



Pour aller plus loin



- PYPI
- autres IDE
- Les packages pour le machine Learning



- **PyPI** « *Python Package Index* » est le dépôt tiers officiel du langage de programmation Python. Son objectif est de doter la communauté des développeurs Python d'un catalogue complet recensant tous les paquets Python libres
- Exemple : installation du package:

pip install numpy

pip install -U scikit-learn

IDE



- IDLE : simple, léger,
- Pour utiliser d'autres modules ou packages, il faut les installer
- Utiliser IDE Pycharm (version Community)

<https://www.jetbrains.com/fr-fr/pycharm/download/#section=windows>

- Pour le machine Learning: **distribution Anaconda**

IDE Pycharm



- IDE très complet et se dédie aux développeurs professionnels.
- bénéficie de toutes les fonctionnalités attendues d'un IDE : complétion du code, saisie automatique, vérification à la volée, gestion de projet, support des principaux frameworks Python (Django, Flask, etc.), développement et debug distant, gestion de version, support des bases de données, etc.
JetBrains propose des mises à jour régulières.
- PyCharm est disponible en deux éditions : Professional Edition et Community Edition. La version communautaire est gratuite, mais toutes les fonctionnalités ne sont pas présentes.

La distribution Anaconda



- **Anaconda** est une distribution libre et open source du langage Python appliqué au développement d'applications dédiées à la science des données et au machine Learning
- La distribution Anaconda est utilisée par plus de 6 millions d'utilisateurs. La version d'installation comprend plus de 250 paquets populaires en science des données adaptés pour Windows, Linux et MacOS. Plus de 7 500 paquets open-source supplémentaires peuvent être installés à partir de [PyPI](#)


Anaconda


 ANACONDA NAVIGATOR



[Sign in to Anaconda.org](#)

 Home

 Environments

 Learning

 Community

[Documentation](#)

[Developer Blog](#)



Applications on

base (root)

Channels

Refresh



JupyterLab

1.1.4

An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Launch



Notebook

6.0.1

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Launch



Spyder

3.3.6

Scientific PYTHON Development Environment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

Launch



VS Code

1.62.3

Streamlined code editor with support for development operations like debugging, task running and version control.

Launch



Glueviz

0.15.2

Multidimensional data visualization across files. Explore relationships within and among related datasets.



Orange 3

3.23.0

Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows



RStudio

1.1.456

A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.



packages Python utiles pour les data scientists

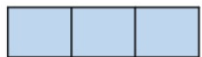
- **NumPy**, manipulation de matrices avec Python
- **SciPy**, référence pour le calcul scientifique (les opérations d'interpolation, d'optimisation, de traitement du signal, et de traitement d'image...)
- **Pandas**: apporte à Python la possibilité de manipuler de grands volumes de données structurées de manière simple et intuitive.
- **Scikit-learn**, package pour le machine learning: Il comprend de nombreuses méthodes pour classifier, faire de la validation croisée, du clustering, ou encore de la sélection de modèle.
- **matplotlib**, package pour la visualisation des données avec de nombreuses possibilités de personnalisations.
- **Bokeh**, de beaux graphiques pour le web
- **seaborn**, matplotlib amélioré
- **Keras**, le deep learning

Numpy

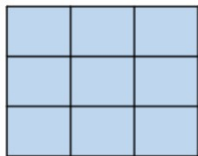


- **Numpy** est **LE package** qui permet de créer des **matrices** et de faire des mathématiques de manière **ultra-performante** (Numpy étant développé en C): les matrices sont la base de tout.
- Au cœur de Numpy se trouve un objet **très puissant**: Le tableau à N-Dimensions (**ndarray**).
- **ndarray** est un objet puissant, il permet d'effectuer beaucoup d'actions mathématiques **avancées**, il permet de contenir une **infinité de données**, et est **très rapide d'exécution**.

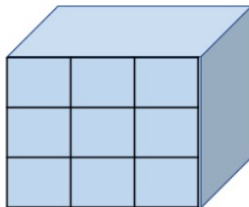
ndarray



1D array



2D array



3D array

.....



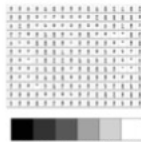
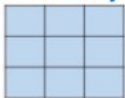
ND array

NUMPY

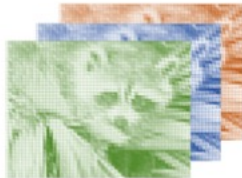
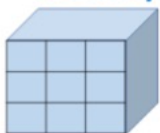
Une image, c'est un tableau de pixels

<https://machinelearningia.com>

2D array




3D array



En ingénierie, machine learning et Data Science, on travaille le plus souvent avec des tableaux à 2 dimensions (dataset, image, matrice). Parfois à 3 dimensions (pour une image en couleur, qui contient les couches Rouge, Vert, Bleu)

Pourquoi ndarray et non pas les listes ?

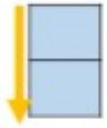


liste = 
Index 0 1 2 3

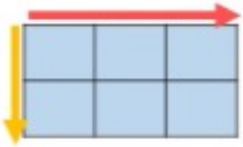
ndarray = 
Index 0 1 2 3

ndarray (beaucoup) plus **rapide**,
meilleur aux **calculs scientifiques**

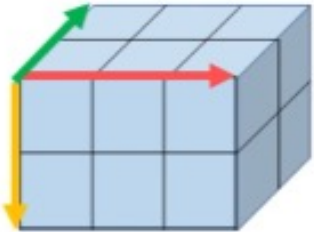
Les attributs de l'objet ndarray: ndim, shape



ndim = 1
shape = (2,)



ndim = 2
shape = (2, 3)



ndim = 3
shape = (2, 3, 2)

Créer un tableau



```
Entrée [2]: import numpy as np
            A=np.array([1,2,3])
            print(type(A))
            print(A)

            <class 'numpy.ndarray'>
            [1 2 3]
```

Exemple



```
Entrée [9]: A = np.zeros((2, 3)) # création d'un tableau de shape (2, 3)
print("A = ",A)

print(A.size) # le nombre d'éléments dans le tableau A
print(A.shape) # les dimensions du tableau A (sous forme de Tuple)

print(type(A.shape)) # voici la preuve que la shape est un tuple

print(A.shape[0]) # le nombre d'éléments dans la première dimension de A
```

```
A =  [[0. 0. 0.]
      [0. 0. 0.]]
6
(2, 3)
<class 'tuple'>
2
```

Exemple



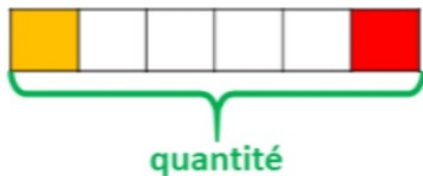
```
Entrée [5]: A = np.zeros((2, 3)) # tableau de 0 aux dimensions 2x3
B = np.ones((2, 3)) # tableau de 1 aux dimensions 2x3
C = np.random.randn(2, 3) # tableau aléatoire (distribution normale) aux dimensions 2x3
D = np.random.rand(2, 3) # tableau aléatoire (distribution uniforme)
E = np.random.randint(0, 10, [2, 3]) # tableau d'entiers aléatoires de 0 a 10 et de dimension 2x3
print("A = ",A)
print("B = ",B)
print("C = ",C)
print("D = ",D)
print("E = ",E)

A =  [[0. 0. 0.]
      [0. 0. 0.]]
B =  [[1. 1. 1.]
      [1. 1. 1.]]
C =  [[ 0.5073393 -1.55278352 -0.04400401]
      [-0.10857495 -1.17464062 -0.40023101]]
D =  [[0.23240925 0.02432664 0.94510161]
      [0.28585648 0.2266489  0.2268275 ]]
E =  [[4 8 8]
      [2 2 2]]
```

Constructeur :linspace



`np.linspace(début, fin, quantité)`



```
Entrée [17]: np.linspace(0,10,20)
```

```
Out[17]: array([ 0.          ,  0.52631579,  1.05263158,  1.57894737,  2.10526316,  
                2.63157895,  3.15789474,  3.68421053,  4.21052632,  4.73684211,  
                5.26315789,  5.78947368,  6.31578947,  6.84210526,  7.36842105,  
                7.89473684,  8.42105263,  8.94736842,  9.47368421, 10.          ])
```


Constructeur :arange



`np.arange(début, fin, pas)`



```
Entrée [18]: np.arange(0,10,0.5)
```

```
Out[18]: array([0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5, 5. , 5.5, 6. ,  
               6.5, 7. , 7.5, 8. , 8.5, 9. , 9.5])
```

Slicing

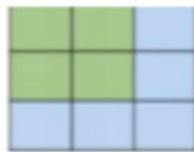


- Dans le cas du slicing, on choisit plutôt d'accéder à plusieurs éléments d'un même axe du tableau. On parlera souvent de **sous-ensemble** (*subset*). Il faudra donc indiquer un **index de début** et un **index de fin** pour chaque dimension de notre tableau
- *Note: L'index de fin n'est jamais compris dans l'opération de Slicing.*

A[0, 1]



A[0:2, 0:2]



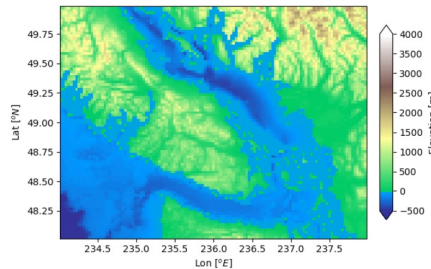
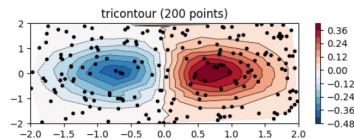
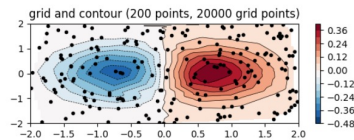
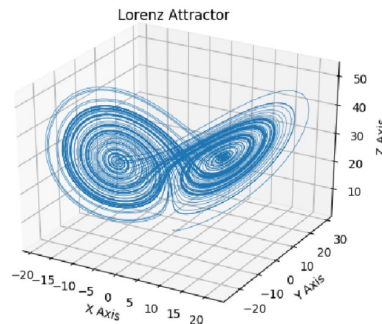
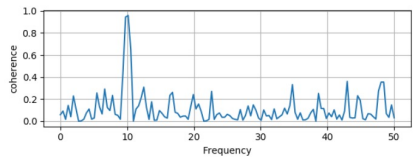
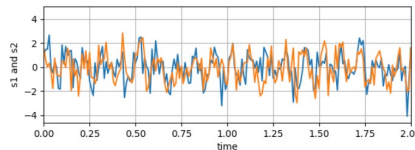


Matplotlib

Matplotlib



- **Matplotlib** est un package (c'est-à-dire un regroupement de plusieurs modules) servant à la création de graphiques Python. Voici quelques exemples de ce que vous pouvez créer avec Matplotlib



Le module pyplot

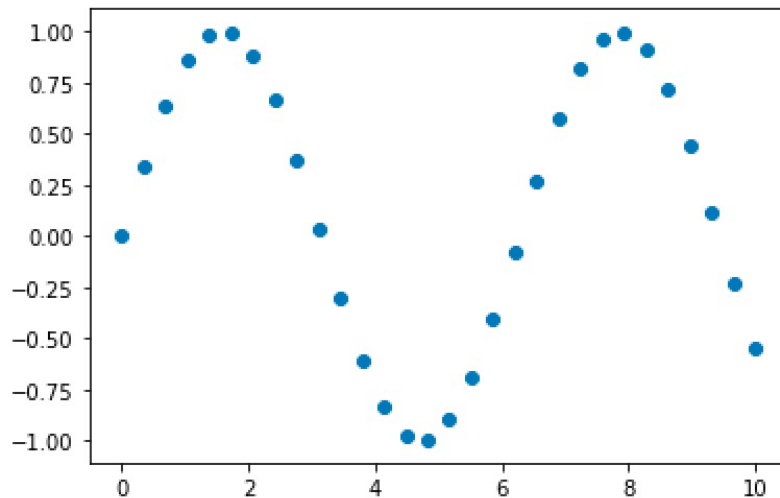


- il suffit d'importer un module appelé **pyplot** et d'utiliser une fonction de graphique parmi les suivantes :
- **scatter(x, y)** : affiche un graphique à points entre x et y
- **plot(x, y)** : affiche un graphique ligne entre x et y
- **hist(x)** : affiche un histogramme de x.

Exemple



```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # création de données
5 x = np.linspace(0, 10, 30)
6 y = np.sin(x)
7
8 plt.scatter(x, y)
9 plt.show() # affiche le graphique (optionnel dans Jupyter)
```



Contrôler le style de vos graphiques Matplotlib



- Pour conférer un style à vos graphiques, il suffit d'ajouter certains **arguments** dans les fonctions `plot()` et `scatter()`. Parmi les arguments les plus utiles, je vous conseille de retenir :
 - **c** : couleur du graphique ('red', 'blue', 'black', etc...)
 - **lw** : épaisseur du trait (pour les graphiques de lignes)
 - **ls** : style de trait (pour les graphiques de lignes)
 - **marker** : style de points (pour les graphiques de points)
 - **alpha** : transparence de la ligne ou des points

Exemple

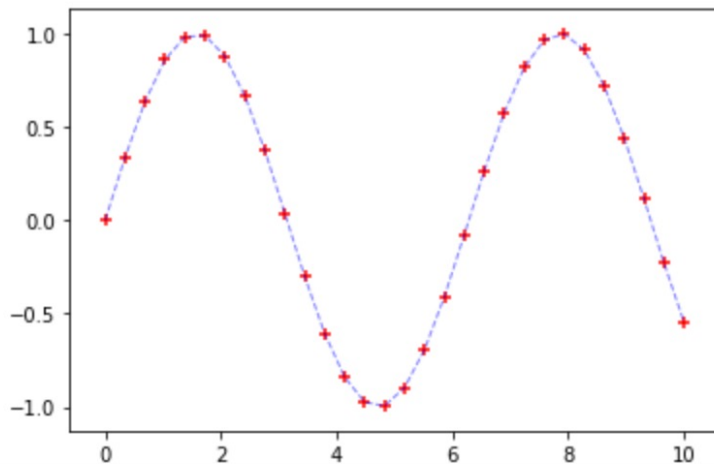


```
Entrée [21]: import numpy as np
import matplotlib.pyplot as plt

# création de données
x = np.linspace(0, 10, 30)
y = np.sin(x)

plt.plot(x, y, c='blue', ls='--', lw=1, alpha=0.5)
plt.scatter(x, y, c='red', marker='+')
plt
```

Out[21]: <module 'matplotlib.pyplot' from '/Users/oumaira/opt/anaconda3/lib/py



Subplot : Diviser votre figure en plusieurs sous-figures

- La fonction **subplot()** permet de diviser une figure en plusieurs parties, créant ainsi une **grille** de sous-figures. Pour cela, il faut faire passer 3 nombres dans cette fonctions. Le nombre de lignes de la grille, le nombre de colonne de la grille, et la sous-figure sur laquelle travailler. On écrit ainsi :
`plt.subplot(#lignes, #colonnes, sous-figure)`

Exemple

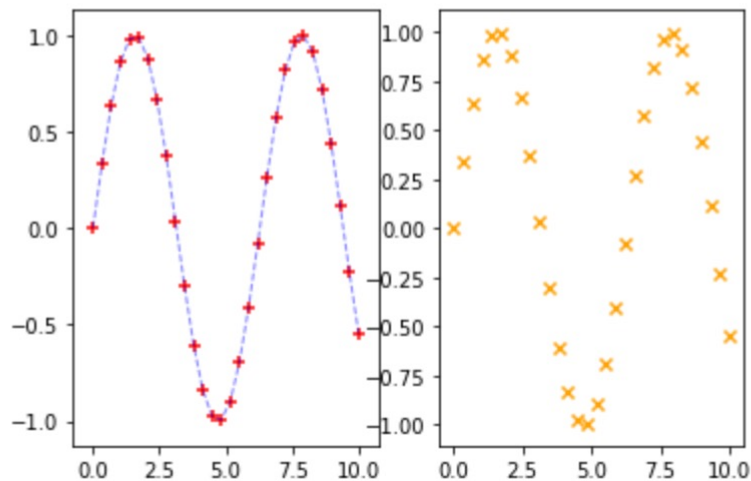


```
Entrée [22]: plt.subplot(1, 2, 1) # 1 ligne, 2 colonnes, sous-figure 1
plt.plot(x, y, c='blue', ls='--', lw=1, alpha=0.5)
plt.scatter(x, y, c='red', marker='+')

plt.subplot(1, 2, 2) # 1 ligne, 2 colonnes, sous-figure 2
plt.scatter(x, y, c='orange', marker='x')

plt
```

Out[22]: <module 'matplotlib.pyplot' from '/Users/oumaira/opt/anaconda3/lib/py



Matplotlib : Titres, Axes, Légendes



- Pour créer des Titres, axes, et légendes dans Matplotlib, il suffit d'utiliser les fonctions suivantes sur chacune de vos figures ou sous-figures:
- `plt.title()` : ajoute un titre à la figure
- `plt.xlabel()` `plt.ylabel()` : ajoute des axes à la figure
- `plt.lenged()` : ajoute une légende (selon le label indiqué dans la fonction `plot()` ou `scatter()`)
-

Exemple



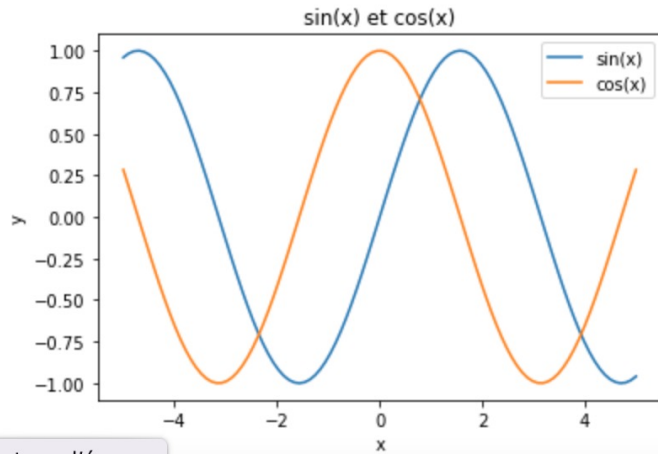
```
Entrée [23]: x = np.linspace(-5, 5, 100)

# création des graphiques
plt.plot(x, np.sin(x), label='sin(x)')
plt.plot(x, np.cos(x), label='cos(x)')

# embellissement de la figure
plt.title('sin(x) et cos(x)')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()

plt
```

Out[23]: <module 'matplotlib.pyplot' from '/Users/oumaira/opt/anaconda3/lib/python3.7/s





[https://www.youtube.com/channel/
UCmpptkXu8ilFe6kfDK5o7VQ](https://www.youtube.com/channel/UCmpptkXu8ilFe6kfDK5o7VQ)