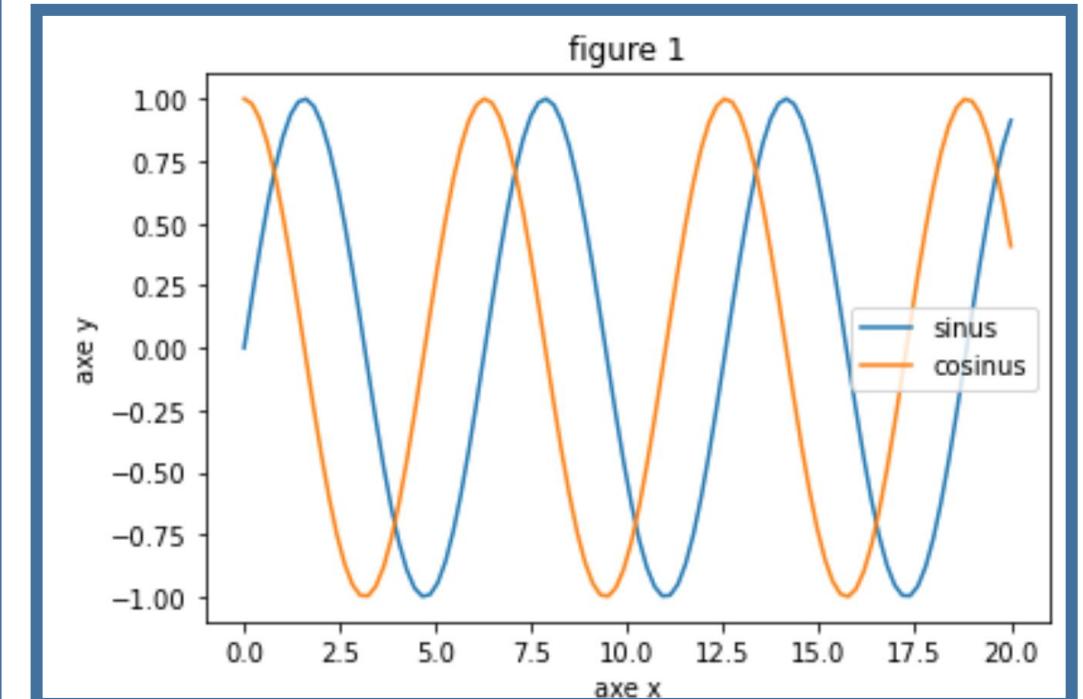
Extras (titre, axes, legendes)

plt.show()

```
x = np.linspace(0, 20, 100)

plt.figure() # Création d'une figure
# première courbe
plt.plot(x, np.sin(x), label='sinus')
# deuxième courbe
plt.plot(x, np.cos(x), label='cosinus')
# Extra information
plt.title('figure 1') # titre
plt.xlabel('axe x') # axes
plt.ylabel('axe y') # axes
plt.legend() # Legende

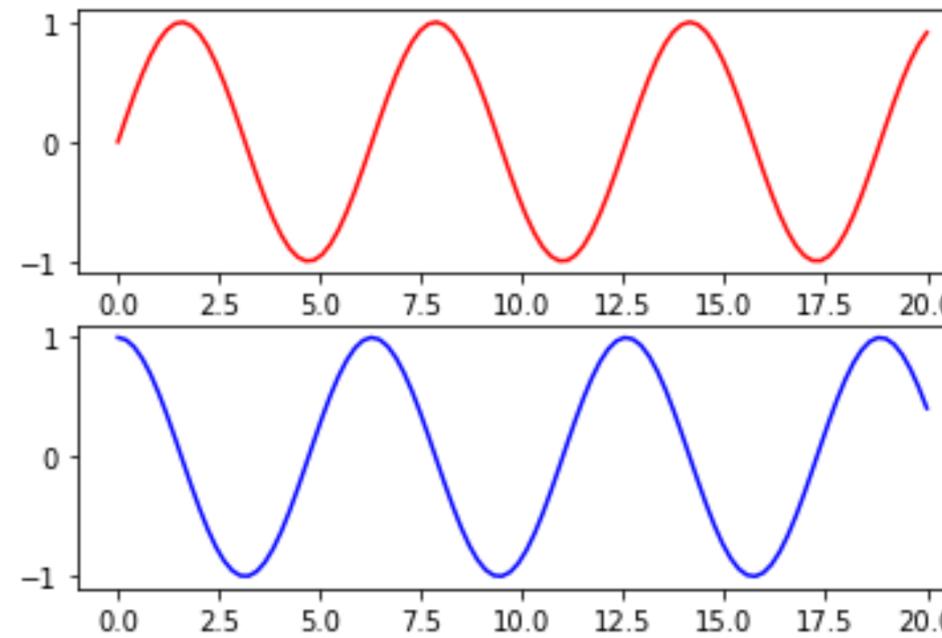
# sauvegarde la figure dans le répertoire de travail
plt.savefig('figure.png')
plt.show() # affiche la figure
```



Subplot

Les subplot permettent de créer plusieurs graphiques sur une même figure:

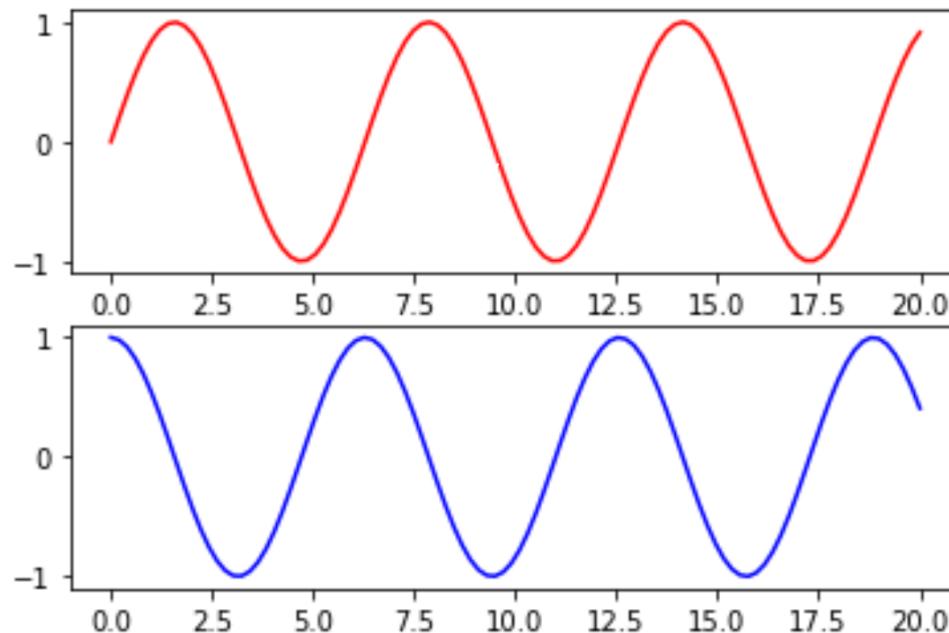
```
plt.figure()
plt.subplot(2, 1, 1)
plt.plot(x, np.sin(x), c='red')
plt.subplot(2, 1, 2)
plt.plot(x, np.cos(x), c='blue')
plt.show()
```



Subplots

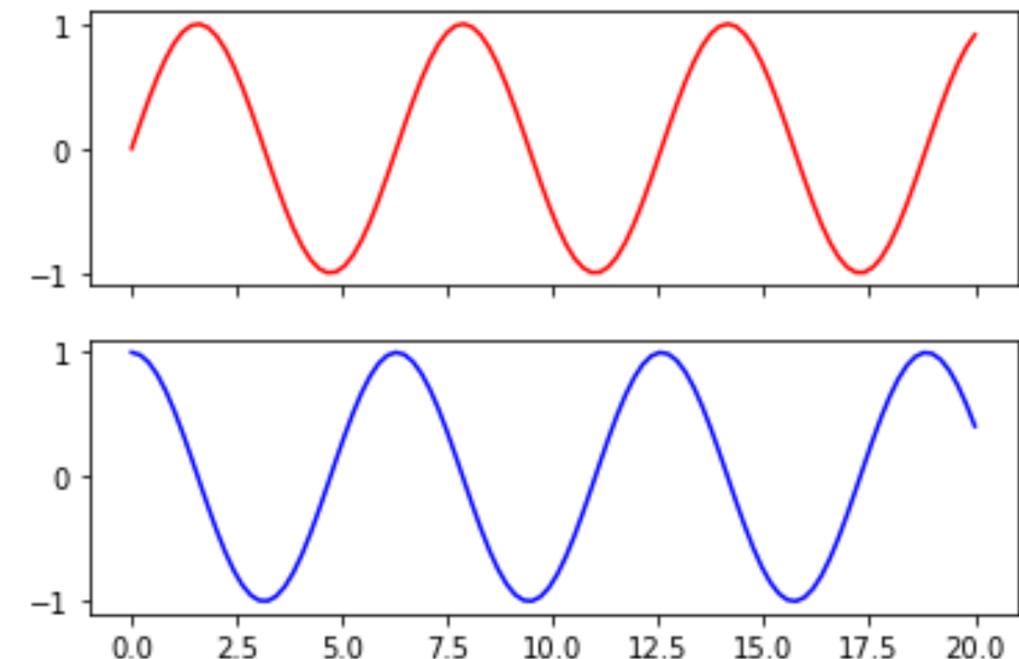
Les subplots permettent de créer plusieurs graphiques sur une même figure:

```
plt.figure()  
plt.subplot(2, 1, 1)  
plt.plot(x, np.sin(x), c='red')  
plt.subplot(2, 1, 2)  
plt.plot(x, np.cos(x), c='blue')  
plt.show()
```



Autre méthode en utilisant la POO:

```
# partage le même axe pour les subplots  
fig, ax = plt.subplots(2, 1, sharex=True)  
ax[0].plot(x, np.sin(x), c='red')  
ax[1].plot(x, np.cos(x), c='blue')  
plt.show()
```



Graphique de classification

Nous allons utiliser pour cet exemple le dataset des blobs gaussiens isotropes générés par le module `sklearn`.

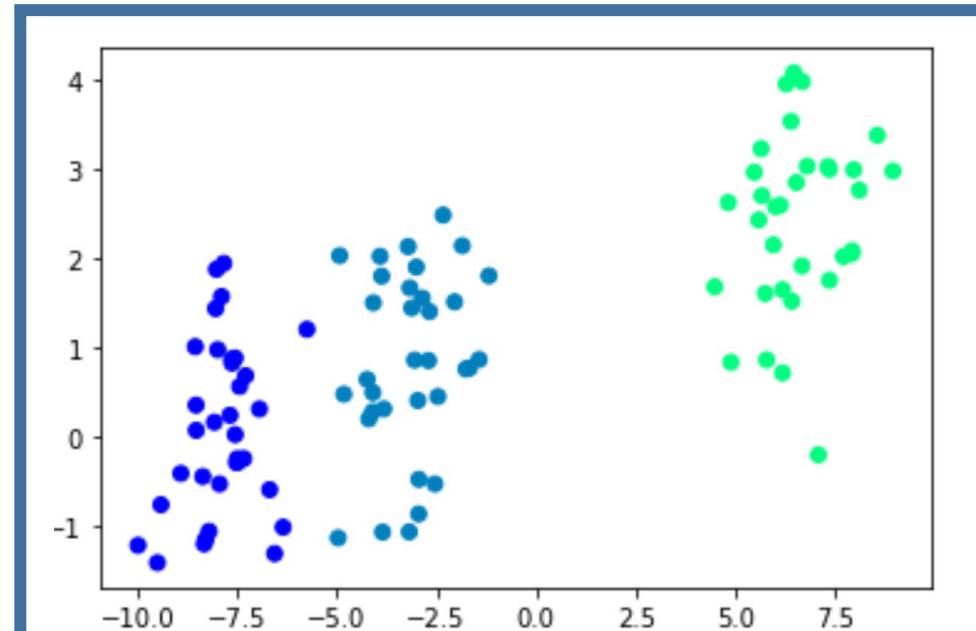
Parameter `n_samples` : default=100 S'il s'agit d'un nombre entier, il s'agit du nombre total de points également répartis entre les grappes.

`n_features` : default=2 Le nombre de caractéristiques pour chaque échantillon.

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs

X,y=make_blobs(n_samples=100, n_features=2)
y=y.reshape((len(y),1))

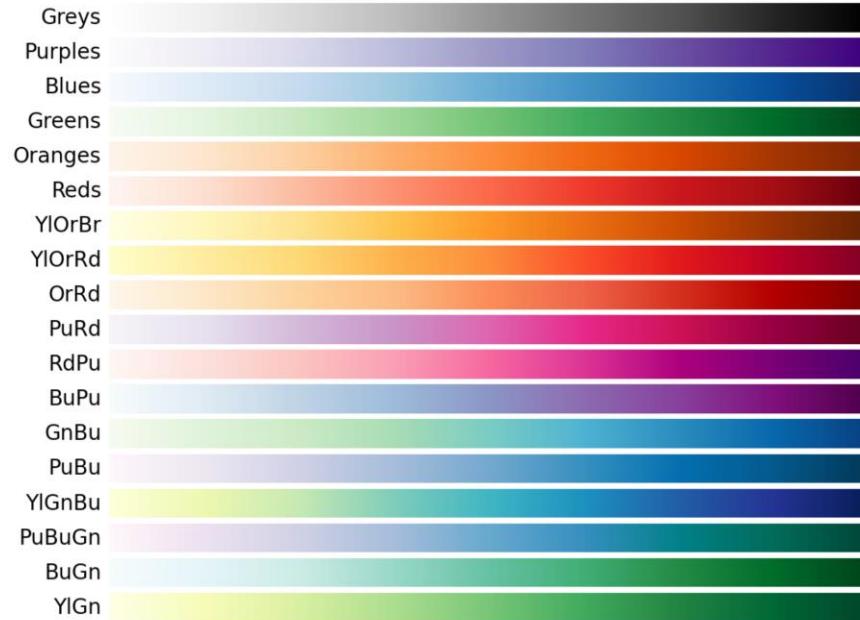
plt.scatter (X[:,0],X[:,1],c=y,cmap='winter')
plt.show()
```



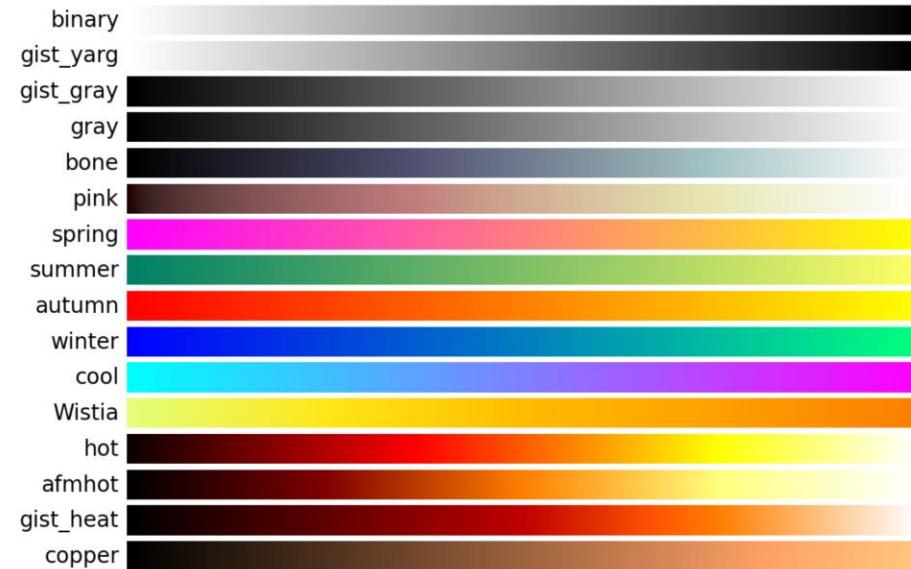
Graphique de classification

ColorMap

Sequential colormaps



Sequential (2) colormaps



Diverging colormaps

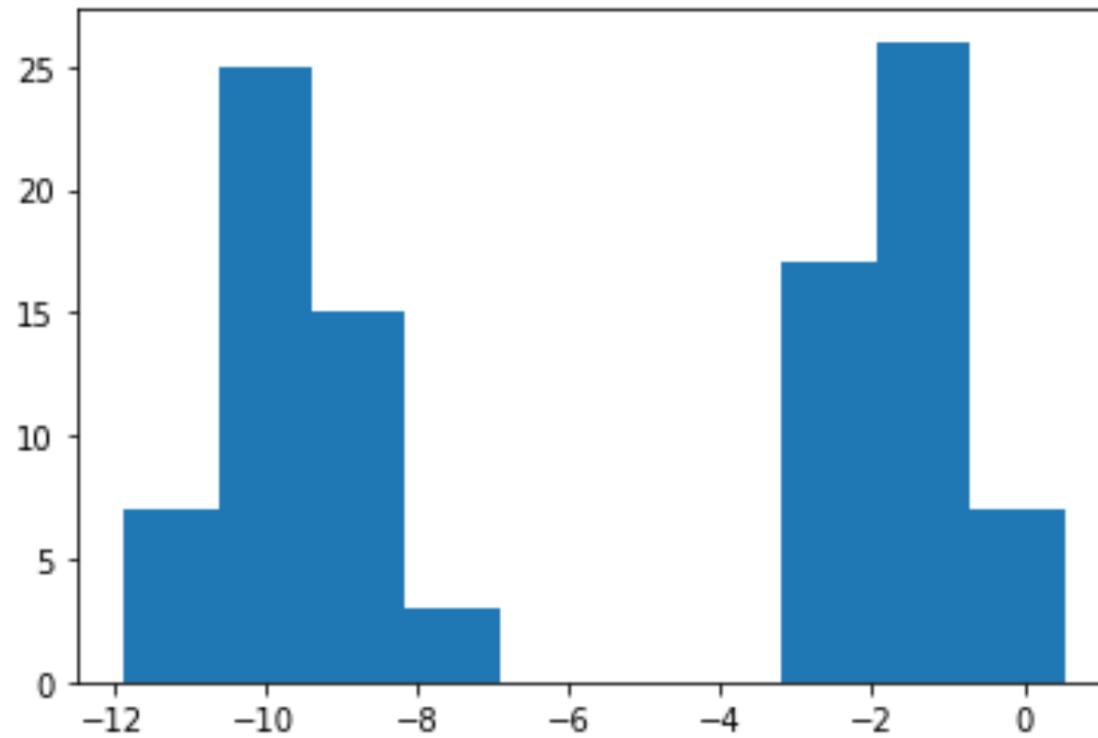


Les histogrammes

Revenons au dataset

```
plt.hist(X[:,0])
```

```
(array([ 7., 25., 15., 3., 0., 0., 0., 17., 26., 7.]),  
 array([-11.85569437, -10.61752338, -9.37935239, -8.14118141,  
        -6.90301042, -5.66483943, -4.42666845, -3.18849746,  
        -1.95032647, -0.71215549,  0.5260155]),  
<BarContainer object of 10 artists>)
```

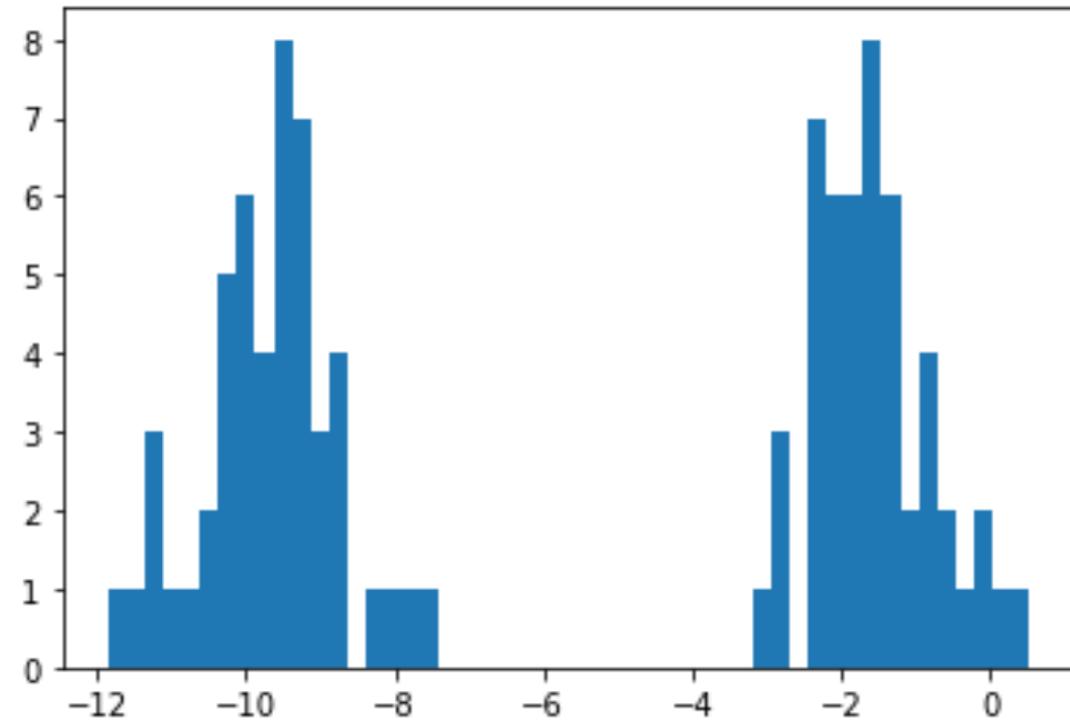


Les histogrammes

Revenons au dataset

```
plt.hist(X[:,0], bins=50)
```

Le paramètre **bins** permet d'augmenter le nombre de section dans notre histogramme.



Les histogrammes

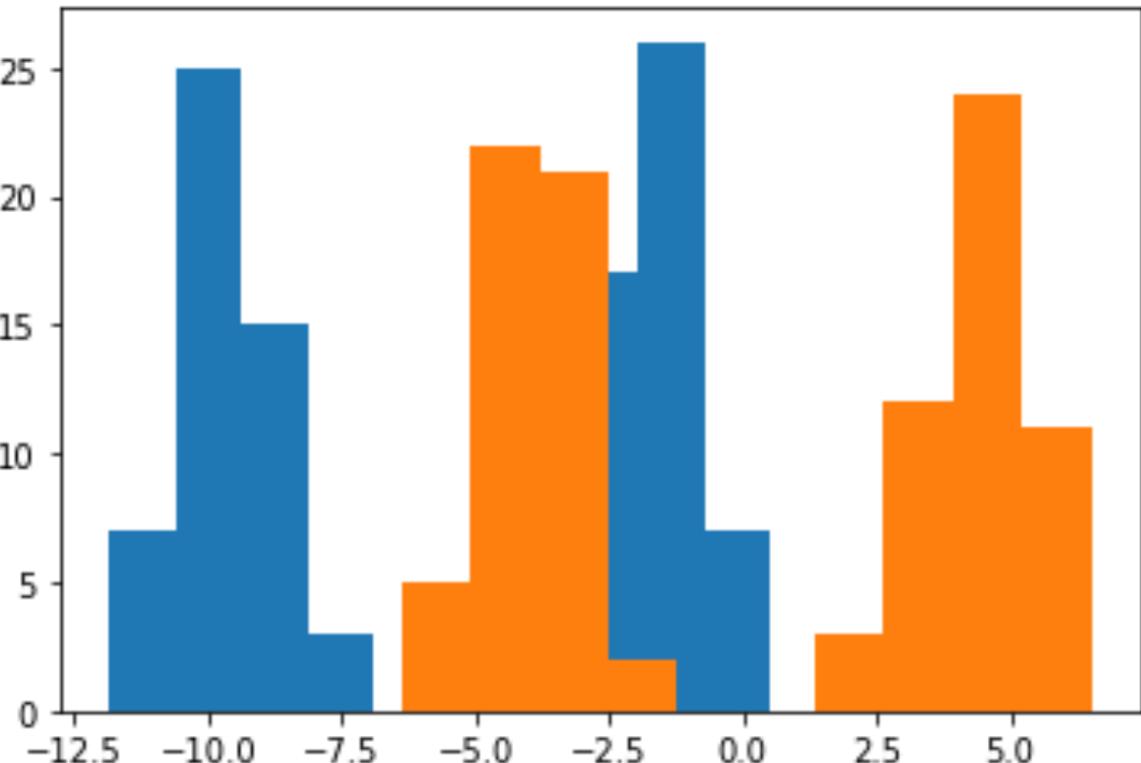
Revenons au dataset

```
plt.hist(X[:,0], bins=50)
```

Le paramètre bins permet d'augmenter le nombre de section dans notre histogramme

On peut aussi créer plusieurs Histogrammes en même temps

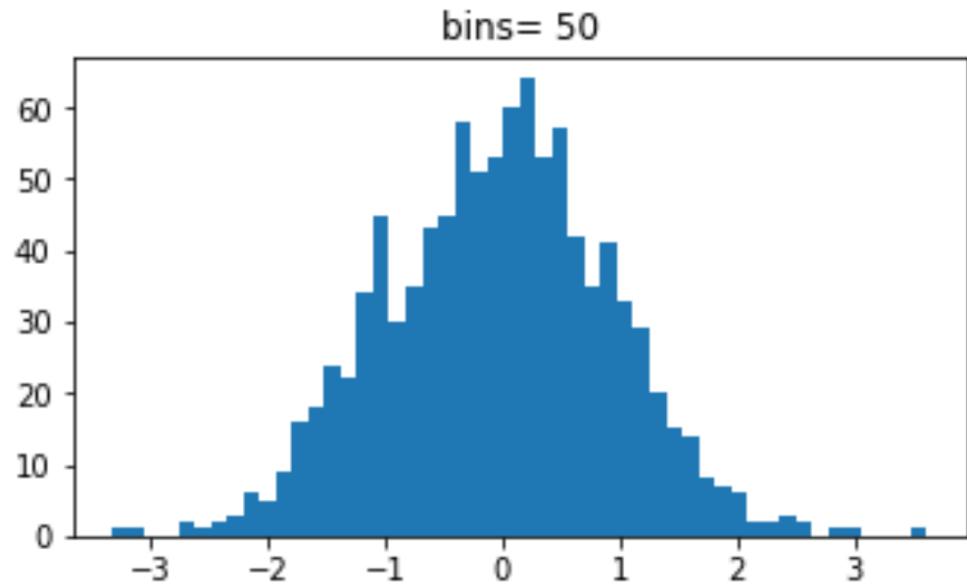
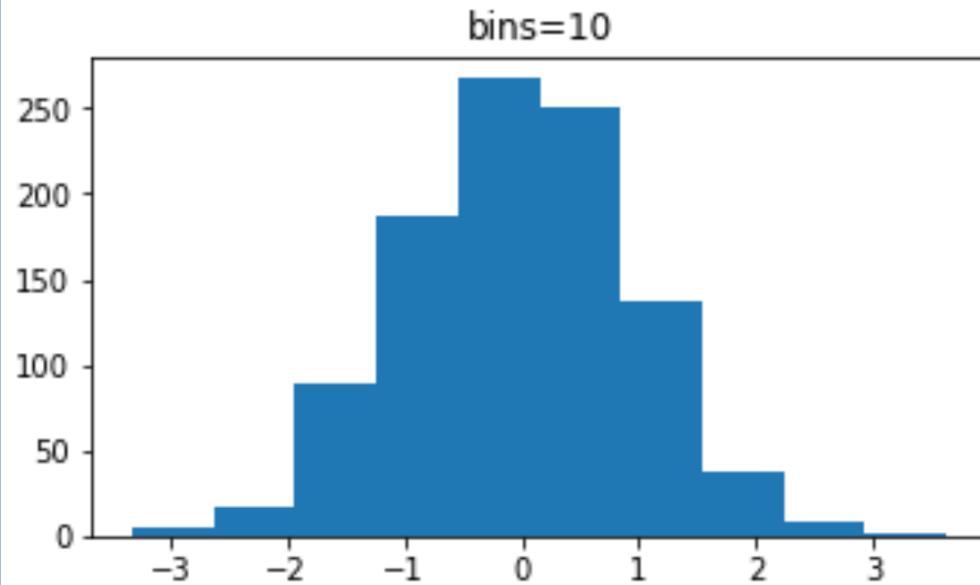
```
plt.hist(X[:,0])  
plt.hist(X[:,1])
```



Les histogrammes: autre exemple

```
x = np.random.randn(1000)

plt.figure(figsize=(12, 3))
plt.subplot(1,2,1)
plt.hist(x, bins=10)
plt.title('bins=10')
plt.subplot(1,2,2)
plt.hist(x, bins=50)
plt.title('bins= 50')
plt.show()
```

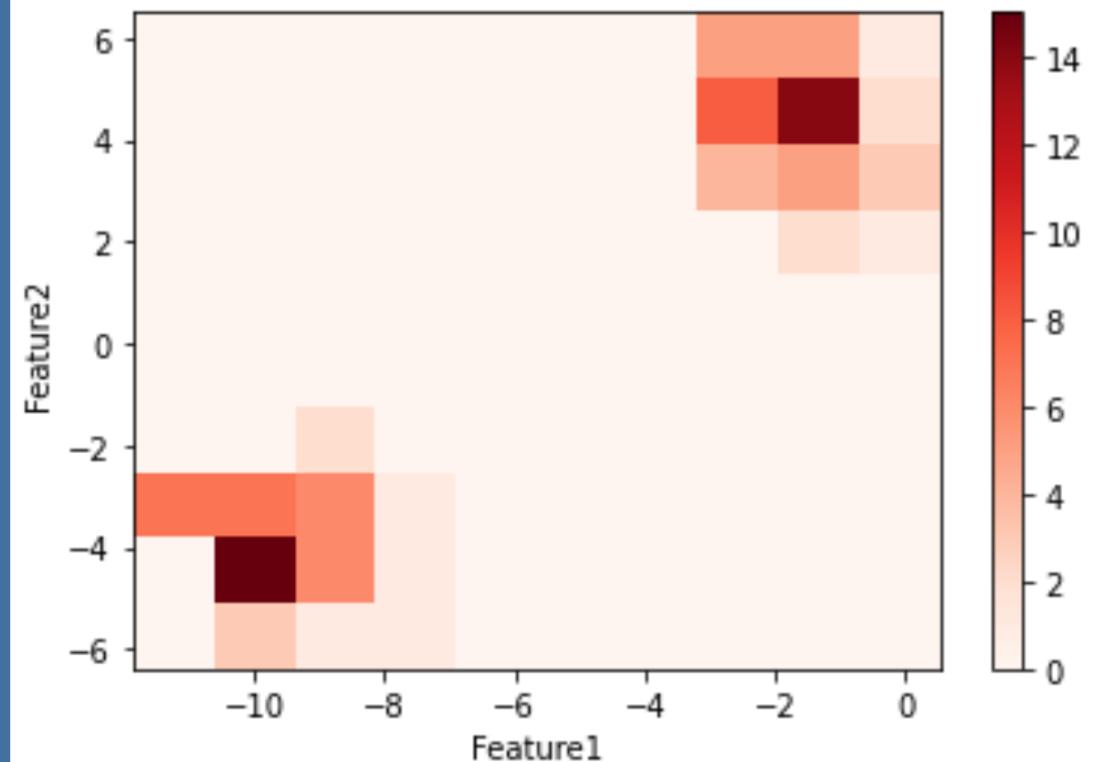


Les histogrammes 2D

On peut tracer des histogrammes en 2D pour visualiser la distribution des données lorsqu'elles suivent 2 variables, on utilise alors la fonction `plt.hist2d`

```
plt.hist2d(X[:,0], X[:,1], cmap='Reds')
plt.xlabel('Feature1')
plt.ylabel('Feature2')
plt.colorbar()
```

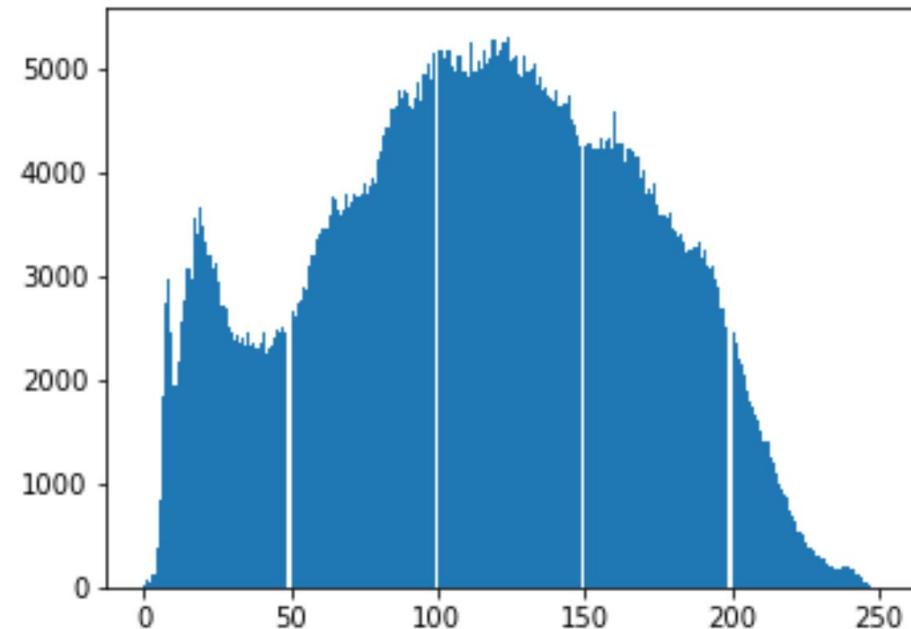
```
<matplotlib.colorbar.Colorbar at 0x1f86b498130>
```



Les histogrammes pour analyser une image

Les histogrammes peuvent aussi être utilisé pour procéder à l'analyse d'une image

```
from scipy import misc
face = misc.face(gray=True)
plt.figure(figsize=(12, 4))
plt.subplot(1,2,1)
plt.imshow(face, cmap='gray')
plt.subplot(1,2,2)
plt.hist(face.ravel(), bins=255)
plt.show()
```



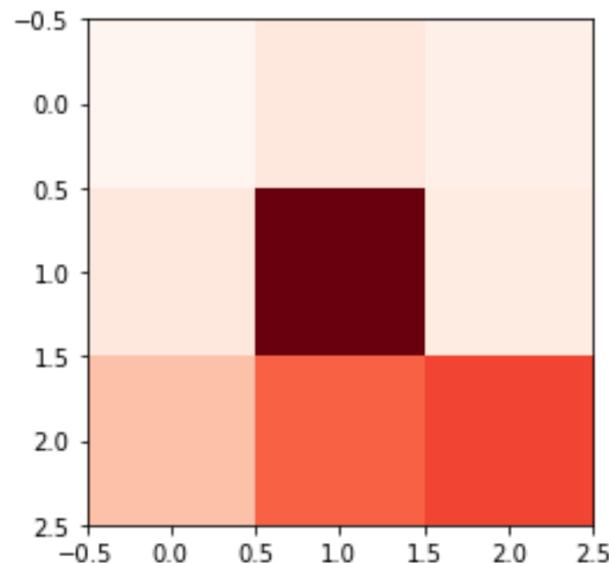
imshow

imshow peut afficher n'importe quelle matrice numpy

```
import numpy as np
A = np.array([[0, 12, 5], [12, 150, 8], [34, 78, 90]])
print(A)
```

```
[[ 0  12   5]
 [ 12 150   8]
 [ 34  78  90]]
```

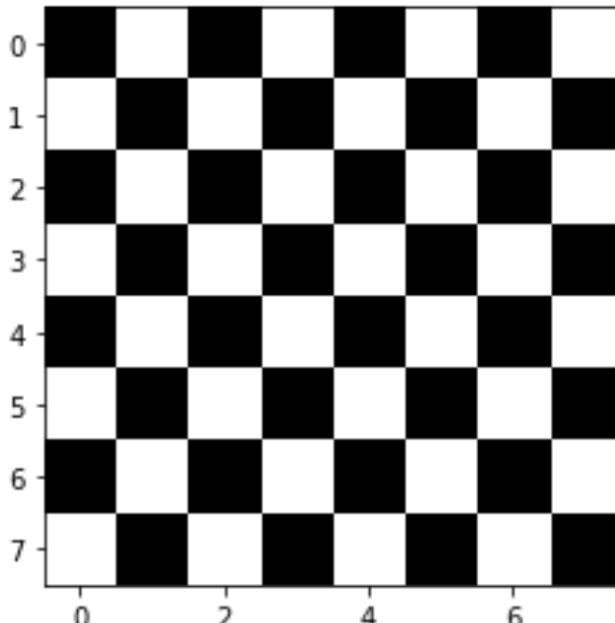
```
import matplotlib.pyplot as plt
plt.imshow(A, cmap='Reds')
plt.show()
```



```
a=np.ones((8,8))
a[::2,::2]=0
a[1::2,1::2]=0
print(a)
```

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

```
plt.imshow(a, cmap='gray')
plt.show()
```



Les graphes 3D

Quand on a besoin de voir la variation de plusieurs variables en même temps et sur un seul graphe on utilise les graphes 3D.

On utilise **mpl-toolkits** qui fait partie du package matplotlib

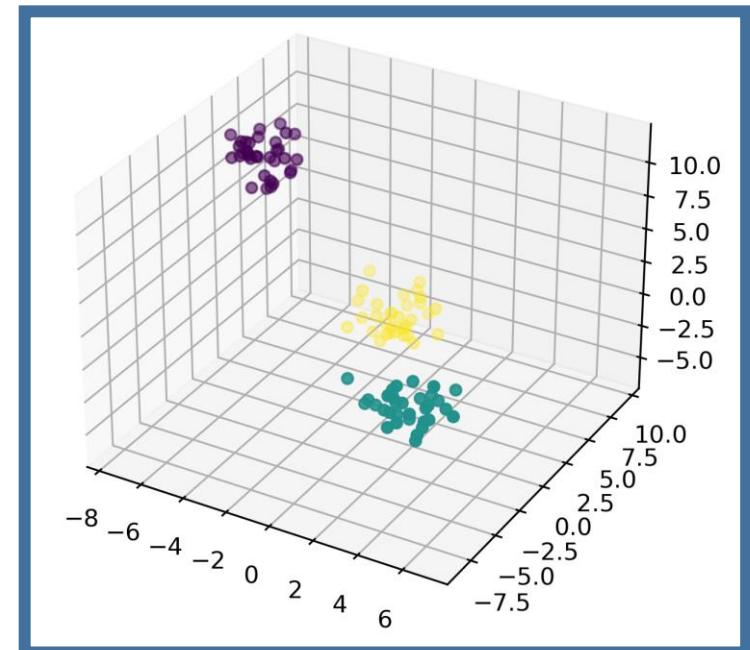
Pour obtenir la figure en 3D sur jupyter il faut écrire **%matplotlib** au début du code !!

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs

%matplotlib

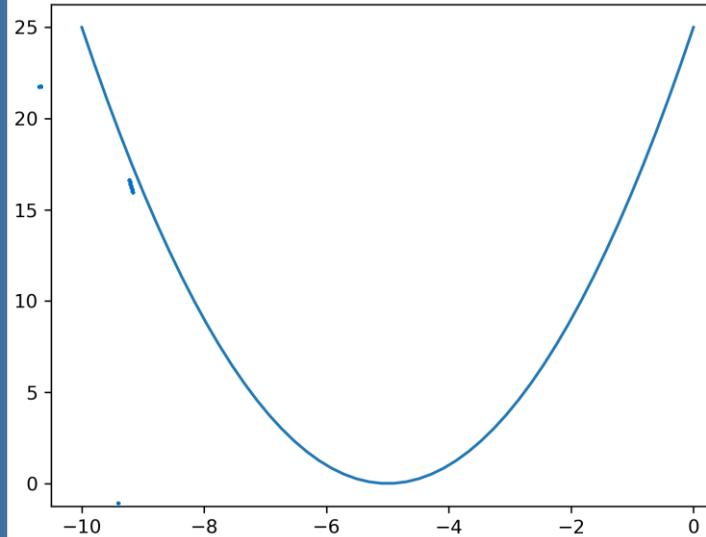
X,y=make_blobs(n_samples=100, n_features=3)
y=y.reshape((len(y),1))

ax = plt.axes(projection='3d')
ax.scatter(X[:, 0], X[:, 1], X[:,2], c=y)
plt.show()
```

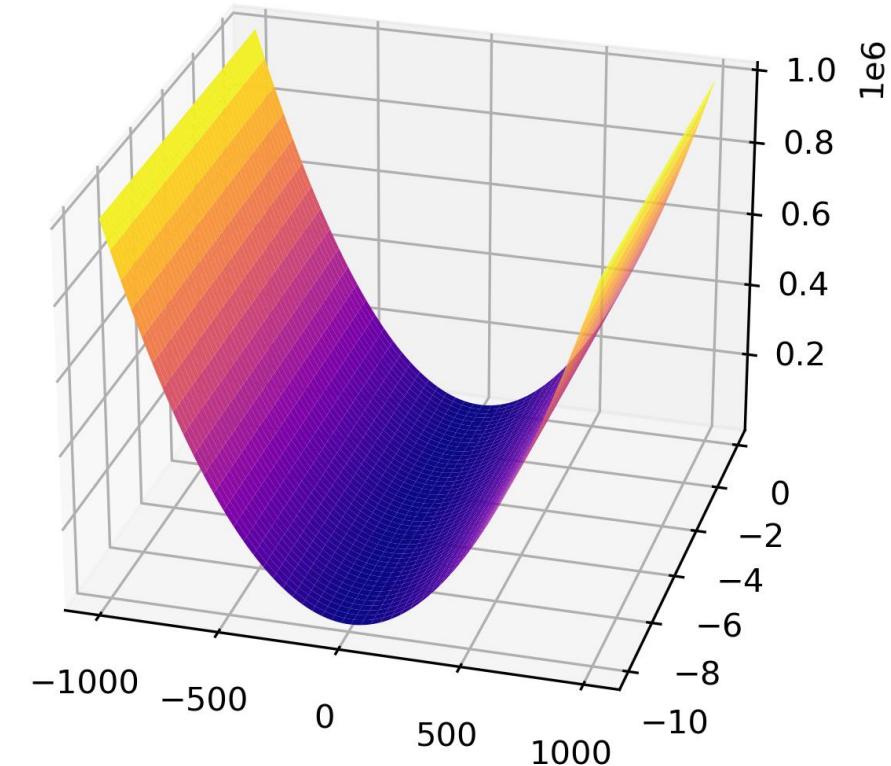


Les graphes 3D: autre exemple

```
def f(X):  
    return (X+5)**2  
X = np.linspace(-10, 0, 50)  
Y=f(X)  
plt.plot(X,Y)
```



```
def f(X,Y):  
    return X**2+Y**2  
X = np.linspace(-1000, 1000, 100)  
Y = np.linspace(-10, 0, 50)  
X, Y = np.meshgrid(X, Y)  
Z = f(X, Y)  
ax = plt.axes(projection='3d')  
ax.plot_surface(X, Y, Z, cmap='plasma')  
plt.show()
```





Manipulation des données avec Pandas

Charger les données dans un DataFrame Pandas

Mise en place de l'environnement de travail

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

```
pd.read_csv(nom_du_fichier.csv)  
  
pd.read_excel(nom_du_fichier.xls)  
  
pd.read_json(nom_du_fichier.json)  
  
pd.read_html()  
  
pd.read_sql()  
Etc...
```

Il est possible de charger les données à partir d'un fichier .txt

```
pd.read_csv(nom_du_fichier.txt",delimiter="\t")
```

Application sur le dataset : Commande de base

`data.shape`



Indique la taille du Dataframe (comme dans numpy)

`data.columns`



Donne les différentes colonne du Dataframe

`data.head()`



Affiche les premières ligne du DataFrame (tail pour la fin)
Peut prendre un argument

`data.describe()`



Affiche les statistiques de base du Datarame
(Moyenne écart type, min, max....)

Application sur le dataset : Commande de base

```
import pandas as pd  
data=pd.read_csv('athlete_events.csv')  
  
data.shape  
  
(271116, 15)
```

Lien de téléchargement du fichier:

<https://www.kaggle.com/datasets/heesoo37/1%20years-of-olympic-history-athletes-and-results>

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 271116 entries, 0 to 271115  
Data columns (total 15 columns):  
 #   Column    Non-Null Count  Dtype     
---    
 0   ID         271116 non-null  int64    
 1   Name        271116 non-null  object    
 2   Sex         271116 non-null  object    
 3   Age         261642 non-null  float64   
 4   Height      210945 non-null  float64   
 5   Weight      208241 non-null  float64   
 6   Team        271116 non-null  object    
 7   NOC         271116 non-null  object    
 8   Games        271116 non-null  object    
 9   Year         271116 non-null  int64    
 10  Season       271116 non-null  object    
 11  City          271116 non-null  object    
 12  Sport         271116 non-null  object    
 13  Event         271116 non-null  object    
 14  Medal         39783 non-null  object    
 dtypes: float64(3), int64(2), object(10)  
memory usage: 31.0+ MB
```

Application sur le dataset : Commande de base

```
data.columns
```

```
Index(['ID', 'Name', 'Sex', 'Age', 'Height', 'Weight', 'Team', 'NOC', 'Games',
       'Year', 'Season', 'City', 'Sport', 'Event', 'Medal'],
      dtype='object')
```

```
data.head()
```

	ID	Name	Sex	Age	Height	Weight	Team	NOC	Games	Year	Season	City	Sport	Event	Medal
0	1	A Dijiang	M	24.0	180.0	80.0	China	CHN	1992 Summer	1992	Summer	Barcelona	Basketball	Basketball Men's Basketball	NaN
1	2	A Lamusi	M	23.0	170.0	60.0	China	CHN	2012 Summer	2012	Summer	London	Judo	Judo Men's Extra-Lightweight	NaN
2	3	Gunnar Nielsen Aaby	M	24.0	NaN	NaN	Denmark	DEN	1920 Summer	1920	Summer	Antwerpen	Football	Football Men's Football	NaN
3	4	Edgar Lindenau Aabye	M	34.0	NaN	NaN	Denmark/Sweden	DEN	1900 Summer	1900	Summer	Paris	Tug-Of-War	Tug-Of-War Men's Tug-Of-War	Gold
4	5	Christine Jacoba Aafink	F	21.0	185.0	82.0	Netherlands	NED	1988 Winter	1988	Winter	Calgary	Speed Skating	Speed Skating Women's 500 metres	NaN

Application sur le dataset : Commande de base

```
data.describe()
```

	ID	Age	Height	Weight	Year
count	271116.000000	261642.000000	210945.000000	208241.000000	271116.000000
mean	68248.954396	25.556898	175.338970	70.702393	1978.378480
std	39022.286345	6.393561	10.518462	14.348020	29.877632
min	1.000000	10.000000	127.000000	25.000000	1896.000000
25%	34643.000000	21.000000	168.000000	60.000000	1960.000000
50%	68205.000000	24.000000	175.000000	70.000000	1988.000000
75%	102097.250000	28.000000	183.000000	79.000000	2002.000000
max	135571.000000	97.000000	226.000000	214.000000	2016.000000

```
data.shape
```

(271116, 15)

Application sur le dataset : Commande de base

```
data.drop(['colonne','colonne', ...])
```

Supprimer des colonnes

```
data.dropna(axis=0)
```

Supprimer les lignes qui contiennent des valeurs manquantes

```
data['colonne'].value_counts()
```

Compter les répétitions

```
data.groupby(['colonne'])
```

Analyse par groupe

Application sur le dataset : Commande de base

```
data=data.drop(['Height','Weight','NOC','Games','Year','Season','City','Event','Medal'],axis=1)
```

```
data
```

	ID	Name	Sex	Age	Team	Sport
0	1	A Dijiang	M	24.0	China	Basketball
1	2	A Lamusi	M	23.0	China	Judo
2	3	Gunnar Nielsen Aaby	M	24.0	Denmark	Football
3	4	Edgar Lindenau Aabye	M	34.0	Denmark/Sweden	Tug-Of-War
4	5	Christine Jacoba Aaftink	F	21.0	Netherlands	Speed Skating
...
271111	135569	Andrzej ya	M	29.0	Poland-1	Luge
271112	135570	Piotr ya	M	27.0	Poland	Ski Jumping
271113	135570	Piotr ya	M	27.0	Poland	Ski Jumping
271114	135571	Tomasz Ireneusz ya	M	30.0	Poland	Bobsleigh
271115	135571	Tomasz Ireneusz ya	M	34.0	Poland	Bobsleigh

271116 rows × 6 columns

Application sur le dataset : Nettoyage des données

```
data.shape
```

```
(271116, 15)
```

```
data.describe()
```

	ID	Age
count	271116.000000	261642.000000
mean	68248.954396	25.556898
std	39022.286345	6.393561
min	1.000000	10.000000
25%	34643.000000	21.000000
50%	68205.000000	24.000000
75%	102097.250000	28.000000
max	135571.000000	97.000000

Dans le cas où des données sont manquantes, on peut procéder de deux manières

- Option 1: Compléter les données par une valeur par défaut

```
data.fillna(X)
```

Exemple:

```
data.fillna(data['Age'].mean)
```

Attention ! On modifie la réalité, les données vont être corrompus

- Option 2: Eliminer les lignes où les données sont manquantes

Nous pouvons supprimer les colonnes où il manque des données, les lignes **dropna** supprime les lignes par défaut

```
data = data.dropna(axis=0)
```

Certes nous avons moins de données mais on altère pas la vérité...

Application sur le dataset : Commande de base

```
data.describe()
```

	ID	Age
count	271116.000000	261642.000000
mean	68248.954396	25.556898
std	39022.286345	6.393561
min	1.000000	10.000000
25%	34643.000000	21.000000
50%	68205.000000	24.000000
75%	102097.250000	28.000000
max	135571.000000	97.000000

```
data.shape
```

(271116, 15)

data.dropna(axis=0)

Supprimer les lignes
qui contiennent des
valeurs manquantes

```
data=data.dropna(axis=0)
```

```
data.describe()
```

	ID	Age
count	261642.000000	261642.000000
mean	68291.263960	25.556898
std	38997.527135	6.393561
min	1.000000	10.000000
25%	34755.250000	21.000000
50%	68198.000000	24.000000
75%	102108.750000	28.000000
max	135571.000000	97.000000

```
data[ 'Age' ].value_counts()
```

Application sur le dataset : Commande de base

```
data=data.iloc[0:500,:]
```

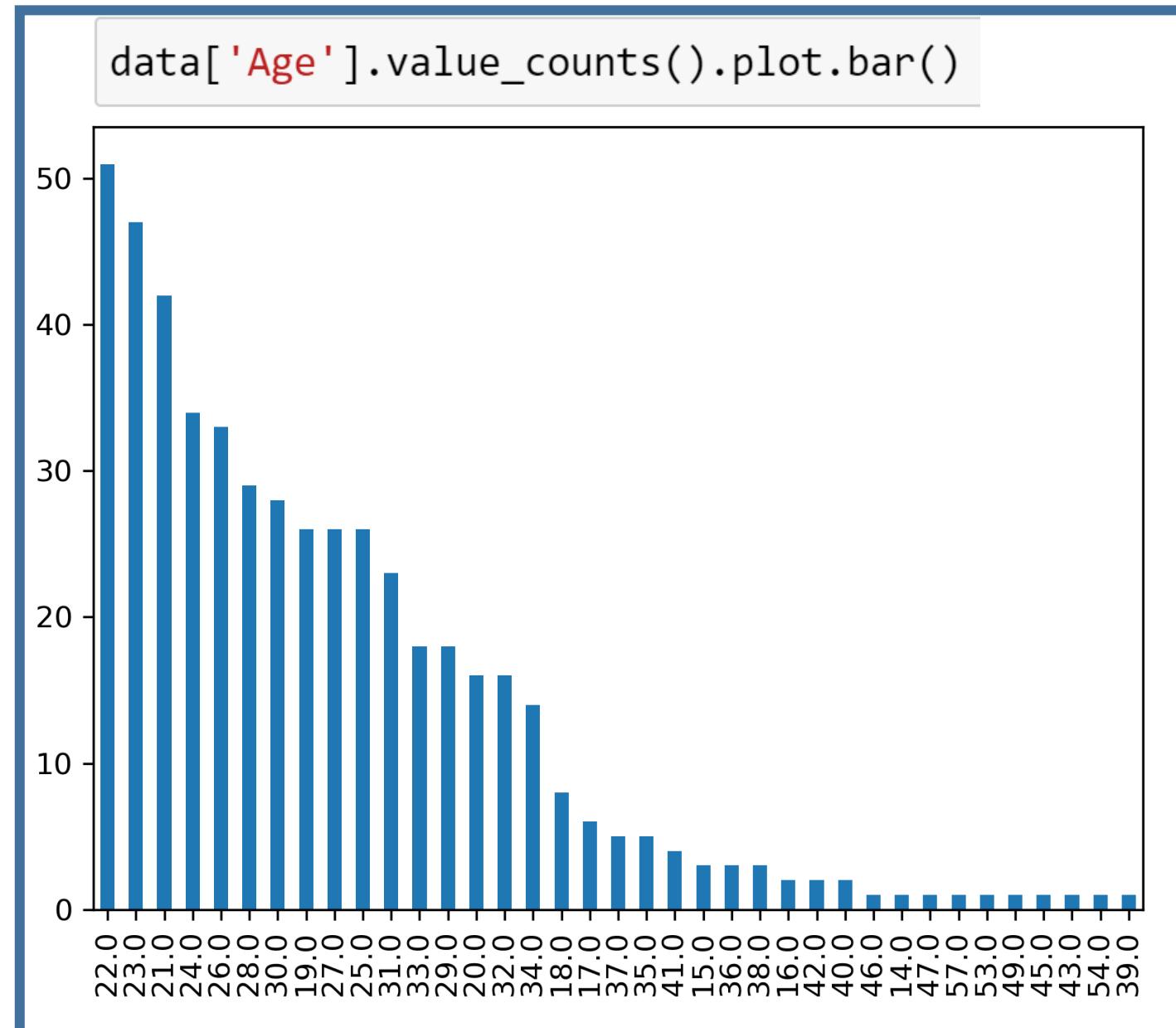
22.0	51
23.0	47
21.0	42
24.0	34
26.0	33
28.0	29
30.0	28
19.0	26
27.0	26
25.0	26
31.0	23
33.0	18
29.0	18
20.0	16
32.0	16
34.0	14
18.0	8
17.0	6
37.0	5
35.0	5
41.0	4
15.0	3
36.0	3
38.0	3
16.0	2
42.0	2
40.0	2
46.0	1
14.0	1
47.0	1
57.0	1
53.0	1
49.0	1
45.0	1
43.0	1
54.0	1
39.0	1

Application sur le dataset : Commande de base

Panda utilise Matplotlib.pyplot :

```
df.plot()  
df.plot.bar()  
df.hist(bins=...)  
df.scatter(x=...,y=...)
```

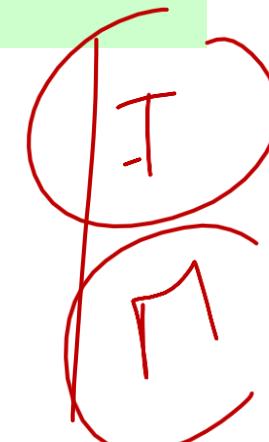
Exemple :



Application sur le dataset : Manipulation des données

`df.groupby()`

Fonction qui permet d'analyser les données par groupe



Exemple :

Déterminer le nombre des féminins et des masculins

`data.groupby(['Sex']).count()`

1	<code>data.groupby(['Sex']).count()</code>			
<hr/>				
<hr/>				
ID	Name	Age	Team	Sport
<hr/>		Sex		
F	70	70	70	70
M	430	430	430	430

Déterminer la conso moyenne par type de boite et de cylindrée

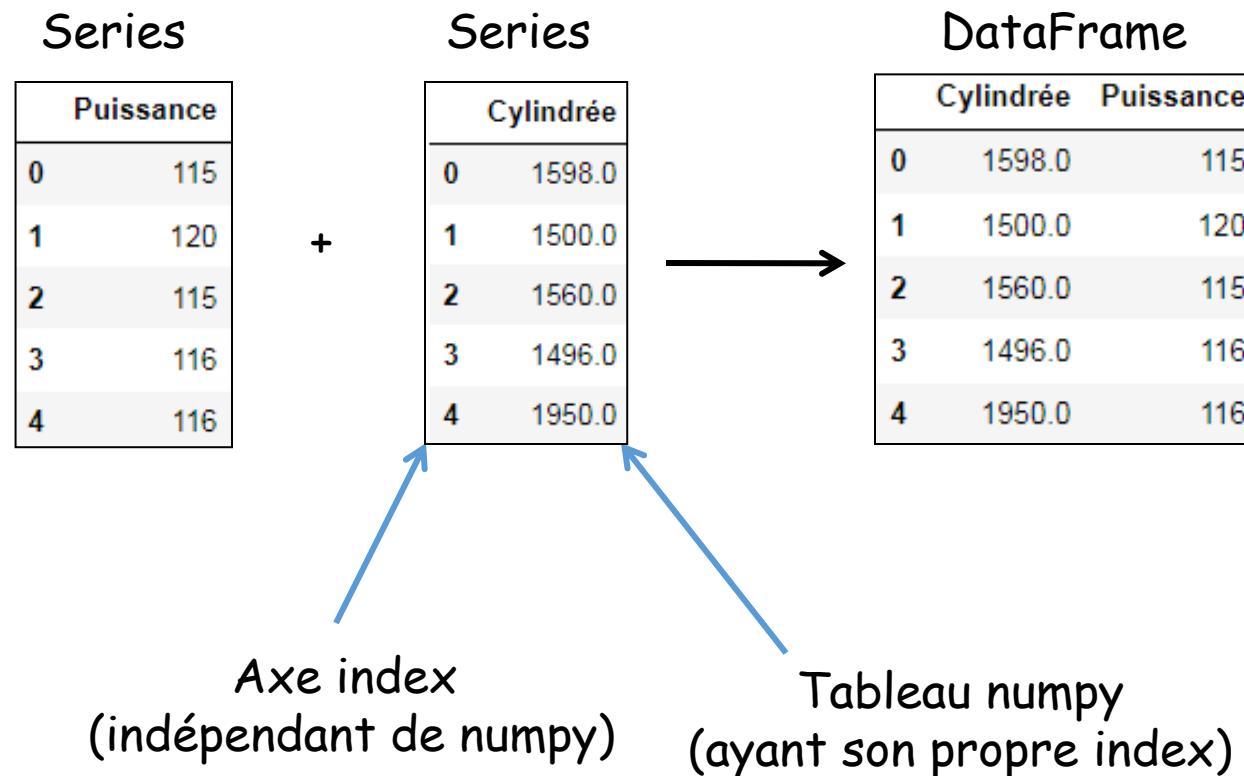
`data.groupby(['Sex']).mean()`

1	<code>data.groupby(['Sex']).mean()</code>
<hr/>	
<hr/>	
ID	Age
<hr/>	
Sex	
F	138.342857 24.185714
M	137.709302 26.467442

Pandas: Qu'est ce qu'une Pandas Series ?

Dans Pandas, il existe deux structures de données:

- Les Séries: qui sont un tableau numpy 1D+axe d'index
- Les Dataframes: qui sont un assemblage de séries



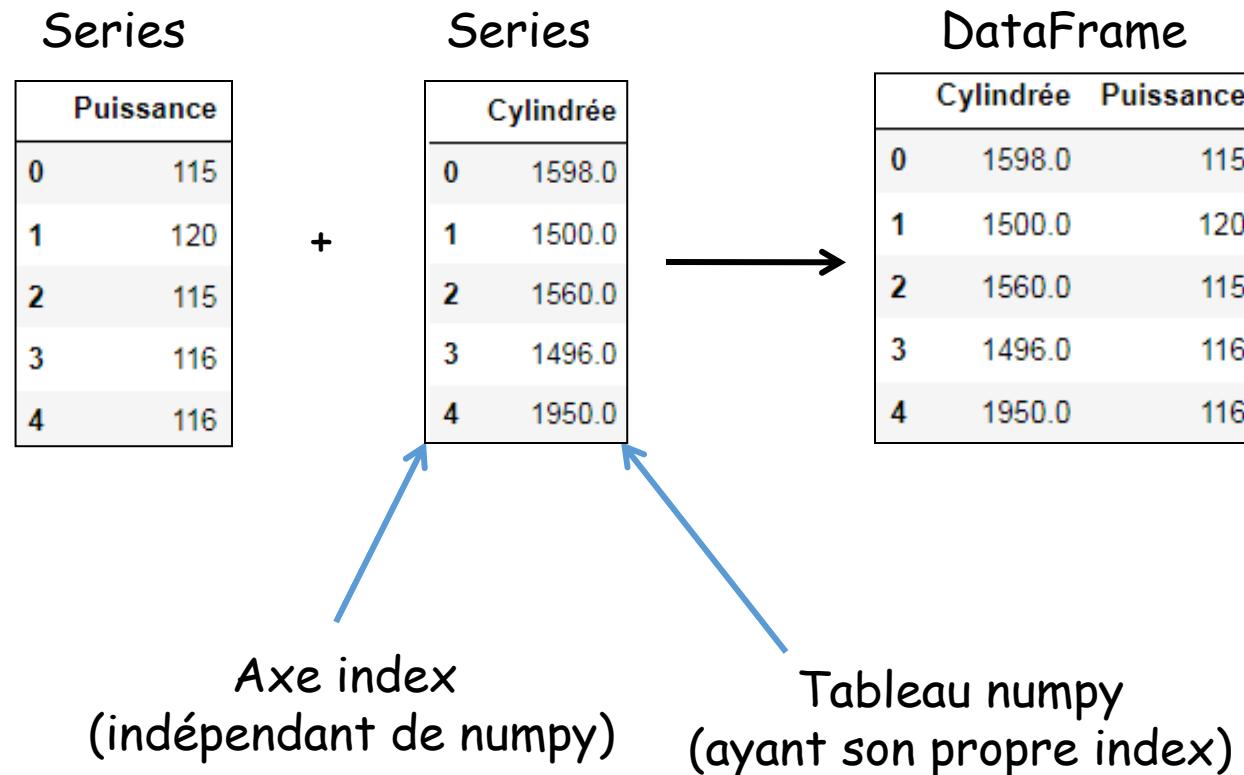
Note1: on peut choisir index autre que 0,1,2,3....

Note2: on ne peut rassembler des séries que quand elle partage le même index

Pandas: Qu'est ce qu'une Pandas Series ?

Dans Pandas, il existe deux structures de données:

- Les Séries: qui sont un tableau numpy 1D+axe d'index
- Les Dataframes: qui sont un assemblage de séries



Note1: on peut choisir index autre que 0,1,2,3....

Note2: on ne peut rassembler des séries que quand elle partage le même index

Pandas: Qu'est ce qu'un Dataframe ?

Pour résumé:

Un Dataframe c'est comme un dictionnaire de série

Pour le dictionnaire:

Dict['key'] = valeur

Pour le Dataframe:

Df['column'] = serie

	Marque	Cylindrée	Puissance
0	Golf	1598.0	115
1	Focus	1500.0	120
2	308	1560.0	115
3	Serie 1	1496.0	116
4	Class A	1950.0	116

Pandas: changement d'index

Exemple :

```
data['Age']
```



1	data['Age']
0	24.0
1	23.0
2	24.0
3	34.0
4	21.0

```
data=data.set_index('Name')  
data['Age']
```



1	data=set_index('Name')
2	data['Age']
	Name
	A Dijiang
	A Lamusi
	Gunnar Nielsen Aaby
	Edgar Lindenau Aabye
	Christine Jacoba Aafink
	24.0
	23.0
	24.0
	34.0
	21.0

Pandas: Extraction des données d'un Dataframe

Ces propriétés des Dataframes et des séries permet d'appliquer dessus ce qui a été vu sur les tableau numpy

```
data[' Age ']
```

#Obtention d'une serie

```
data[' Age '][0:4]
```

#Indication (indexing)

```
data[' Age '] <40
```

#Indication booléen (boolean indexing)

```
data[data[' Age '] < 40]
```

#Application d'un masque

```
data[[' Age ','Sex']]
```

#Obtention d'un Dataframe

→ Un Dataframe peut être utilisé comme un tableau numpy et pour accéder à une valeur on écrit:

```
data[0,0]
```

→ Envoi un message Error

```
data[:,0:10]
```

Au lieu de cela on utilise les fonctions iloc et loc:

```
data.iloc[0:1,0:3]  
data.loc[0:8,'Age ']
```