

Formation Machine Learning

Résumé des 30 vidéos
de la série YouTube





Variables
& Fonctions

Structures
de données

Modules

Les Bases de Python pour Data Science

Structures
de contrôle

Built-in Functions

P.O.O.

Seaborn

Analyse de Données et Visualisation

Scipy

Numpy

Pandas

Les modules Scientifiques

Matplotlib

Apprentissage supervisé

Méthodes
Ensemblistes

EDA

Modélisation

Machine Learning et Datasets

Apprentissage non-supervisé

Projet

Pre-processing



Sommaire

LES BASES PYTHON POUR LE MACHINE LEARNING

- 2: [Python Variables et fonctions](#)
- 3: [Python IF/ELSE, WHILE, FOR](#)
- 4: [Python LISTES et TUPLES](#)
- 5: [Python Dictionnaires](#)
- 6: [Python List Dict Compréhension](#)
- 7: [Python Built-in functions](#)
- 8: [Python Modules et Packages](#)
- 9: [Programmation orientée objet](#)

LES MODULES SCIENTIFIQUES

- 10: [Python NUMPY machine Learning](#)
- 11: [Python NUMPY Indexing Slicing Masking](#)
- 12: [Python NUMPY Statistiques et Mathématiques](#)
- 13: [Python NUMPY Broadcasting](#)
- 14: [MATPLOTLIB – Les bases !](#)
- 15: [MATPLOTLIB – Graphiques Importants](#)
- 16: [Python SCIPY Tutorial – Optimize, Fourier, NdImage](#)

ANALYSE DE DONNEES ET VISUALISATION

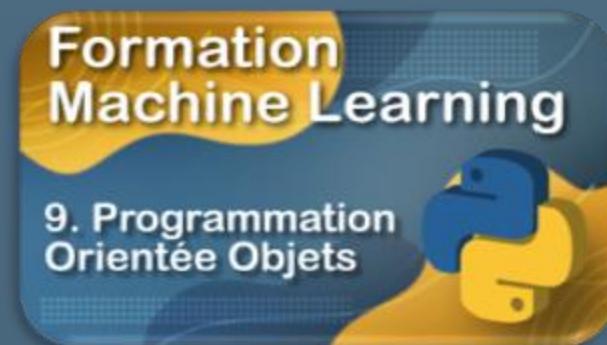
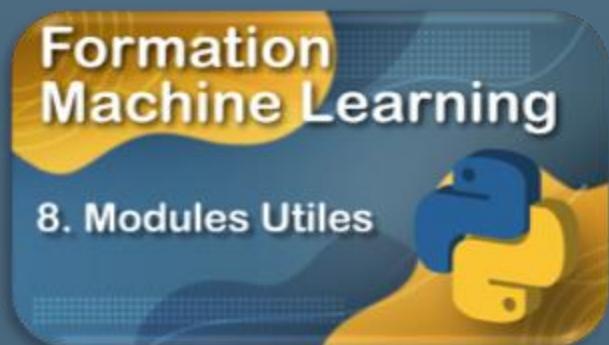
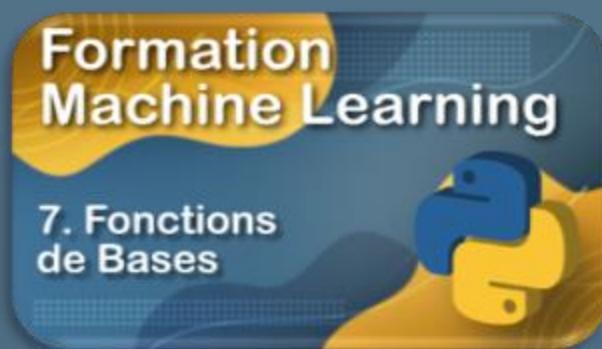
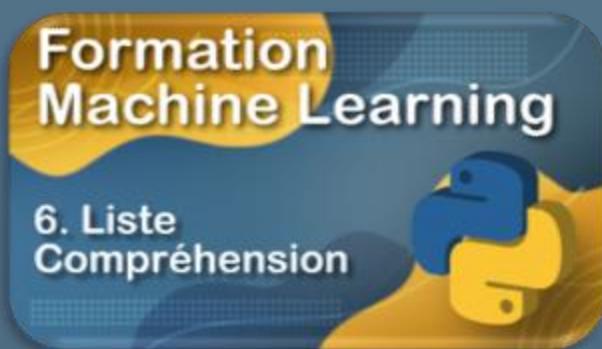
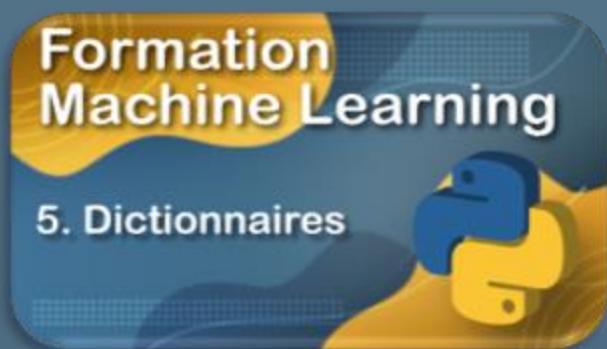
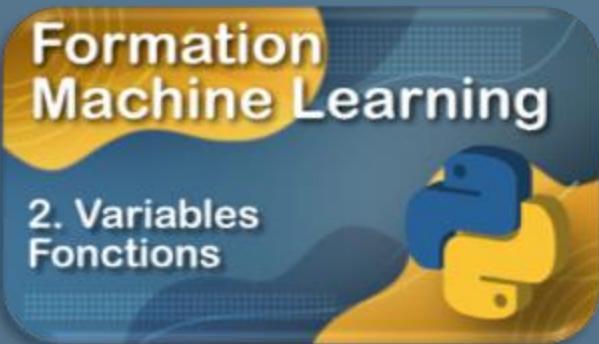
- 17: [PANDAS Python – Introduction + Analyse du Titanic](#)
- 18: [PANDAS Python – Time Series](#)
- 19: [SEABORN Les plus beaux graphiques en 1 Ligne de Code](#)

MACHINE LEARNING ET DATASETS

- 20: [SKLEARN : Supervised Learning](#)
- 21: [SKLEARN – Model Selection](#)
 - 21.1: [Cross-Validation](#)
 - 21.2: [Métrique de Regression](#)
- 22: [SKLEARN Pre-processing](#)
 - 22.1: [SKLEARN Pipeline avancée](#)
 - 22.2: [SKLEARN Nettoyage de données](#)
- 23: [SKLEARN Feature Selection](#)
- 24: [Python Apprentissage non-supervisé](#)
- 25: [Ensemble Learning](#)

PROJET

- 26: [Data Science et Démarche de Travail](#)
- 27: [Exploratory Data Analysis – Corrigé](#)
- 28: [Python Pré-Traitement de données](#)
- 29: [Modèle de Machine Learning](#)
- 30: [Conclusion](#)



Variables et Fonctions

Une variable est une zone de la mémoire de l'ordinateur caractérisée par un **nom**, où l'on stocke de l'information. La **déclaration** et l'**initialisation** se font en même temps.

Une fonction se définit avec **def** suivi de son **nom**, d'une liste de **paramètres**, du caractère «**:**» et de son **corps**.

Dans son corps, il est possible de faire des opérations arithmétiques, de comparaison, de logiques, à l'aide de 4 types de variable.

Déclaration d'une fonction :

```
def nom_de_la_fonction(arg1, arg2):  
    resultat = arg1 + arg2  
    return resultat
```

Appel de la fonction :

```
nom_de_la_fonction(3, 2)
```

Type :

```
x = 3 # type int  
y = 2.5 # type float  
prenom = 'Pierre' # type string  
z = True # type Bool
```

Comparaison :

Egalité :	x == y
Inégalité :	x != y
Inférieur ou égale à :	x <= y
Supérieur ou égale à :	x >= y

Fonction :

print(...) : affiche les informations dans le terminal
input(...) : reçoit les informations de l'utilisateur
int(...) : convertie une valeur en entier
float(...) : convertie une valeur en nombre décimal

Opérations entre x et y :

x + y	x - y
x / y	x // y
x * y	x ** y
x % y	

Opérateurs logiques :

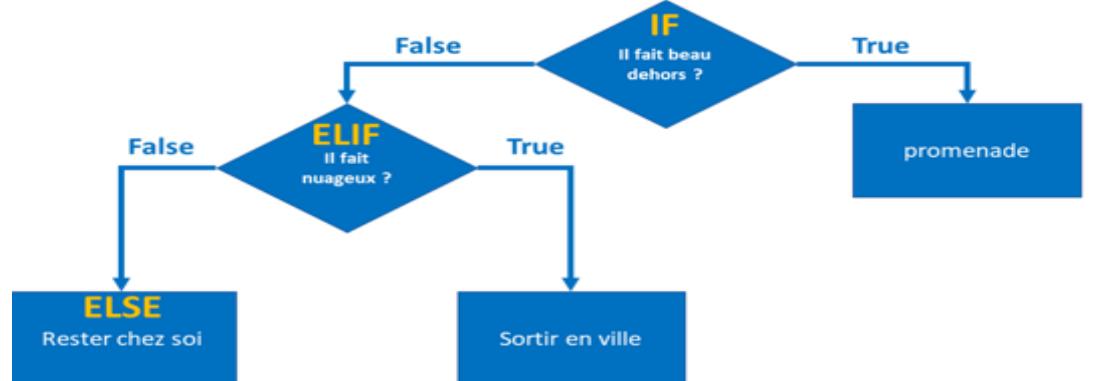
ET :	x and y
OU :	x or y
OU exclu :	x ^ y
NON :	not x

Structure de Contrôle: If, Else, For, While

Il n'y a que 3 structures algorithmiques à connaître en programmation : **condition**, **boucle for**, **boucle while**.

Condition :

Permet d'examiner l'état actuel d'un programme et de réagir de manière appropriée à cet état



```
def test_signe(valeur):  
    if valeur < 0:  
        print('negatif')  
    elif valeur == 0:  
        print('nul')  
    else:  
        print('positif')
```

Boucle While :

S'exécute tant que certaines conditions restent vraies

While (tant que) $x < 10$ TRUE



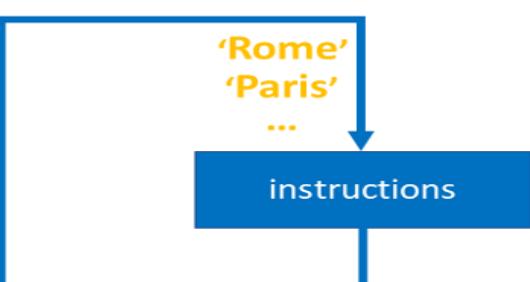
Incrémenté à chaque passage :

```
x = 0  
while x < 10:  
    print(x)  
    x += 1
```

Boucle for :

S'exécute un certain nombre de fois

For i in Liste:



Compte de 0 à 9
for i in range (0, 10):
 print(i)

Compte de 0 à 9
range(0, 10)

Compte à rebours 2 par 2
range(10, 0, -2)

Listes et Tuples

<https://machinelearnia.com>

Une **liste** contient une série d'items dans un ordre particulier. On commence par définir la liste. On peut accéder à un élément de celle-ci, ajouter, enlever, interférer des éléments.
Une tuple ressemble à une liste mais les items contenus ne peuvent pas être changés.

Liste =	'Paris'	'Berlin'	'Londres'	'Bruxelles'
(index)	0	1	2	3

Indexing: Pour accéder à un élément.

Liste[#index] = élément

Slicing: Pour accéder à une mini-séquence

Liste[#début : #fin] = fourchette

If/Else



Boucle For

For i in Liste:
-> 'Paris' - 'Berlin' - 'Londres' - 'Bruxelles'

For index, valeur in enumerate(Liste):
-> 0 'Paris' - 1 'Berlin' - 2 'Londres' - 3 'Bruxelles'

For A, B in zip(Liste_A, Liste_B):
-> A0 B0 - A1 B1 - A2 B3 - ...

Indexing :

liste # toute la liste

liste[0] # le premier indice

liste[1] # le deuxième indice

liste[-1] # le dernier indice

Slicing :

liste # toute la liste

liste[0:3] # indice 0 à 2

liste[1:3] # indice 1 à 2

liste[::-1] # inverse la séquence

Méthodes

Liste.sort(reverse=False)

'Berlin'	'Bruxelles'	'Londres'	'Paris'
0	1	2	3

= 2

Liste.count(truc)

'Paris'	'Berlin'	'Paris'	'Bruxelles'
0	1	2	3

= 4

len(liste)

'Paris'	'Berlin'	'Londres'	'Bruxelles'
0	1	2	3

'Madrid'

Liste.append(truc)

'Paris'	'Berlin'	'Londres'	'Bruxelles'
0	1	2	3

'Madrid'

Liste.extend(liste)

'Paris'	'Berlin'	'Londres'	'Bruxelles'
0	1	2	3

'Rome'

Liste.insert(index, truc)

'Paris'	'Berlin'	'Londres'	'Bruxelles'
0	1	2	3

'Madrid'

Dictionnaires

<https://machinelearnia.com>

Les **dictionnaires** permettent de relier les éléments d'information connexes. Chaque information est stockée sous la forme d'une paire clé-valeur. Lorsque l'utilisateur fourni une clef, python lui retourne la valeur associée

Dictionnaire

'bananes' : 340
'pommes' : 120
'poires' : 80

A la différence d'une liste, dans un dictionnaire il n'y a pas d'ordre...

Item = (key, value)

Item comprend la clef et la valeur

Déclaration :

```
dictionnaire = {'clef' : valeur, 'clef': valeur}  
dictionnaire.values() # retourne les valeurs  
dictionnaire.keys() # retourne les clefs
```

Ajout :

```
dictionnaire['clef'] = valeur
```

Retrait :

```
dictionnaire.pop(clef)  
del dictionnaire['clef']
```

Parcours : for key, value in dictionnaire.items():

Taille : longueur = len(dictionnaire)

Accès :

```
dictionnaire.get('clef') -> valeur
```

crée un dictionnaire à partir de clef et de valeur :
dict.fromkeys(clés, valeurs)

Built-In functions

<https://machinelearnia.com>

UTILES EN TOUTES CIRCONSTANCES :

`abs()` # valeur absolue

`min()` # minimum

`len()` # longueur

`any()` # y-a-t'il au moins un élément True ?

`all()` # est-ce-que tous les éléments sont True ?

`round()` # arrondi

`max()` # maximum

`sum()` # somme

`format():`

Utile pour insérer la valeur d'une variable au sein d'une chaîne de caractère (string).

f-string permet de faire la même chose plus rapidement

`temperature = 5`

`ville = 'Paris'`

`message = 'il fait {} degrés à {}'.format(x, ville)`

`type():`

Utile pour inspecter le type des variables

`type([1, 2, 10])`

`type(3)`

`type(4.9)`

...

`input():`

Utile pour demander à l'utilisateur d'entrer une valeur dans le programme.

`Input()` retourne un String. Pour demander un âge, il faut convertir en int

`age = int(input())`

`open():`

Utile pour ouvrir n'importe quel fichier de l'ordinateur
Options : 'w', 'r', 'a'

`with open('text.txt', 'w') as f:`
 `f.write('texte')`
 `f.close()`

Quelques Modules de Bases

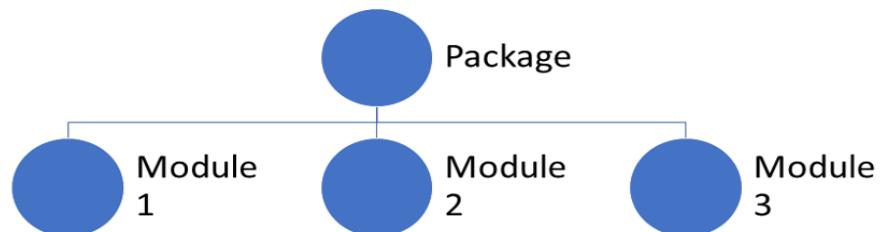
<https://machinelearnia.com>

Pour ne pas avoir à réécrire la même fonction dans tous les fichiers, on l'écrit dans un fichier, appelé **module**, qu'on importe dans ceux qui en ont besoin

Import module (importer tout le module)

Import module as md (donne un surnom au module)

from module import fonction (importe une fonction du module)



Les modules **GLOB** et **OS** sont essentiels pour effectuer des opérations sur le disque dur, comme ouvrir un fichier situé dans un certain répertoire de travail

Fixe le générateur aléatoire pour toujours produire le même résultat : `random.seed(0)`

Choisit un élément au hasard dans la liste : `random.choice(liste)`

Génère un nombre aléatoire entre 0 et 1 : `random.random()`

Génère un nombre entier aléatoire entre 5 et 10 : `random.randint(5, 10)`

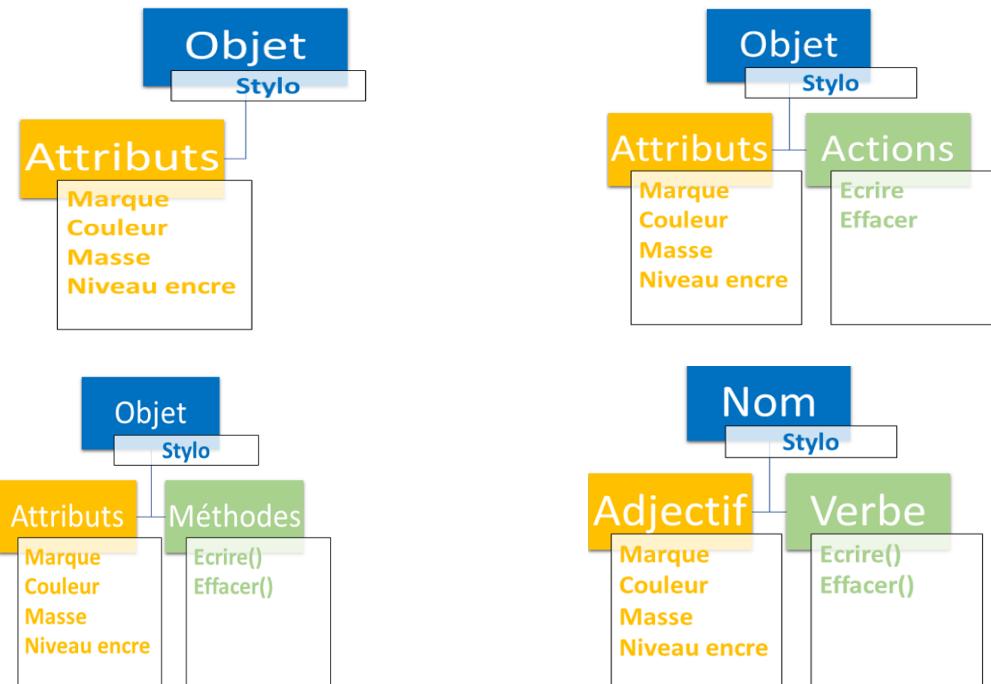
Affiche le répertoire de travail actuel :
`os.getcwd()`

Affiche tout le contenu du répertoire de travail actuel : `glob.glob('*')`

Principe de la Programmation Orientée Objet

<https://machinelearnia.com>

Permet de créer des **entités (objets)**, via des structures solides et claires, que l'on peut manipuler. Les objets peuvent interagir entre eux. Cette approche, en comparaison à la programmation procédurale, **facilite** grandement la **compréhension** du code et sa **maintenance**.



```
class vehicule:  
    """  
    Voici un exemple de classe "vehicule" qui contient le plan de conception  
    d'objets "véhicules"  
    """  
  
    # Une classe commence par une fonction initialisation qui contient les différents attributs  
def __init__(self, couleur='noire', vitesse=0, roues=4):  
    self.couleur = couleur  
    self.vitesse = vitesse  
    self.roues = roues  
  
    # voici une méthode "accelerer" qui modifie un attribut de l'objet  
def accelerer(self, vitesse):  
    self.vitesse += vitesse  
  
    # voici une autre méthode  
def stop(self):  
    self.vitesse = 0  
  
    # voici une dernière méthode, très souvent utilisée  
def afficher(self):  
    print(f'couleur: {self.couleur}\nroues: {self.roues}\nvitesse: {self.vitesse}')
```

Formation Machine Learning

10. Numpy
Tableau N-dim



Formation Machine Learning

11. Numpy
Indexing et Slicing



Formation Machine Learning

12. Numpy
Mathématiques



Formation Machine Learning

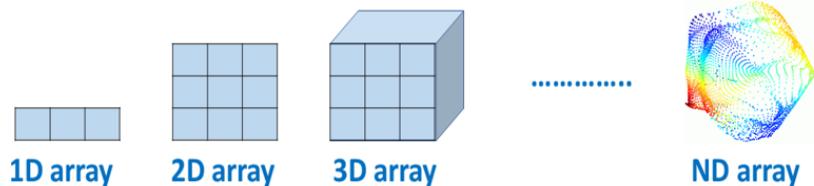
13. Numpy
Broadcasting



Numpy: Le tableau N-dimensionnel

<https://machinelearnia.com>

Numpy est la librairie de base du Data Scientist. Permet de créer et de manipuler des **tableaux** à N - Dimension (1D, 2D, 3D, etc)



Charger Numpy : import numpy as np

Générateur par défaut : **ndarray()**

Générateur 1D ; **np.linspace()** et **np.arange()**

Générateur ND : **np.zeros()**, **np.ones()**, **np.random()**

ndim = 1
shape = (2,)

shape définit la dimension de vos tableaux. C'est un **tuple** !

ndim = 2
shape = (2, 3)

np.float16
 $\pi = 3.14$

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

Moins précis
Plus rapide

np.float64
 $\pi = 3.1415$
926535897
93

Plus précis
Moins rapide

1D Constructeurs :

Génère un tableau de n valeurs entre a et b:

`np.linspace(a, b, n)`

Renvoie des nombres entre a et b avec un pas p:

`np.arange(a, b, p)`

ND Constructeurs :

Tableau de forme et de type donnés, rempli avec value :
`np.full(shape, value)`

tableau de zeros : `np.zeros(shape)`

tableau de 1 : `np.ones(shape)`

matrice identité : `np.eye()`

tableau d'entier aléatoire :
`np.random.randint(shape)`

Manipulations :

`array[ligne, colonne]`

Empilez les tableaux dans l'ordre horizontal : `np.hstack((array A, array B))`
Si shape égales, empile les tableaux dans l'ordre vertical :

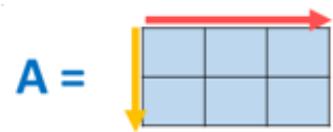
`np.vstack((array A, array B))`

Renvoie un tableau contigu aplati : `A.ravel()`

Donne une nouvelle forme à un tableau sans en modifier les données :
`A.reshape(3,4)`

Numpy: Indexing et Slicing

<https://machinelearnia.com>



les tableaux de n dimensions peuvent être **indexés**

A[*ligne, colonne*]

1 Dimension :

Premier élément : Array[0]

Dernier élément : Array[-1]

Deuxième élément : Array[1]

n-ième élément : Array[n]

2 Dimension :

Première ligne, première colonne : Array[0, 0]

Première ligne, deuxième colonne : Array[0, 1]

Deuxième ligne, première colonne : Array[1, 0]

N Dimension :

Même principe...

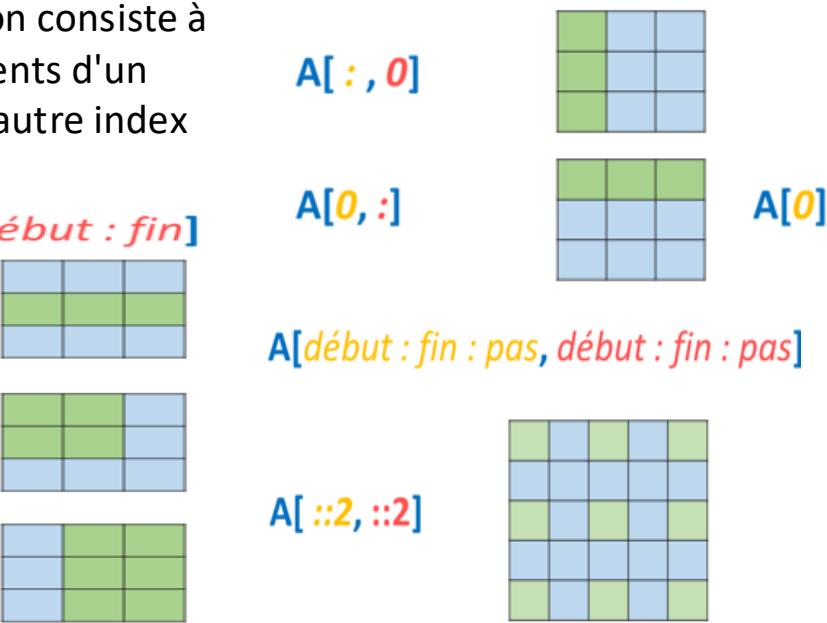
Le **slicing** en python consiste à prendre des éléments d'un index donné à un autre index donné.

A[*début : fin, début : fin*]

A[*1, :*]

A[*0:2, 0:2*]

A[*?, ?*]



Boolean Indexing :

Cet indexage avancé se produit lorsque l'objet est un tableau d'objets de type booléen, comme ceux renvoyés par les opérateurs de comparaison.

Exemple :

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

Masque booléen : A<5

Sous-ensemble filtré par le masque booléen : A[A < 5]

Converti les valeurs sélectionnées : A[A < 5]

Numpy: Opérations Mathématiques

<https://machinelearnia.com>

Quelques méthodes :

A.sum(axis), A.cumsum(axis)
A.prod(axis), A.cumprod(axis)

Quelques fonctions :

np.exp(axis), np.log(axis)
np.sin(axis), np.cos(axis)

Quelques méthodes de ndArray :

Retourne la valeur min/max sur l'axe spécifié :

A.min(axis), A.max(axis)

Retourne la position du min/max sur l'axe : A.argmin(axis),
A.argmax(axis)

Retourne l'écart type des éléments le long de l'axe : A.std(axis)

Retourne la moyenne des éléments sur l'axe : A.mean(axis)

Trie le tableau A, l'ordre des éléments change : A.sort(axis)

Retourne les index dans l'ordre du tableau : A.argsort(axis)

Retourne la matrice de
corrélation de A :

np.corrcoef(A)

	L1	L2
L1	1	L1L2
L2	L2L1	1

Retourne les valeurs distinctes du
tableau et leur fréquence

np.unique(A, return_counts = True)

A =	5	0	3	Entités =	0	3	5	7	9
	3	7	9	Répétitions =	1	2	1	1	1

Constructeurs :

np.array(objet, dtype)
np.zeros(shape, dtype)
np.ones(shape, dtype)
np.random.randn(lignes, colonnes)

Attributs :

ndarray.shape
ndarray.size

NaN :

Insère un Nan dans la matrice A :

A[0, 2] = np.nan

Calcule la proportion de NaN dans A :

np.isnan(A).sum()/A.size

Calcule la moyenne de A en ignorant les NaN :

np.nanmean(A)

Manipulation :

A.reshape

A.squeeze

A.concatenate

Algèbre Linéaire :

Transposée de A : A.T

Produit matriciel A.B : A.dot(B)

Calcul de l'inverse de A : np.linalg.inv(A)

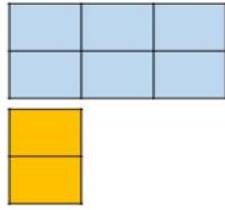
Calcul du déterminant de A : np.linalg.det(A)

Valeur propre, vecteur propre : val, vec = np.linalg.eig(A)

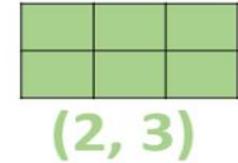
Numpy: Broadcasting

<https://machinelearnia.com>

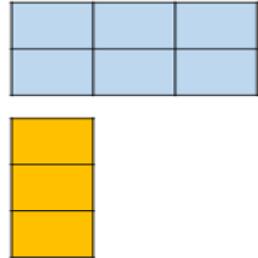
Le terme **broadcasting** décrit la manière dont NumPy traite les tableaux de formes différentes lors des opérations arithmétiques.



(2, 3)
(2, 1)



(2, 3)



(2, 3)
(3, 1)



La règle est simple : A et B dimensions **égales** ou égale à 1

$$\begin{matrix} X_1 \\ X_2 \\ X_3 \\ \dots \\ X_{99} \\ X_{100} \end{matrix} - \begin{matrix} Y_1 \\ Y_2 \\ Y_3 \\ \dots \\ Y_{99} \\ Y_{100} \end{matrix} = \begin{matrix} X_1 - Y_1 \\ X_2 - Y_2 \\ \dots \\ \dots \\ \dots \\ X_{100} - Y_{100} \end{matrix}$$

Exemples :

A = np.ones((2, 3))

B = 3

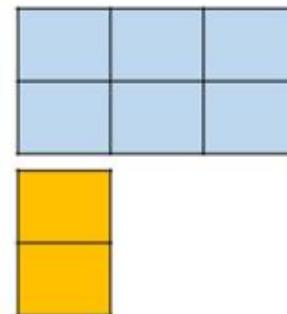
A + B = 1

A = np.ones((2,3))

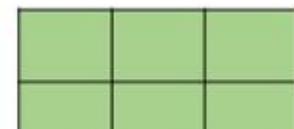
B = np.ones((2,1))

A + B = 2

B a une colonne, elle sera étendue sur les trois colonnes de A.



(2, 3)
(2, 1)



(2, 3)

Formation Machine Learning

14. Matplotlib
Les Bases



Formation Machine Learning

15. Graphiques
en Data Sciences



Matplotlib: Construction d'un graphique

<https://machinelearnia.com>

Matplotlib est la librairie python la plus utilisée pour créer des graphiques.

Charger Matplotlib : import matplotlib.pyplot as plt

2 Graphiques :

Tracer un graphe : plt.plot(X,Y)

Tracer un nuage de point : plt.scatter(X,Y)

Attention : X et Y doivent être de dimensions égales

Affichage :

plot(x, y, **label=...**, **lw=...**, **ls=...**, **c=...**)



Rester simple...

Options d'affichage :

Couleur de la ligne : c

Epaisseur de la ligne (pour les graphiques plot) : lw

Style de la ligne (pour les graphiques plot) : ls

Taille du point (pour les graphiques scatter) : size

Style de points (pour les graphiques scatter) : marker

Transparence du graphique : alpha

Début de la figure : plt.figure()

Création :

plt.plot(...,...)

plt.plot(...,...)

Titre de l'axe : plt.xlabel('axe')

Titre du graphique : plt.title('titre')

Affiche la légende : plt.legend()

Affiche le graphe : plt.show()

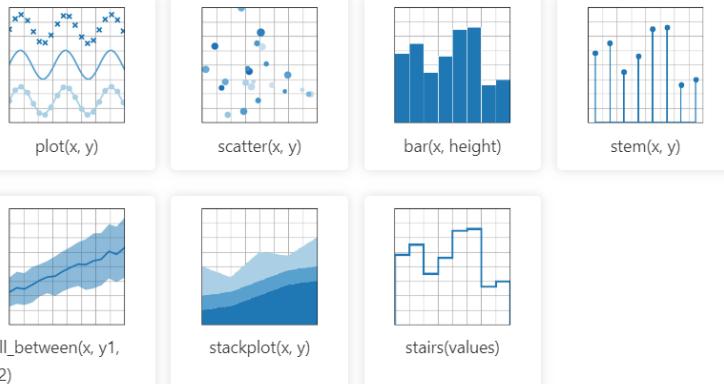
Enregistre la figure : plt.savefig('text.png')

Matplotlib: Les différentes types de graphiques

<https://machinelearnia.com>

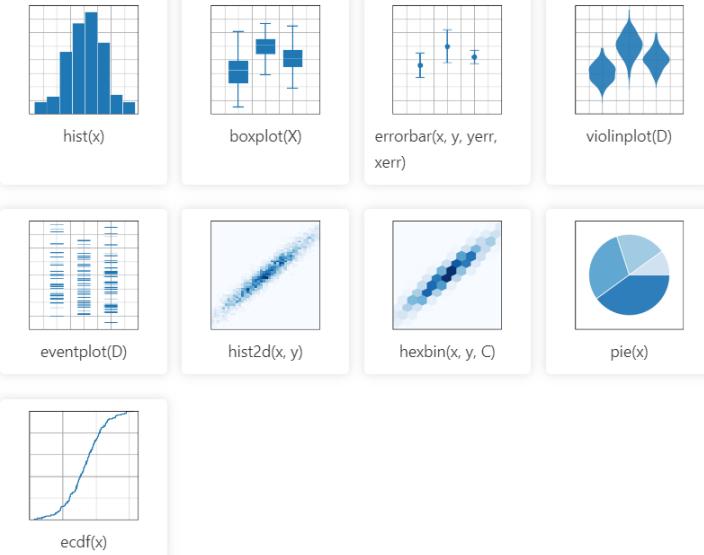
Pairwise data

Plots of pairwise (x, y) , tabular ($\text{var}_0, \dots, \text{var}_n$), and functional $f(x) = y$ data.



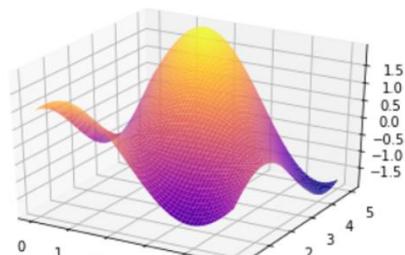
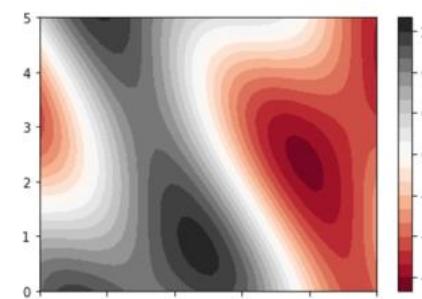
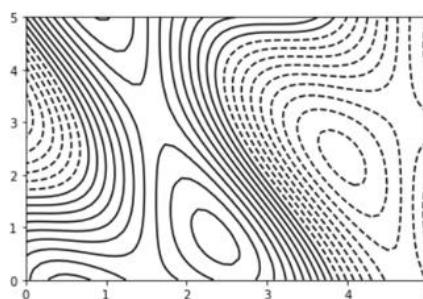
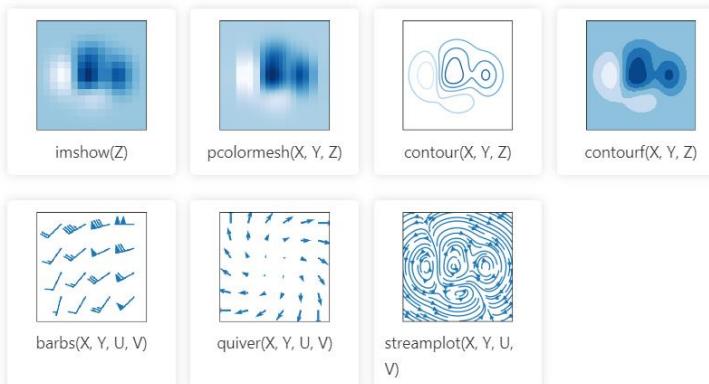
Statistical distributions

Plots of the distribution of at least one variable in a dataset. Some of these methods also compute the distributions.



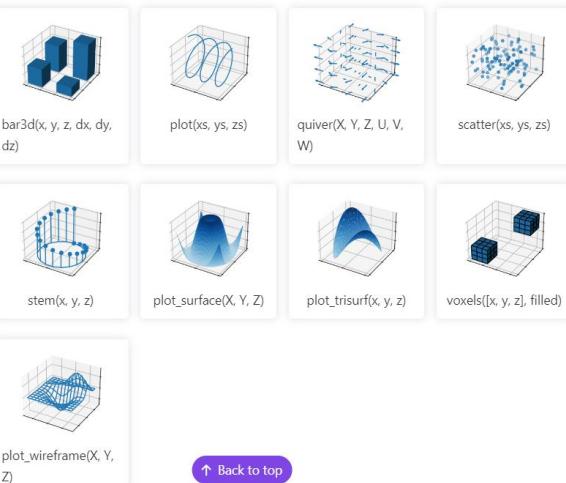
Gridded data

Plots of arrays and images $Z_{i,j}$ and fields $U_{i,j}, V_{i,j}$ on [regular grids](#) and corresponding coordinate grids $X_{i,j}, Y_{i,j}$.



3D and volumetric data

Plots of three-dimensional (x, y, z) , surface $f(x, y) = z$, and volumetric $V_{x,y,z}$ data using the `mpl_toolkits.mplot3d` library.



[↑ Back to top](#)

Formation Machine Learning

16. Scipy
Calcul Scientifique



Scipy: Interpolation

<https://machinelearnia.com>

Dans un dataset, il peut arriver que des valeurs manquent. Scipy propose plusieurs fonctions pour lisser les données et d'**interpolations** pour les dimensions 1 et 2.

Le choix d'une routine d'interpolation spécifique dépend des données : si elles sont unidimensionnelles, si elles sont données sur une grille structurée ou si elles ne sont pas structurées.

```
from scipy.interpolate import interp1d
```



Capteur 1	Capteur 2
0.02	25
0.24	Nan
0.60	Nan
0.93	27
1.23	Nan
1.45	Nan
1.8	28.5

Ces valeurs ne sont pas gâchées, on peut tenter une interpolation

Création de la fonction interpolation f

```
f = interp1d(x, y, kind = 'cubic')
```

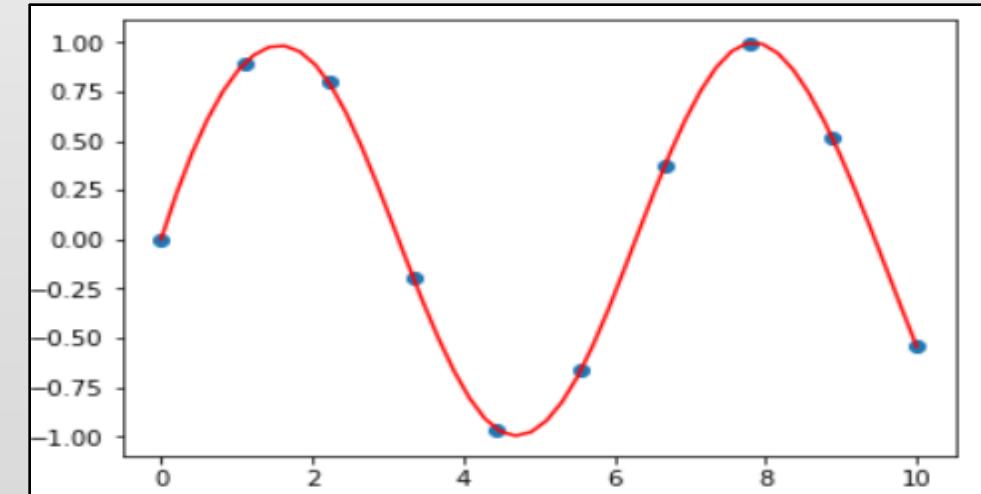
Résultats de la fonction interpolation f sur de nouvelles données :

```
new_x = np.linspace(0, 10, 50)
```

Visualisation avec matplotlib :

```
plt.scatter(x, y)
```

```
plt.plot(new_x, result, c='r')
```



Attention à ne pas cacher la réalité...

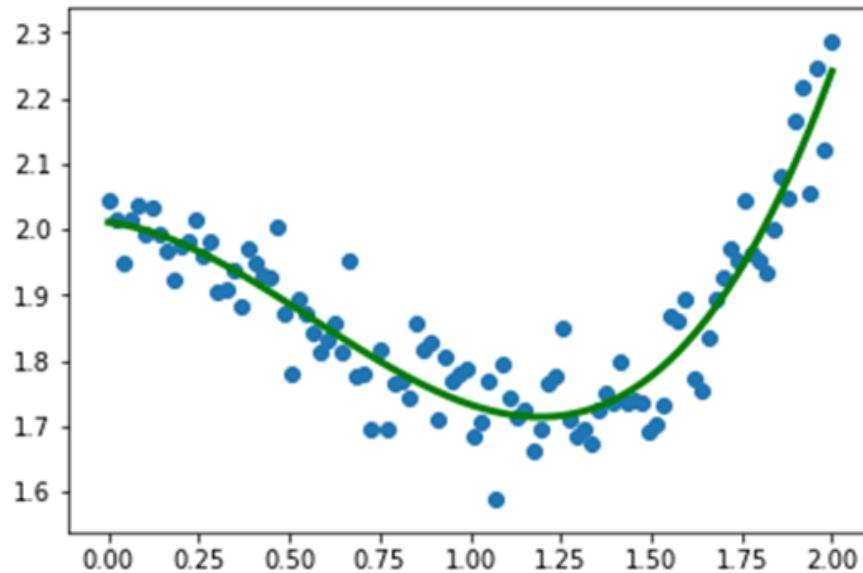
Scipy: Optimisation

<https://machinelearnia.com>

Le paquet **scipy.optimize** fournit plusieurs algorithmes d'optimisation couramment utilisés.

```
from scipy import optimize
```

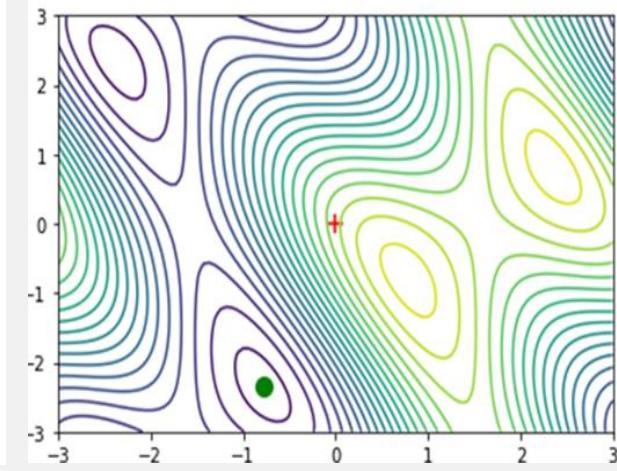
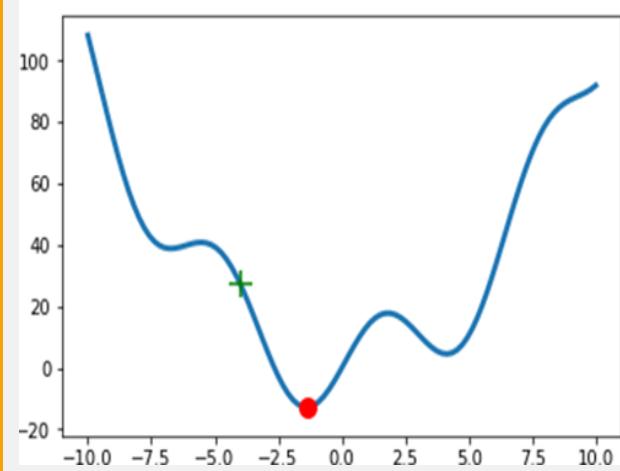
Méthode des moindres carrés pour trouver les bons paramètres : `optimize.curve_fit`



Méthode du minimum local pour trouver les bons paramètres, selon un point de départ x_0 :
`optimize.minimize`

Point de départ : $x_0 = -5$

Résultat : `optimize.minimize(f ,x0=x0).x`



Fonctionne aussi avec N-dimension...

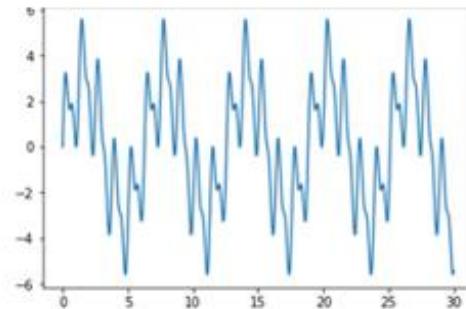
Scipy: Fourier

<https://machinelearnia.com>

Transformation de Fourier : Analyse les fréquences présentes dans un signal périodique.

```
from scipy import signal  
from scipy import fftpack
```

Signal =



Création des variables Fourier et Fréquences, pour construire le spectre du signal :

```
fourier = fftpack.fft(signal)
```

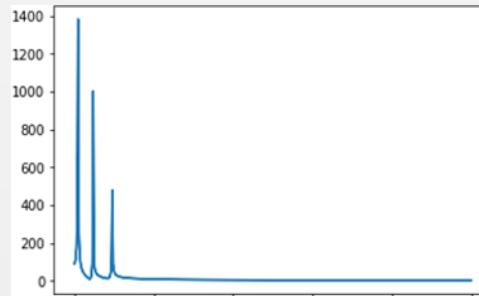
Pour éliminer les amplitudes négatives :

```
power = np.abs(fourier)
```

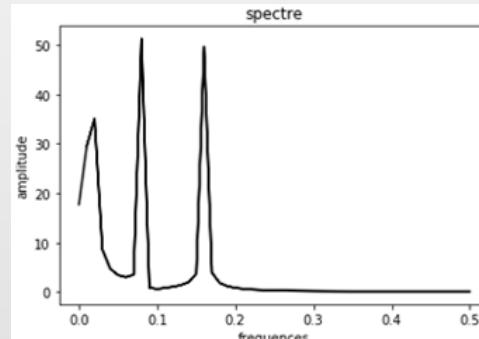
Construction du spectre :

```
fréquences = fftpack.fftfreq(signal.size)
```

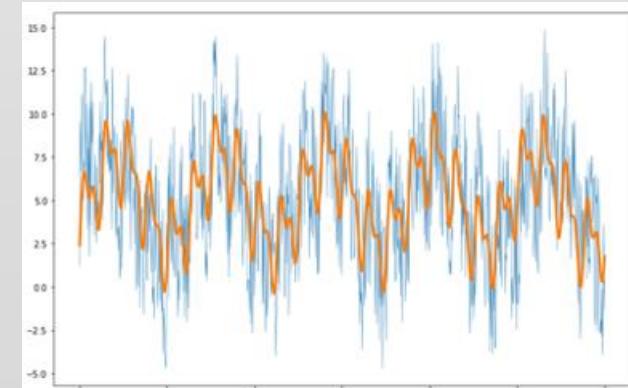
Filtre du spectre avec du boolean indexing de Numpy :
`fourier[power<400] = 0`



Visualisation du spectre propre :
`plt.plot(np.abs(fréquence),
np.abs(fourier))`



Transformation de Fourier inverse : génère un nouveau signal temporel depuis le spectre filtré :
`fftpack.ifft(fourier)`



Scipy: Traitement d'image

<https://machinelearnia.com>

Scipy comporte également des fonctions basiques pour le traitement d'image.

Le package **scipy.ndimage** contient diverses fonctions pour le traitement d'images multidimensionnelles.

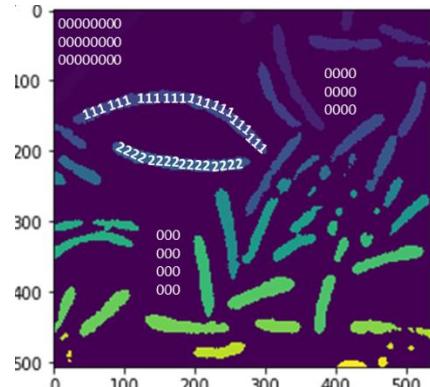
```
from scipy import ndimage
```

Importer l'image :

```
image = plt.imread('bacteria.png')
```

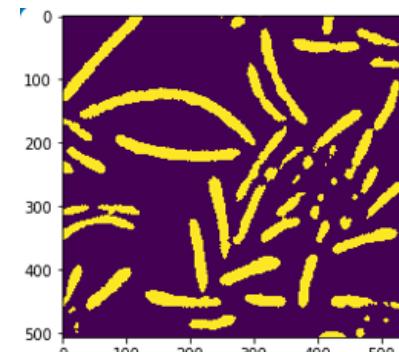
Réduire l'image en 2D :

```
image = image[:, :, 0]
```



Afficher l'image :

```
plt.imshow(image, cmap='gray')  
image.shape
```



Morphologie utilisée pour enlever les artefacts :

```
open_image = ndimage.binary_opening(image)  
plt.imshow(open_image)
```

Segmentation de l'image :

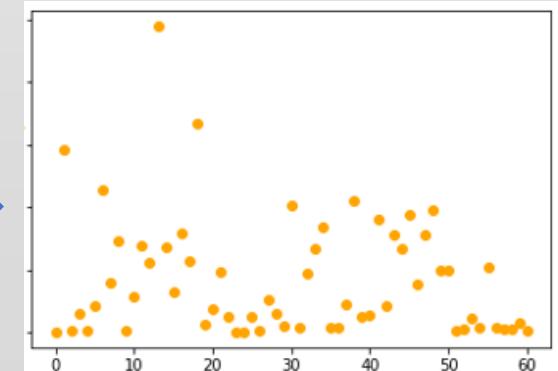
```
label_image, n_labels = ndimage.label(open_image)
```

Visualisation de l'image étiquetée :

```
plt.imshow(label_image)
```

Mesure de la taille de chaque groupe de label_images (fait la somme des pixels) :

```
Sizes = ndimage.sum(open_image, label_image, range(n_labels))
```



Formation Machine Learning

17. Pandas
Les Bases



Formation Machine Learning

18. Pandas
Série Temporelle



Formation Machine Learning

19. Seaborn
Data Visualisation



Pandas: Le DataFrame et ses opérations de Bases

<https://machinelearnia.com>

Pandas est souvent connu comme étant Excel dans Python. Au cœur de Pandas, on retrouve le DataFrame, qui est un tableau 2D dont on peut accéder aux éléments grâce à un index et leurs colonnes.

Charger Pandas : import pandas as pd

Ouvrir un fichier .xls : pd.read_excel()

Ouvrir un fichier .csv : pd.read_csv()

Ouvrir un fichier .json : pd.read_json()

Etc...

Affiche le début du DataFrame : df.head()

Statistiques rapides : df.describe()

Éliminer certaines colonnes : df.drop(['column', ...])

Éliminer les lignes aux données manquantes :

df.dropna(axis = 0)

Compter les répétitions : df.[column'].value_counts()

Analyse par groupe : df.groupby(['column'])

Définit une ou plusieurs colonnes d'un DataFrame comme
index : set_index()

Remplace les valeurs d'un DataFrame : replace()

Panda utilise Matplotlib.pyplot :

Génère un graphique linéaire du DataFrame : df.plot()

Génère un graphique en barres du DataFrame : df.plot.bar()

Génère des histogrammes pour chaque colonne du DataFrame avec bins le nombre de bacs dans l'histogramme : df.hist(bins =...)

Génère un graphique de dispersion avec des colonnes spécifiées du DataFrame : df.plot.scatter(x=..., y=...)

Génère une matrice de graphique de dispersion pour chaque colonne numérique du DataFrame :

pd.plotting.scatter_matrix(df)

Pour plus de performance, on utilisera **Seaborn**

Pandas: Traitement sur les Séries temporelles (1)

<https://machinelearnia.com>

Une série temporelle est une série de points de données indexés dans le temps = Tableau Numpy 1D + axe. Son analyse est fréquemment utilisée notamment dans la finance.

Code	Meaning
Y	Year
M	Month
W	Week
D	Day
h	Hour
m	Minute
s	Second
ms	Millisecond
us	Microsecond
ns	Nanosecond

On peut étudier les données selon une fréquence. Une fois qu'on a des groupes on peut étudier les données :
mean(), std(), min(), max(), sum() ...

pclass	survived	sex	age
0	1	1	female 29.0000
1	1	1	male 0.9167
2	1	0	female 2.0000
3	1	0	male 30.0000
4	1	0	female 25.0000

Df [‘column’] = une Série

Ndarry : data[‘age’] = Série

Indexing : data[‘age’][0:10]

Mask : data[‘age’] < 18

Boolean indexing :

data[data[‘age’] < 18]

On peut également faire de l’indexing sur des dates :

data[‘2016’ : ‘2019’, ‘Close’]

Panda s’adapte au format de date :

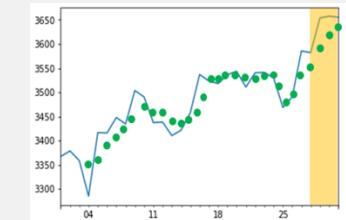
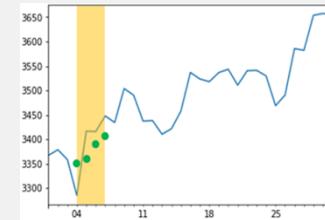
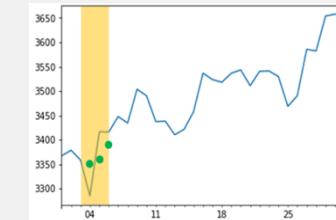
‘2019-03-20’ ‘2019/03/20’ ‘2019.03.20’ ‘2019 03 20’ ...

Convertit les arguments en un objet datetime :

pd.to_datetime(‘2019/03/20’)

Moyenne mobile :

On fait la moyenne sur une succession de fenêtre de valeurs :



→ rolling(window, center = True).mean()

Moyenne mobile exponentielle (ewm) :

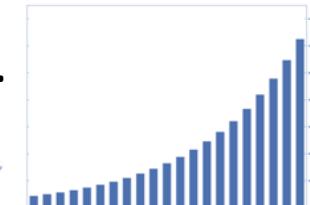
Accorde + de poids aux observations récentes, qui permet de saisir les **tendances récentes rapidement**.

$$\bar{x}_t = \sum_{n=0}^{\infty} \alpha(1 - \alpha)^n x_{t-n}$$

ewm(alpha=0.5).mean()

Exemple :

```
for i in np.arange(0.2, 1, 02):  
    df[‘2019-09’, ‘Close’].ewm(alpha=i).mean()
```



Pandas: Traitement sur les Séries temporelles (2)

<https://machinelearnia.com>

Merge et Join :

Series		Series		=	DataFrame	
apples	3	oranges	0		apples	oranges
0	3	0	0		0	0
1	2	1	3		1	3
2	7	2	7		2	Nan
3	2	3	Nan		3	2

Ou bien

	apples	oranges
0	3	0
1	2	3

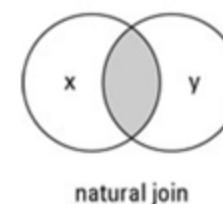
Problème de dimension avec **Numpy**. Mais pas avec **Panda**.

Exemple :

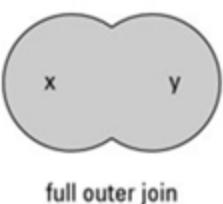
```
pd.merge(Serie1, Serie2, on='X1',  
how='inner', suffixes=('S1', 'S2'))
```

On peut merger des DataFrames ou séries de différentes manières : inner, outer, ...

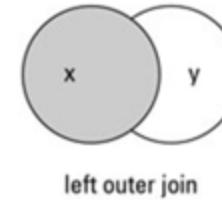
how='inner'



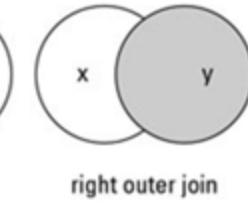
how='outer'



how='left'



how='right'



Map et Apply :

La fonction map() applique une fonction sur chaque élément d'une colonne.

```
data['age'].map(lambda x: x+1)
```

La fonction apply() applique une fonction sur chaque élément du DataFrame.

```
s = pd.Series([1, 2, 3, 4, 5])  
s = s.apply(lambda x: x * 2)
```

Pour faire des calculs (ML, stats, etc...) il faut absolument avoir des données numériques

pclass	survived	sex	age
0	1	1 female	29.0000
1	1	1 male	0.9167
2	1	0 female	2.0000
3	1	0 male	30.0000
4	1	0 female	25.0000

Conversion :

```
Data['sex'].map( {'male':0 , 'female':1} )
```

```
Data['sex'].replace(['male', 'female'] , [0,1] )
```

```
Data['sex'].astype('category').cat.codes
```

Seaborn: Une alternative à Matplotlib

<https://machinelearnia.com>

Seaborn est un package construit sur la base de Matplotlib, et permet de gagner du temps pour constituer les graphiques les plus utilisés en Data Science. Il s'intègre particulièrement bien à Pandas.

Charger Seaborn : import seaborn as sns

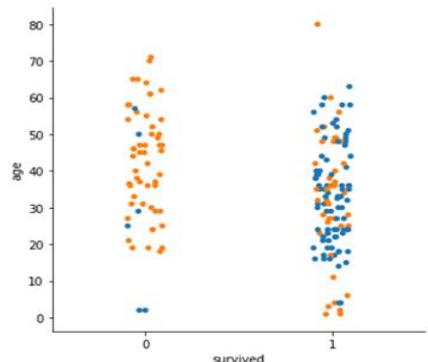
Les fonctions Seaborn ont presque toutes la même **structure**

`sns.fonction(x, y, data, hue, size, style)`

Les données à afficher *Options de segmentation*

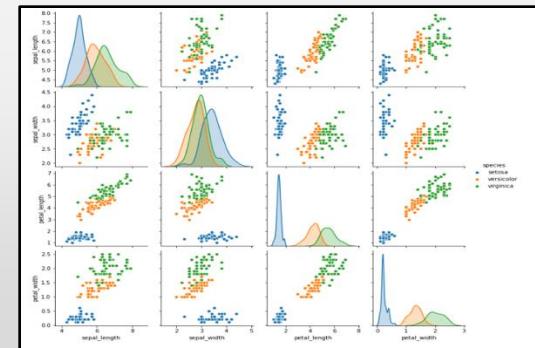
Exemple :

```
sns.catplot(x = 'survived',  
y='age',  
data=titanic,  
hue='sex')
```

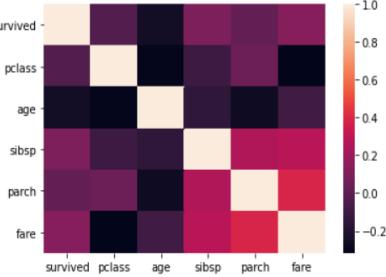
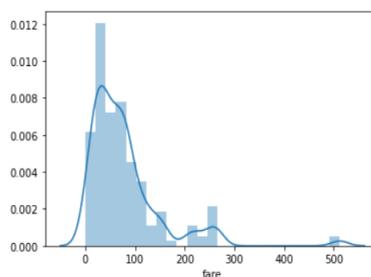
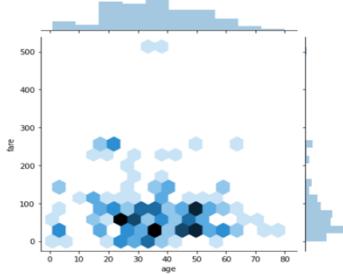
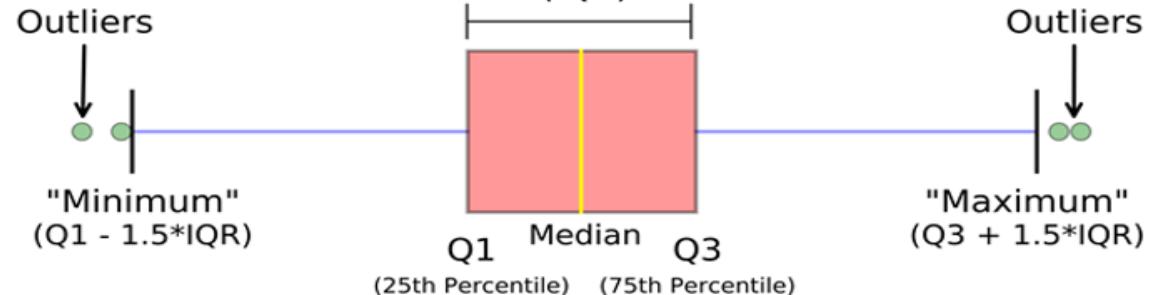


Les fonctions les plus utiles :

Pairplot()
Distplot()
Catplot()
Jointplot()
Boxplot()
Heatmap()



Interquartile Range (IQR)



Formation Machine Learning

20. Les Modèles
de Base

scikit
learn

Sklearn: Modele de Regression et Classification

Sklearn est la librairie Python de référence pour développer des modèles de Machine Learning :
On distingue l'**apprentissage supervisé, non-supervisé et par renforcement**



Différents mécanismes, mais la même interface : fit + score + predict

Charger Sklearn : import sklearn as sk

1. Sélectionner un **estimateur** et préciser ses **hyperparamètres** :

`model = LinearRegression(...)`
objet Constructeur Hyperparamètres

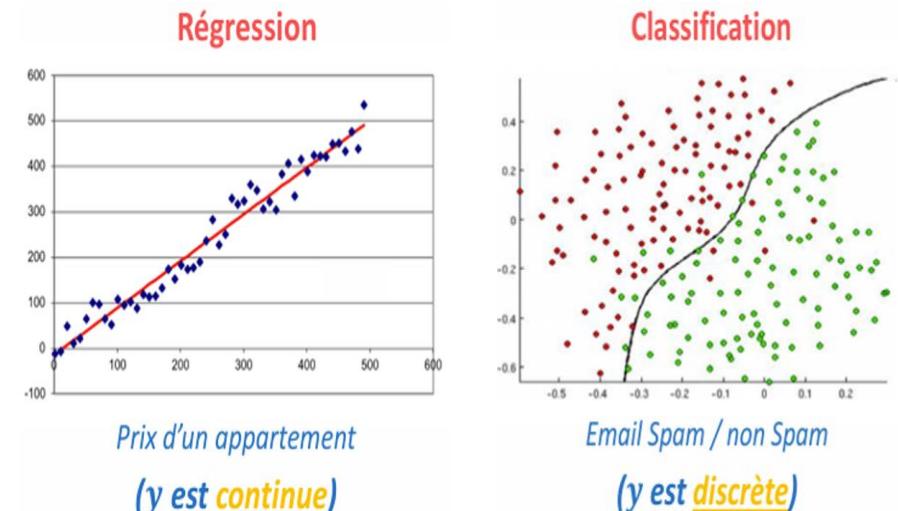
2. Entrainer le modèle sur les données **X, y** (divisées en 2 tableaux **Numpy**)
`model.fit(X, y)`

3. Évaluer le modèle
`model.score(X, y)`

4. Utiliser le modèle
`model.predict(X)`

Exemple :

`model = RandomForestClassifier(n_estimators = 100)`
`Learning rate = 0.3 : model = SGDRegressor(eta0 = 0.3)`



`model = LinearRegression()`
`model.fit(X, y)`
`model.score(X, y)`
`model.predict(X)`



`model = DecisionTreeClassifier()`
`model.fit(X, y)`
`model.score(X, y)`
`model.predict(X)`

Formation Machine Learning

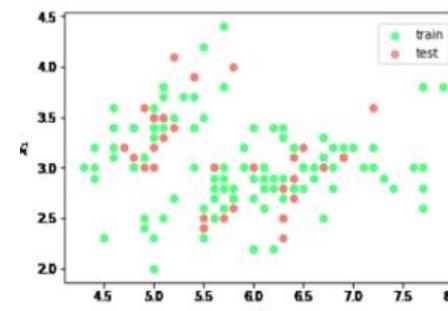
21. Évaluation
Cross-Validation

scikit
learn

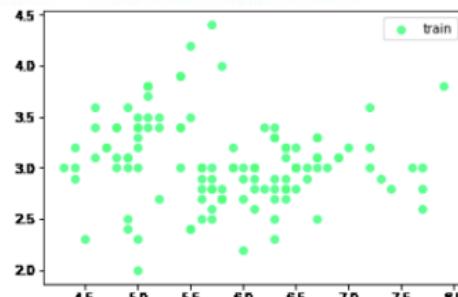
Sklearn: Train Test Split

<https://machinelearnia.com>

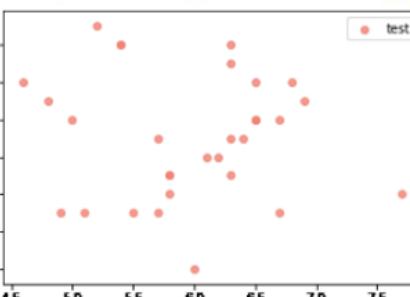
La fonction **train_test_split** permet de séparer le jeu de données initial en deux ensembles : un **training set** et un **test set**. Le training set est utilisé pour **fit** (entrainer) le modèle sur une partie des données. Le test set est utilisé pour **évaluer les performances** du modèle.



$X_{train}, X_{test}, y_{train}, y_{test} = \text{train_test_split}(X, y, \text{random_state} = 0)$



$\text{model.fit}(X_{train}, y_{train})$



$\text{model.score}(X_{test}, y_{test})$

```
from sklearn.neighbors import KNeighborsClassifier
```

Entrainer le modèle :

```
model = KNeighborsClassifier(n_neighbors = 1)
```

```
model.fit(X_train, y_train)
```

Attention :

Limite principale liée à **random_state**. Ce nombre (seed) peut fournir des résultats différents lorsqu'il est changé pour un même dataset.

Solution : Calculer la moyenne des scores obtenus sur différents essais.

Validation Set :

Data set		
Train set	Val set	Test set
100%	92%	91%

The table illustrates a validation set scenario. It shows a "Data set" divided into three parts: "Train set", "Val set", and "Test set". The "Train set" contains three cat icons. The "Val set" contains two cat icons, with one circled in red. The "Test set" contains two cat icons. Below the table, two rows of numbers represent scores: 100% for the first row and 92% for the second row under the "Val set" column, and 91% for the third row under the "Test set" column. A red circle highlights the 92% value.

Sklearn: Validation Curve

<https://machinelearnia.com>

Validation Curve : Permet de tester toutes les valeurs pour un hyperparamètre donné. Calcule le score sur Train set et Val set grâce à la Cross Validation

```
validation_curve(model, Xtrain, ytrain,  
                  'hyperparamètre', valeurs, cv=5)
```



train_{score}, val_{score}

```
from sklearn.neighbors import validation_curve
```

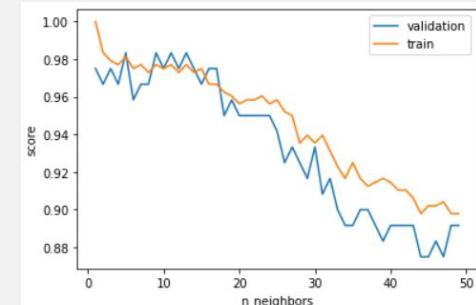
Exemple :

49 valeurs sont testées avec cv = 5
→ trainscore et valscore sont de dimensions (49,5)

```
model = KNeighborsClassifier()
```

```
K = np.arange(1, 50)
```

```
Train_score, val_score = validation_curve(model,  
                                         X_train,  
                                         y_train,  
                                         'n_neighbors',  
                                         k,  
                                         cv = 5)
```

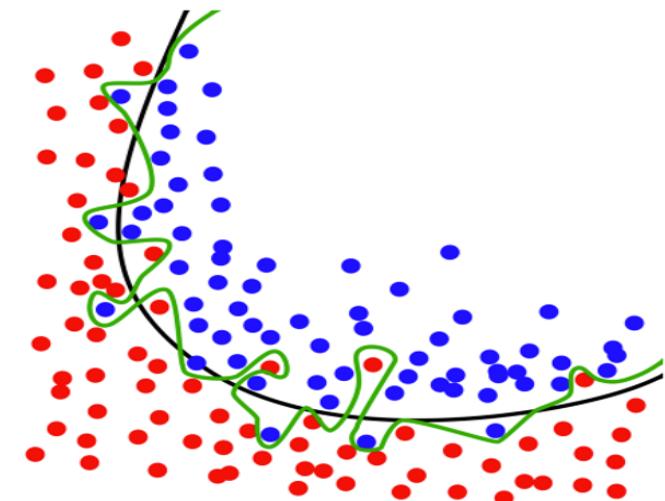


Overfitting :

Le modèle s'est trop perfectionné sur le Trainset et a perdu tout sens de généralisation

→ Bon *Train_{score}*

→ Mauvais *Test_{score}*



Sklearn: Learning Curve

<https://machinelearnia.com>

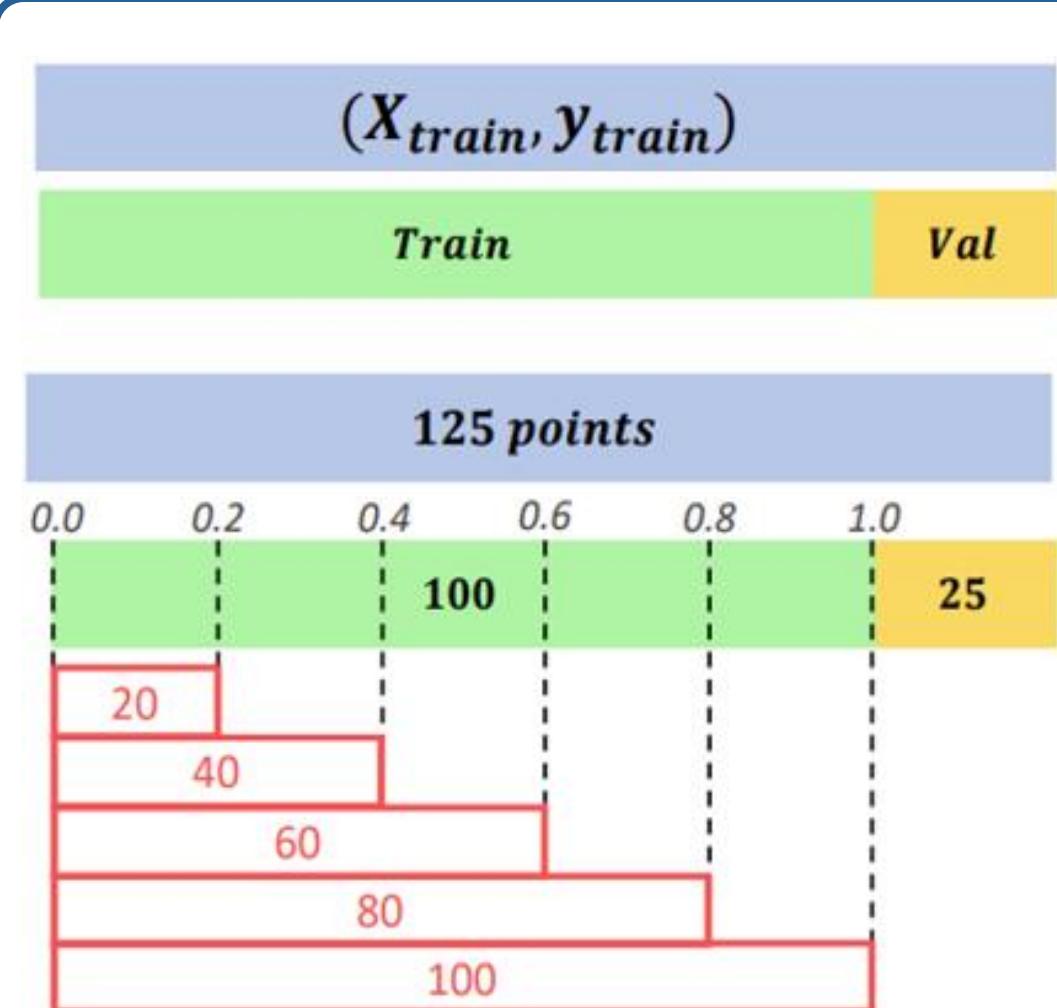
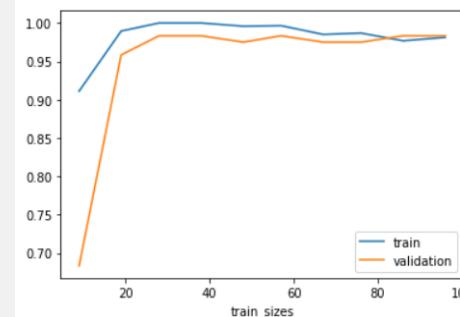
Learning Curve : Détermine les scores de formation et de test validés par cross-validation pour différentes tailles d'ensembles de formation.

`learning_curve(model, X, y, train_sizes, cv=5)`

→ N , $\text{train}_{\text{score}}$, $\text{val}_{\text{score}}$

```
from sklearn.neighbors import learning_curve
```

```
N, train_score, val_score = learning_curve(model,  
X_train,  
y_train,  
train_size=np.linspace(0.1, 1, 10),  
cv = 5)
```



`train_sizes = np.linspace(0.2, 1.0, 5)`

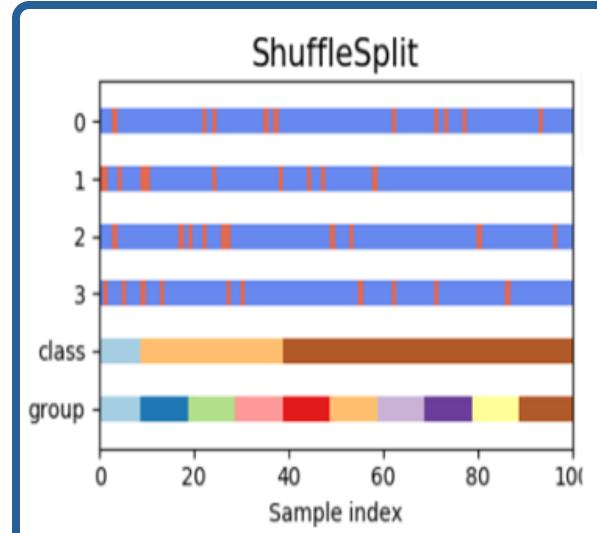
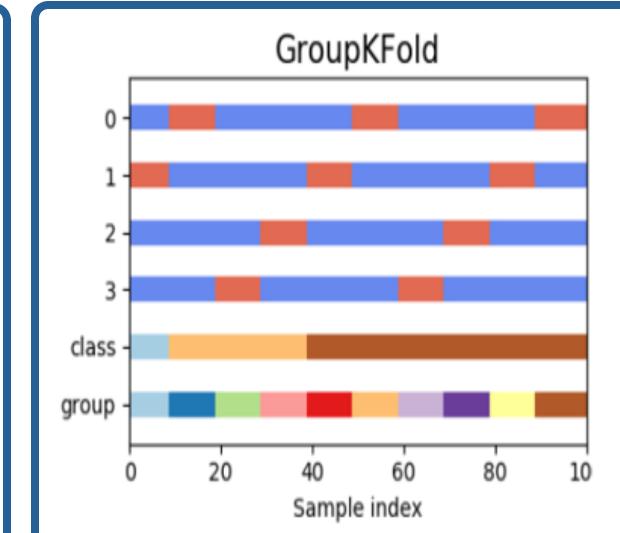
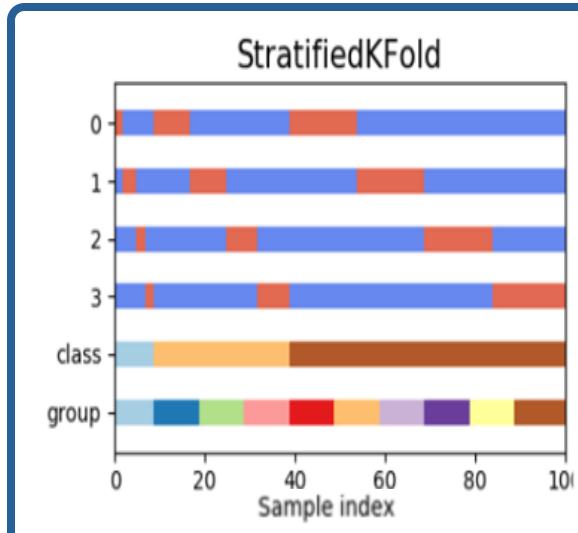
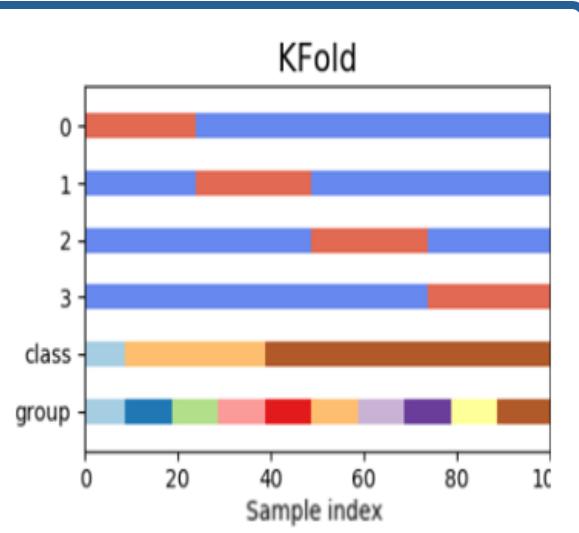
Sklearn: Cross Validation

<https://machinelearnia.com>



```
from sklearn.neighbors import cross_val_score
```

```
model = KNeighborsClassifier()  
cross_val_score(model, X_train, y_train, cv=5, scoring='accuracy')  
  
val_score = []  
For k in range(1, 50):  
    score = cross_val_score(KNeighborsClassifier(k),  
                           X_train,  
                           y_train,  
                           cv=5).mean()  
    val_score.append(score)
```



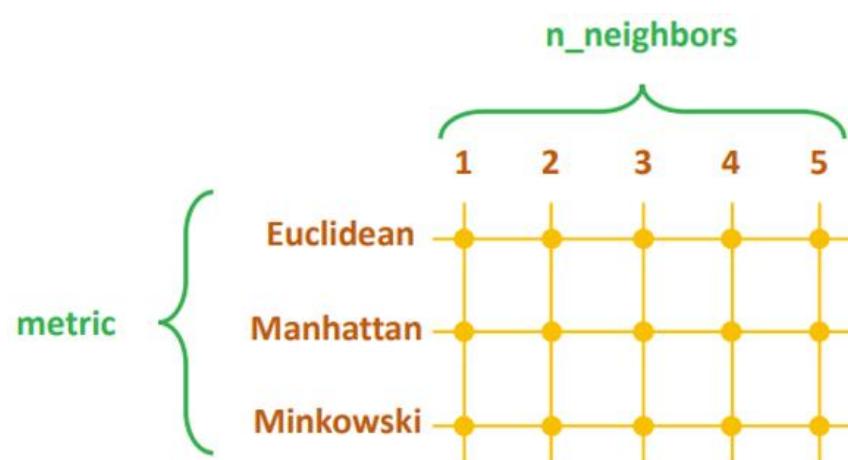
GridSearchCV : Permet de construire une grille de modèles avec toutes les combinaisons d'hyperparamètres présents dans param_grid.

Grid = GridSearchCV(model, param_grid, cv)

```
from sklearn.neighbors import GridSearchCV
```

```
Param_grid = {'n_neighbors' : np.arange(1, 20),  
             'metric' : ['euclidean', 'manhattan']}
```

```
Grid = GridSearchCV(KNeighborsClassifier(),  
                     param_grid,  
                     cv = 5)  
  
Grid.fit(X_train, y_train)
```



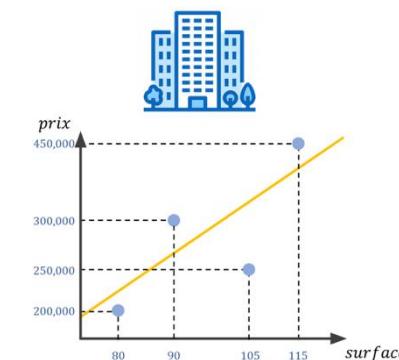
La grille est un **estimator** !
→ entraînement: **fit()**
→ **best_score_**
→ **best_params_**
→ **best_estimator_**

```
model =  
grid.best_estimator_.score(X_test, y_test)
```

Sklearn: Métrique de Regression

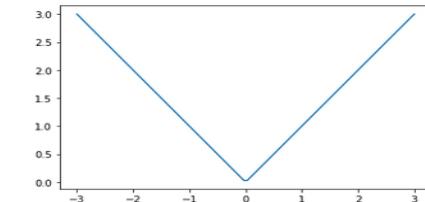
<https://machinelearnia.com>

Les **métriques de régressions** permettent d'évaluer les performances des modèles de regression. On en distingue principalement 3.



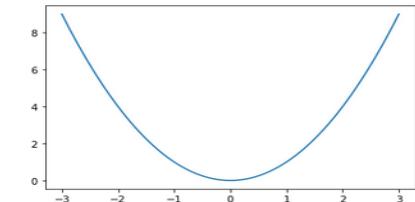
Mean Absolute Error

l'importance d'une erreur est **linéaire** avec son amplitude



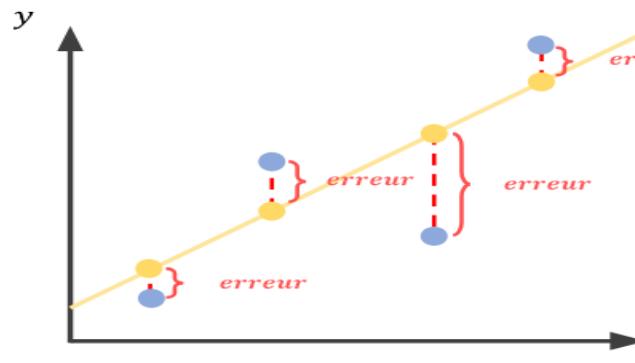
Root Mean Squared Error

l'importance d'une erreur est **exponentielle** avec son amplitude



la moyenne des erreurs

$$\frac{1}{m} \sum \text{erreur}$$

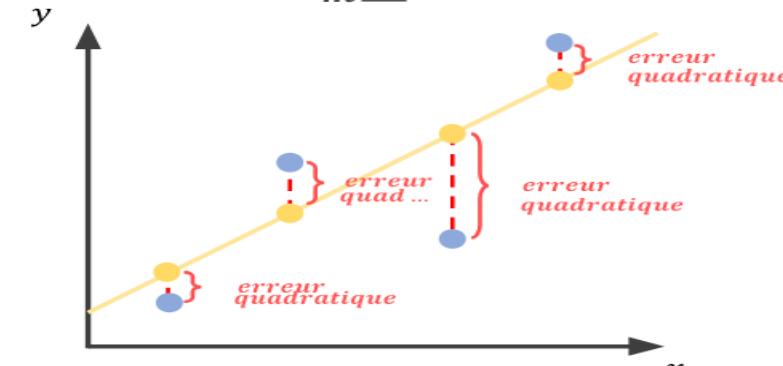


Mean Absolute Error

$$\frac{4 + 0}{2} = 2$$

Erreur quadratique moyenne

$$MSE = \frac{1}{m} \sum (y_{vrai} - y_{pred})^2$$

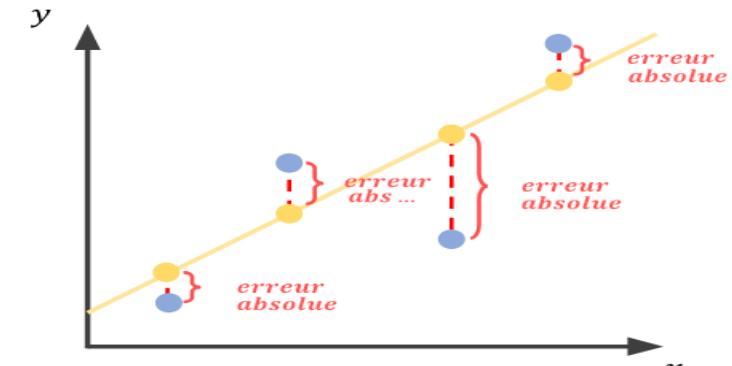


Root Mean Squared Error

$$\sqrt{\frac{4^2 + 0^2}{2}} = 2.8$$

Erreur absolue moyenne

$$MAE = \frac{1}{m} \sum |y_{vrai} - y_{pred}|$$



$$\sum (\text{erreurs})^2 \neq \left(\sum \text{erreurs} \right)^2$$

$$MSE \neq (MAE)^2$$

Sklearn: Métrique de Régression

<https://machinelearnia.com>

MSE → vous accordez une grande importance aux grandes erreurs.

MAE → l'importance d'une erreur est linéaire avec son amplitude. Si le Dataset contient des valeurs aberrantes (*outliers*).

Il existe aussi la *Median Absolute Error*.

Très peu sensible aux grandes erreurs.

$$MAE = median \{ |y_{vrai} - y_{pred}| \}$$

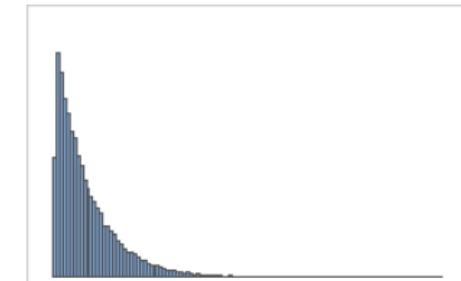
Attention quand vous l'utilisez....

MAE, MSE, RMSE, Médiane...

Il semble que chaque mesure est des **avantages** et des **inconvénients**... Du coup, laquelle choisir ?

→ Utilisez les toutes !

Vous récolterez ainsi beaucoup plus d'information !



$$R^2 = 1 - \frac{\sum(y_{vrai} - y_{pred})^2}{\sum(y_{vrai} - \bar{y}_{vrai})^2}$$

variance *moyenne*

Évalue la **performance** du modèle par rapport au niveau de variation présent dans les données.

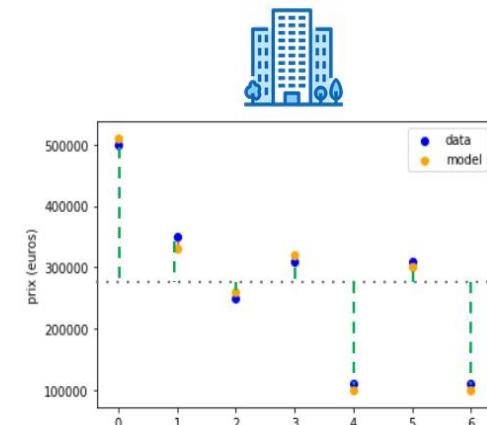
$$R^2 = 1 - \frac{\text{erreurs}}{\text{variance}}$$

erreurs << variance

$$R^2 = 1 - \frac{\sum(y_{vrai} - y_{pred})^2}{\sum(y_{vrai} - \bar{y}_{vrai})^2}$$

$$R^2 = 1 - 0.01$$

$R^2 = 0.99$



Sklearn: Métrique de Classification

<https://machinelearnia.com>

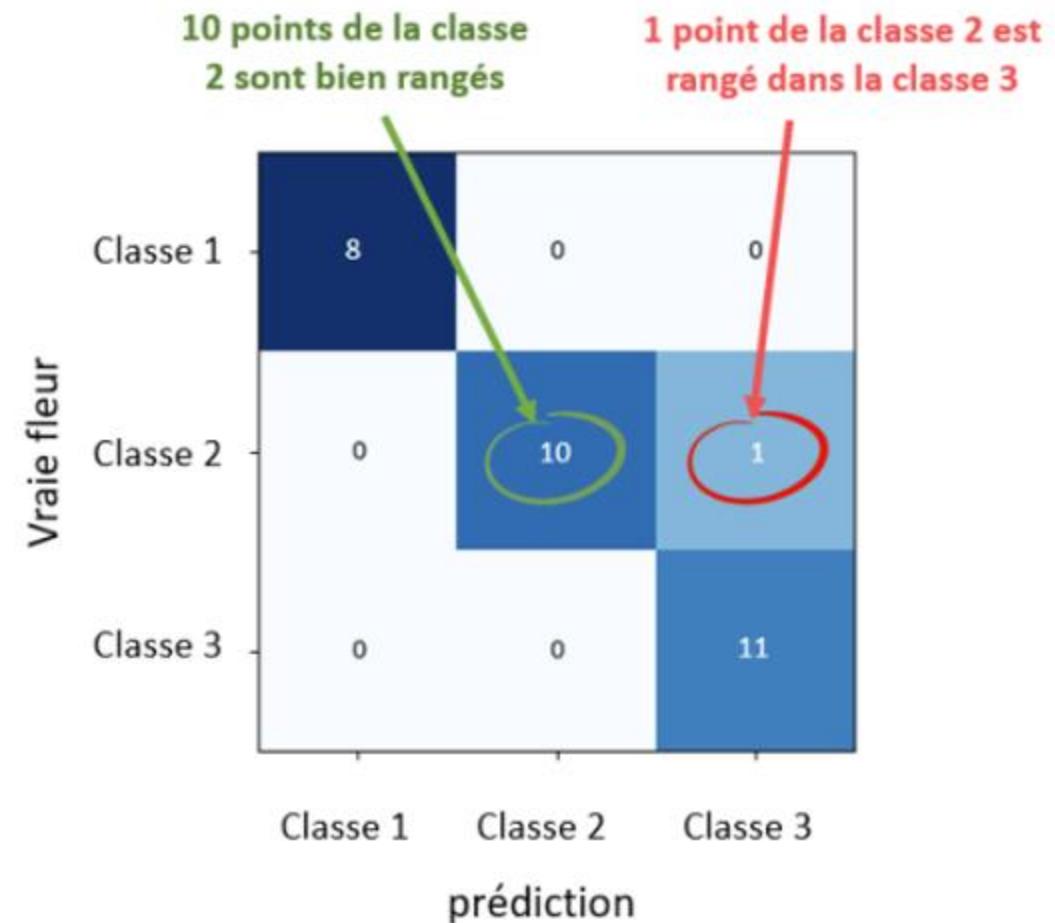
Par définition, une **matrice de confusion** C est telle que $C(i,j)$ est égale au nombre d'observations connues pour être dans le groupe i et prédites comme étant dans le groupe j.

C'est un outil de mesure très utile pour **évaluer** la qualité d'un **modèle de classification**, qui montre également les erreurs de classement

```
from sklearn.metrics import confusion_matrix
```

```
confusion_matrix(y_test, model.predict(X_test))
```

Retourne la matrice de confusion dont l'entrée i-ème ligne et j-ème colonne indique le nombre d'échantillons dont l'étiquette vraie est la i-ème classe et l'étiquette prédite la j-ème classe.



Formation Machine Learning

22. Encodage
et Normalisation

scikit
learn

```
preprocessing.Binarizer([threshold, copy])  
preprocessing.FunctionTransformer([func, ...])  
preprocessing.KBinsDiscretizer([n_bins, ...])  
preprocessing.KernelCenterer()  
preprocessing.LabelBinarizer([neg_label, ...])  
preprocessing.LabelEncoder  
preprocessing.MultiLabelBinarizer([classes, ...])  
preprocessing.MaxAbsScaler([copy])  
preprocessing.MinMaxScaler([feature_range, copy])  
preprocessing.Normalizer([norm, copy])  
preprocessing.OneHotEncoder([categories, ...])  
preprocessing.OrdinalEncoder([categories, dtype])  
preprocessing.PolynomialFeatures([degree, ...])  
preprocessing.PowerTransformer([method, ...])  
preprocessing.QuantileTransformer([...])  
preprocessing.RobustScaler([with_centering, ...])  
preprocessing.StandardScaler([copy, ...])
```

Encodage
Normalisation
Création de polynômes
Transformation non linéaire
Discrétilisation
Personnalisation

```
preprocessing.Binarizer([threshold, copy])  
preprocessing.FunctionTransformer([func, ...])  
preprocessing.KBinsDiscretizer([n_bins, ...])  
preprocessing.KernelCenterer()  
preprocessing.LabelBinarizer([neg_label, ...])  
preprocessing.LabelEncoder  
preprocessing.MultiLabelBinarizer([classes, ...])  
preprocessing.MaxAbsScaler([copy])  
preprocessing.MinMaxScaler([feature_range, copy])  
preprocessing.Normalizer([norm, copy])  
preprocessing.OneHotEncoder([categories, ...])  
preprocessing.OrdinalEncoder([categories, dtype])  
preprocessing.PolynomialFeatures([degree, ...])  
preprocessing.PowerTransformer([method, ...])  
preprocessing.QuantileTransformer([...])  
preprocessing.RobustScaler([with_centering, ...])  
preprocessing.StandardScaler([copy, ...])
```

preprocessing.Binarizer([threshold, copy])
preprocessing.FunctionTransformer([func, ...])
preprocessing.KBinsDiscretizer([n_bins, ...])
preprocessing.KernelCenterer()
preprocessing.LabelBinarizer([neg_label, ...])
preprocessing.LabelEncoder
preprocessing.MultiLabelBinarizer([classes, ...])
preprocessing.MaxAbsScaler([copy])
preprocessing.MinMaxScaler([feature_range, copy])
preprocessing.Normalizer([norm, copy])
preprocessing.OneHotEncoder([categories, ...])
preprocessing.OrdinalEncoder([categories, dtype])
preprocessing.PolynomialFeatures([degree, ...])
preprocessing.PowerTransformer([method, ...])
preprocessing.QuantileTransformer([...])
preprocessing.RobustScaler([with_centering, ...])
preprocessing.StandardScaler([copy, ...])

Encodage
Normalisation
Création de polynômes

Si vous
débutez

Sklearn: Transformers vs Estimators

<https://machinelearnia.com>

Un **Transformateur** transforme les données d'entrée (X) d'une certaine manière.

L'**estimateur** prédit une ou plusieurs nouvelle(s) valeur(s) y en utilisant les données d'entrée X

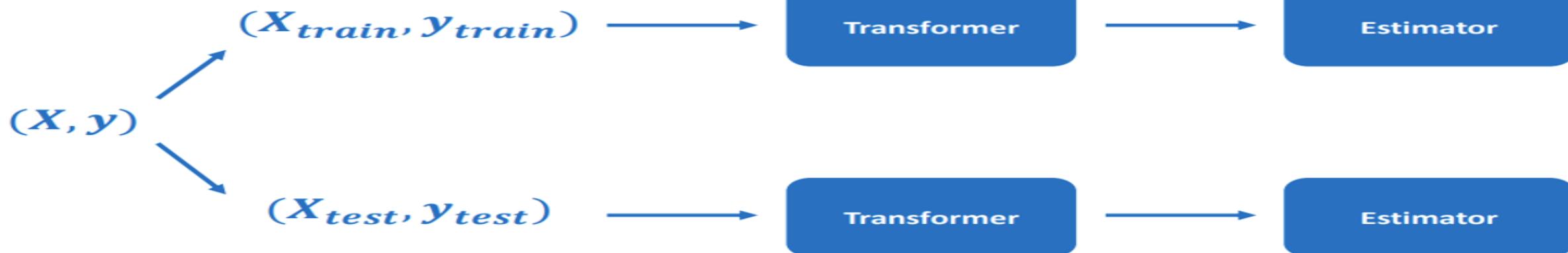
Le transformateur et l'estimateur doivent tous deux disposer d'une méthode fit() qui peut être utilisée pour les entraîner en apprenant certaines caractéristiques des données.

Remarque : fit() ne renvoie aucune valeur, mais stocke les données apprises dans l'objet.

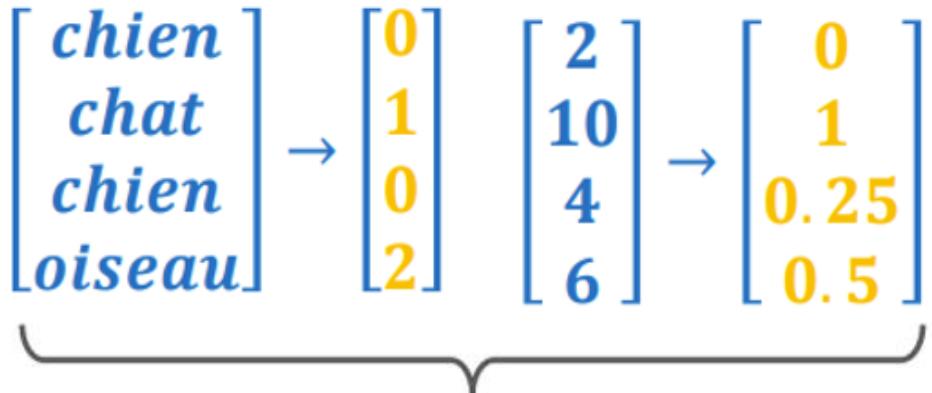
fit(X_{train}) : développe une fonction de **transformation** à partir de X_{train}

transform(X) : applique la **transformation** sur les données X_{train} , X_{test} et toutes autres données futures.

fit_transform(X_{train}) : développe la fonction de **transformation** puis l'utilise pour transformer X_{train}



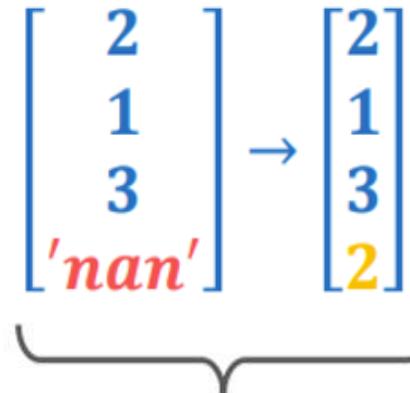
Encodage



sklearn.preprocessing

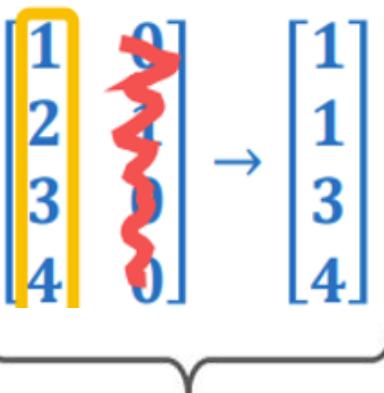
Normalisation

Imputation



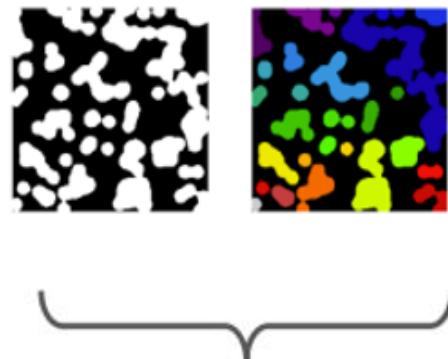
sklearn.impute

Sélection



sklearn
feature_selection

Extraction



sklearn
feature_extraction

On distingue 2 techniques pour encoder les données catégorielles, à appliquer avant de pouvoir ajuster et évaluer un modèle.

Encodage
Ordinal

LabelEncoder

Encodage
One-Hot

LabelBinarizer
(et MultiLabelBinarizer)

	y	X
OrdinalEncoder	LabelEncoder	OneHotEncoder

Variable nominale (catégorielle) : La variable comprend un ensemble fini de valeurs discrètes sans relation entre les valeurs.

Pour vos prédictions : `Inverse_transform`
Pour OneHotEncoder : `sparse=True`

Encodage Ordinal :

Associe chaque catégorie ou classe d'une variable à une valeur **décimale unique**.

<i>Chat</i>	-----	0
<i>Chat</i>	-----	0
<i>Chien</i>	-----	1
<i>Oiseau</i>	-----	2
<i>Chien</i>	-----	1

A utiliser pour certains cas :
`LabelEncoder()`
`OrdinalEncoder()`

Encodage One Hot :

Encode les catégories des **variables X** en valeurs **numériques** (0, $n_{\text{classe}}-1$)

Chat
Chat
Chien
Oiseau
Chien

<i>Chat</i>	-----	1	0	0
<i>Chat</i>	-----	1	0	0
<i>Chien</i>	-----	0	1	0
<i>Oiseau</i>	-----	0	0	1
<i>Chien</i>	-----	0	1	0

Sparse Matrix :

<i>a</i>	0	0	0	0
0	0	<i>b</i>	0	0
0	0	0	0	0
0	0	0	0	<i>c</i>
0	0	0	0	0

values = [*a, b, c*]

rows = [0, 1, 3]

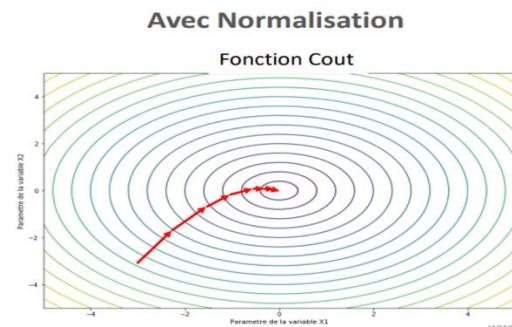
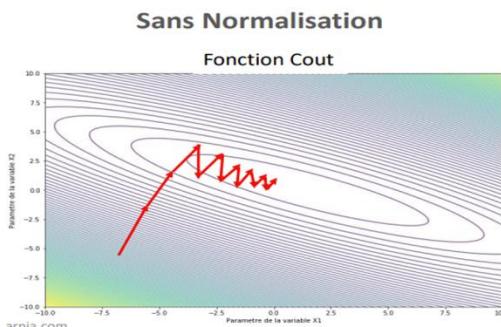
cols = [0, 2, 4]

A utiliser pour certains cas :
`LabelBinarizer()`
`MultiLabelBinarizer()`
`OneHotEncoder()`

Sklearn: Normalisation

<https://machinelearnia.com>

Avant d'entraîner des modèles d'apprentissage automatique sur des données, il est courant de **normaliser** les données afin d'obtenir des **Résultats plus rapides et de meilleure qualité**.

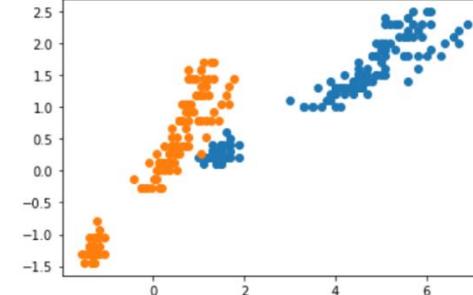


```
from sklearn.preprocessing import StandardScaler, MinMaxScaler,  
RobustScaler
```

StandardScaler : Standardise chaque variable X : La Moyenne est nulle et l'écart type égale 1.

$$\begin{bmatrix} 70 \\ 80 \\ 120 \end{bmatrix} \rightarrow \begin{bmatrix} -0.92 \\ -0.46 \\ 1.38 \end{bmatrix}$$

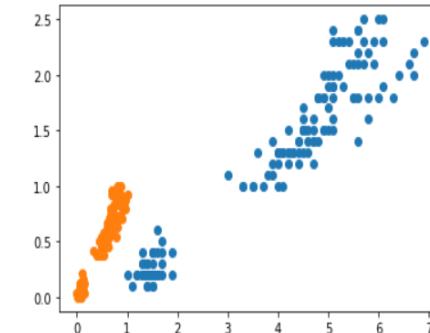
Utiliser `inverse_transform` pour revenir à l'échelle normale



MinMaxScaler : Transforme chaque variable X de telle sorte à être comprise entre 0 et 1.

$$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

$$\begin{bmatrix} 70 \\ 80 \\ 120 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ 0.2 \\ 1 \end{bmatrix}$$



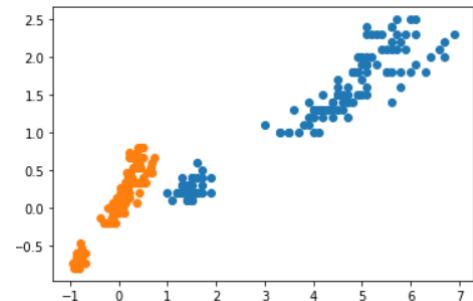
Utiliser `inverse_transform` pour revenir à l'échelle normale

RobustScaler : Transforme chaque variable X en étant peu sensible aux Outliers.

$$X_{scaled} = \frac{X - median}{IQR}$$



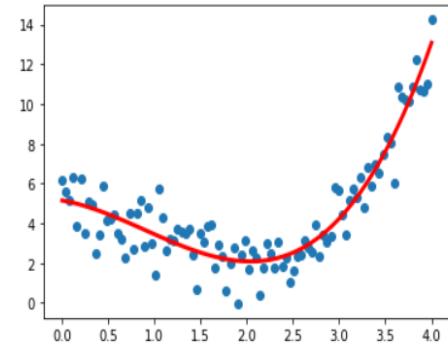
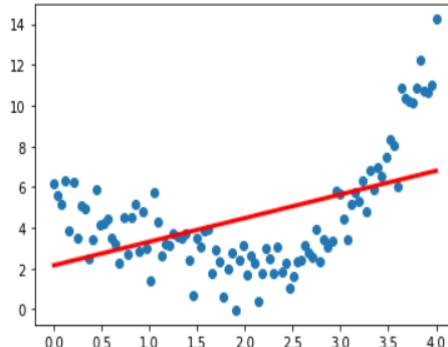
$$\begin{bmatrix} 70 \\ 80 \\ 120 \end{bmatrix} \rightarrow \begin{bmatrix} -0.4 \\ 0 \\ 1.6 \end{bmatrix}$$



Sklearn: Polynomial Feature

<https://machinelearnia.com>

Souvent, les caractéristiques d'entrée d'une tâche de modélisation prédictive interagissent de manière **inattendue et non linéaire**. On utilise donc une **forme polynomiale** à base de notre modèle.



Exemple : 1 variable $x \rightarrow \text{Polynôme 2}$

$$\begin{aligned} x &\rightarrow 1 \quad x^1 \quad x^2 \\ \begin{bmatrix} 1 \\ 2 \\ 0.5 \end{bmatrix} &\rightarrow \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 0.5 & 0.25 \end{bmatrix} \end{aligned}$$

La machine apprend un modèle :
 $f(x) = ax^2 + bx + c$

```
from sklearn.preprocessing import PolynomialFeatures  
from sklearn.linear_model import LinearRegression
```

Exemple : 2 variables $x_1, x_2 \rightarrow \text{Polynôme 2}$

$x_1, x_2 \rightarrow 1, x_1, x_2, x_1^2, x_1x_2, x_2^2$

Ces variables n'étant **pas sur la même échelle**, il faut les **normaliser** avant leur passage dans l'estimateur !

Crée de nouvelles variables polynomiales à partir des variables existantes :

```
X_Poly = PolynomialFeatures(3).fit_transform(X)  
model = LinearRegression().fit(X_poly, y)  
y_pred = model.predict(X_poly)
```

SimpleImputer :

Remplace toutes les valeurs manquantes par une valeur statistique. **transform(Xtest)** remplace les valeurs manquantes par les moyennes calculées sur Xtrain (principe du TRANSFORMER).

$$X_{test} = \begin{bmatrix} 12 & 5 \\ 40 & 2 \\ 5 & 5 \\ nan & nan \end{bmatrix} \rightarrow \begin{bmatrix} 12 & 5 \\ 40 & 2 \\ 5 & 5 \\ 5 & 3.25 \end{bmatrix}$$

ATTENTION : Ne pas calculer la moyenne de Xtrain et Xtest ensemble pour éviter une fuite d'information

Solution :

```
imputer = SimpleImputer(missing_values = nan, strategy= 'mean')
imputer.fit_transform(Xtest)
```

Paramètres : mean, median, most_frequent, constant, fill_value

```
from sklearn.impute import SimpleImputer
```

MissingIndicator :

Variable booléenne qui indique l'absence de valeurs dans le dataset

Parfois le manque d'information représente une information !

$$\begin{bmatrix} Classe \\ passager \end{bmatrix} \rightarrow \begin{bmatrix} Membre \\ équipe \end{bmatrix}$$

$$\begin{bmatrix} 2 \\ 3 \\ 3 \\ nan \end{bmatrix} \rightarrow \begin{bmatrix} False \\ False \\ False \\ True \end{bmatrix}$$

Solution : MissingIndicator().fit_transform(X)

KNNImputer :

Remplace toutes les valeurs manquantes par les valeurs des plus proches voisins

1. **fit(X_train)** garde en mémoire toutes les données X_{train}
2. **transform(X)** remplace les valeurs manquantes selon ce qu'il a gardé en mémoire



Solution :

```
imputer = KNNImputer(n_neighbors=1)
imputer.fit_transform(X)
```

```
from sklearn.impute import KNNImputer
```

```
from sklearn.impute import MissingIndicator
from sklearn.pipeline import make_union
```

$$\begin{bmatrix} Classe \\ passager \end{bmatrix} \rightarrow \begin{bmatrix} Classe \\ passager & Membre \\ équipe \end{bmatrix}$$

$$\begin{bmatrix} 2 \\ 3 \\ 3 \\ nan \end{bmatrix} \rightarrow \begin{bmatrix} 2 & 0 \\ 3 & 0 \\ 3 & 0 \\ -99 & 1 \end{bmatrix}$$

Application :
Utiliser avec la fonction
make_union +
OneHotEncoder

Sklearn: Pipelines

<https://machinelearnia.com>

Pour construire un estimateur composite, les transformateurs sont généralement combinés avec d'autres transformateurs prédicteurs. On utilise alors un **pipeline**. Il exige que toutes les étapes, à l'exception de la dernière, soient des **transformateurs**.

```
from sklearn.pipeline import make_pipeline  
from sklearn.linear_model import SGDClassifier  
from sklearn.model_selection import GridSearchCV  
from sklearn.compose import make_column_selector,  
    make_column_transformer
```

Pour créer un pipeline, 2 options :

- **Classe Pipeline**
- Fonction **make_pipeline**

```
model = make_pipeline(StandardScaler(), SGDClassifier())  
model.fit(Xtrain,ytrain)  
model.score(Xtest, ytest)  
model.predict(Xtest)
```

GridSearchCV :

Utile pour trouver les meilleurs paramètres du pipeline.

```
grid = GridSearchCV(pipeline, params, cv)
```

```
Params = { <Composant>_<paramètre> : [...] }
```

```
grid.fit(Xtrain, ytrain) grid.best_estimator
```

ColumnTransformer :

Permet d'appliquer vos transformers sur les colonnes :

```
make_column_transformer((transformer, ['colonne 1', 'colonne  
2', ...]))
```

Column Selector :

Depuis sklearn 0.22, **make_column_selector** permet de sélectionner certains types de colonnes

3 arguments au choix :

dtype_include

dtype_exclude

pattern (avec des Regular Expressions)

Exemple de types :

np.number → variables numériques

object → variables catégorielles

Pipeline

Column_transformer

Numerical_pipeline

SimpleImputer

StandardScaler

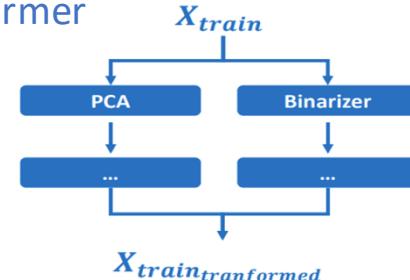
Categorical_pipeline

SimpleImputer

OneHotEncoder

Feature Union :

make_union() permet de créer des pipelines parallèles. Les résultats sont concaténés à la sortie du transformer



SGDClassifier

Formation Machine Learning

23. Sélection de Variables

scikit
learn

Sklearn: Selection des variables

<https://machinelearnia.com>

VarianceThreshold élimine les variables dont la **variance** est inférieure à un certain seuil.

```
from sklearn.feature_selection import VarianceThreshold
```

Explication :

Pour effectuer des prédictions, un estimateur a besoin d'informations qui **varient** en accord avec la **Target**

Ville	Surface	Prix
Paris	50	600,000
Paris	40	450,000
Paris	45	550,000
Paris	65	700,000

« A quoi sert une variable qui ne varie pas ? »

SelectKBest sélectionne les K variables X dont le score du test de dépendance avec y est le plus élevé

```
from sklearn.feature_selection import SelectKBest, chi2, f_classif
```

Solution :

```
selector = SelectKBest(f_classif, k=2)  
selector.fit(X,y)
```

RecursiveFeatureElimination classe les caractéristiques avec éliminations récursives de celles-ci à l'aide d'un estimateur externe qui attribue des poids aux caractéristiques.

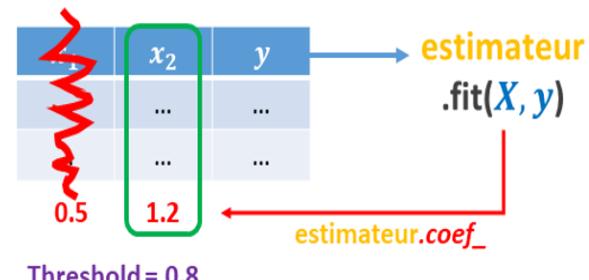
```
from sklearn.feature_selection import RFECV
```

Solution :

```
selector = RFECV(SGDClassifier(random_state=0),  
                  step=1,  
                  min_features_to_select=2,  
                  cv=5)  
  
selector.fit(X,y)
```

SelectFromModel entraîne un estimateur puis sélectionne les variables les plus importantes pour cet estimateur

```
from sklearn.feature_selection import SelectFromModel
```



Note : Compatible avec les estimateurs qui développent une **fonction paramétrée** (attribut **.coef_** ou **.feature_importance_**)
⇒ **K-Nearest Neighbour** incompatible

Formation Machine Learning

24. Apprentissage
Non Supervisé

The Scikit-learn logo is located in the bottom right corner. It consists of two overlapping circles: a blue circle on the left and an orange circle on the right. The word "scikit" is written in a small, white, sans-serif font above the word "learn". The word "learn" is written in a large, black, cursive font.

scikit
learn

Sklearn: Apprentissage Non-Supervisé

<https://machinelearnia.com>

L'apprentissage **non supervisé** est une catégorie de techniques d'apprentissage machine (ML) utilisées pour trouver des modèles dans les données.

Dans l'apprentissage non supervisé, les algorithmes sont laissés à eux-mêmes pour découvrir des structures intéressantes dans les données : Les variables d'entrées (X) sont données **sans variables de sorties** correspondantes. Cette technique s'utilise pour gagner en adaptabilité, en scalabilité, et pour l'exploration.

Apprentissage Supervisé

On montre à la machine des exemples (X, y) de ce qu'elle doit apprendre



Apprentissage Non-Supervisé

La machine analyse **la structure** des données X pour apprendre elle-même à réaliser certaines tâches...



On distingue 3 techniques pour découvrir des structures ou motifs dans les données :

K-MEANS clustering : Permet de partitionner un ensemble de données en un nombre fixe de clusters. Chaque point est attribué au cluster dont le centre appelé centroïde est le plus proche.

Détection d'anomalies - Isolation Forest : Permet de détecter les anomalies dans un jeu de données. On isole les observations en les coupant de manière récursive dans des sous-ensembles aléatoires de données.

Réduction de Dimension : Vise à réduire le nombre de variables d'entrée dans un jeu de données tout en préservant autant que possible l'information importante. Utile avec un grand nombre de caractéristiques rendant l'apprentissage difficile ou inefficace.

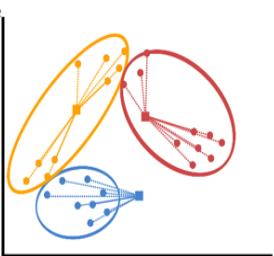
```
from sklearn.cluster import Kmeans  
from sklearn.ensemble import IsolationForest  
from sklearn.decomposition import PCA
```

Sklearn: K-Means Clustering (1)

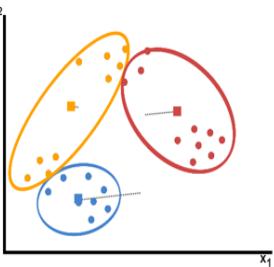
<https://machinelearnia.com>

K-Means est un algorithme itératif qui fonctionne en 2 étapes :

1. Affectation des points au centre le plus proche



2. Déplacement du centre
A la moyenne du cluster



Pour réaliser cette opération, la machine apprend à projeter du mieux possible nos données dans des espaces de plus petite dimension.

Reduction de dimension : En étudiant la structure de nos données, la machine apprend à la simplifier en conservant les principales informations

Attention :
Selon la position initiale des centroïdes, K-Means peut donner de mauvais clusters.

Hyper-Paramètres :

n_clusters : nombre K de clusters

n_init : nombre d'exécutions (10)

max_iter : nombre d'itérations (300)

Init : type d'initialisation (**k-means++**)

Attributs :

cluster_centers_ : position des centroids

N_iter_ : Nombre d'itérations

Labels_ : équivalent de Predict(X_{train})

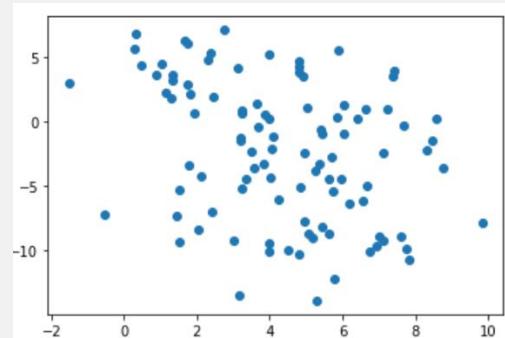
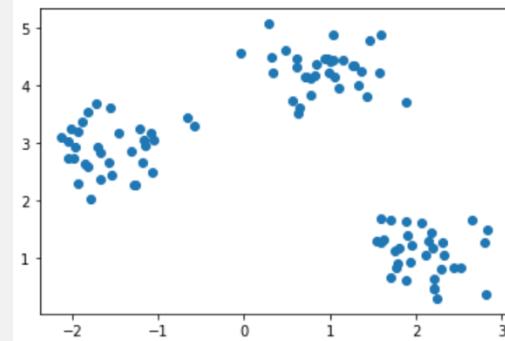
Inertia_ : calcul de l'inertie (positif)

Méthodes :

Fit(X) : exécute l'algorithme K-Means

Predict(X) : centroid le plus proche de X

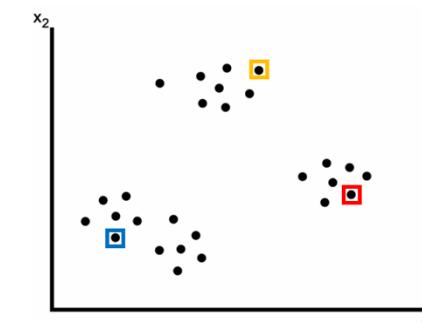
Score(X) : calcul de l'inertie (négatif)



Dans cet exemple :
Facile de définir n_cluster (on peut voir 3 clusters)

Mais dans la pratique :
Datasets avec de nombreuses dimensions...
→ Difficile de voir un nombre de clusters à l'œil.

Comment déterminer n_clusters ?



K-Means++ est une méthode d'initialisation qui consiste à placer les centroids sur des points du dataset éloignés les uns des autres.

→ Cela facilite la convergence !

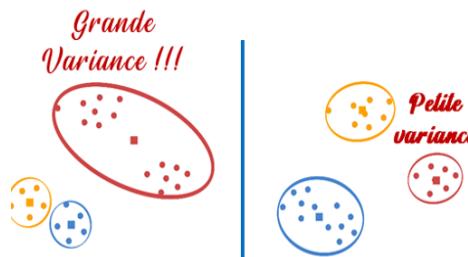
Sklearn: K-Means Clustering (2)

<https://machinelearnia.com>

K-Mean cherche la position des centres qui minimise la distance entre les points d'un cluster (x_i) et le centre (μ_j) de ce dernier:

$$\sum_{i=0}^n \min_{\mu_j \in C} (\|x_i - \mu_j\|^2)$$

Note : Cela équivaut à minimiser la variance des clusters.



Pour trouver le bon nombre de clusters :

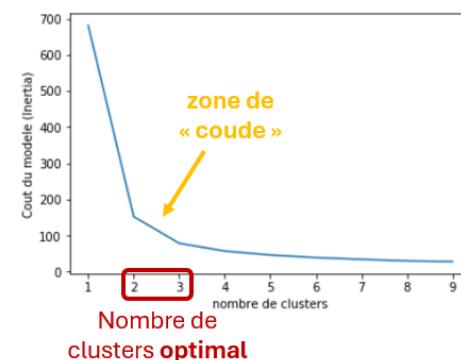
- Prédéterminer le nombre
- Examiner les résultats du clustering
- « Elbow Method » : Détecter une zone de « coude » dans la minimisation du cout

Solution :

```
inertia = []
K_range = range(1, 20)
```

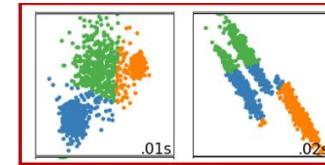
For k in K_range:

```
    model = Kmeans(n_clusters=k).fit(X)
    inertia.append(model.inertia)
```

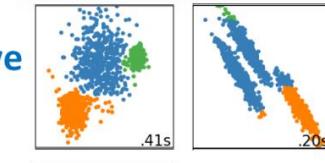


Autres algorithmes de Clustering :

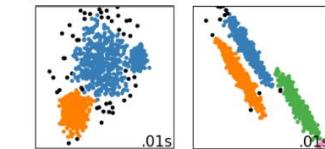
K-Means



Agglomerative Clustering



DBSCAN



Sur des clusters non convexes et anisotropes, K-Means donne de pauvres résultats...

Le module **sklearn.cluster** regroupe les algorithmes de clustering non supervisés les plus répandus.

Classes

<code>cluster.AffinityPropagation([damping, ...])</code>	Perf	AgglomerativeClustering : (hierarchical Clustering)
<code>cluster.AgglomerativeClustering([...])</code>	Agglomerative Clustering	
<code>cluster.Birch([threshold, branching_factor, ...])</code>	Implements the Birch clustering algorithm.	
<code>cluster.DBSCAN([eps, min_samples, metric, ...])</code>	Perform DBSCAN clustering from vector array or distance matrix.	
<code>cluster.FeatureAgglomeration([n_clusters, ...])</code>	Agglomerate features.	
<code>cluster.KMeans([n_clusters, init, n_init, ...])</code>	K-Means	DBSCAN : Density-based spatial clustering of applications with noise
<code>cluster.MiniBatchKMeans([n_clusters, init, ...])</code>	Mini-Batch K-Means	
<code>cluster.MeanShift([bandwidth, seeds, ...])</code>	Mean shift clustering using a flat kernel.	
<code>cluster.OPTICS([min_samples, max_eps, ...])</code>	Estimate clustering structure from vector array.	
<code>cluster.SpectralClustering([n_clusters, ...])</code>	Apply clustering to a projection of the normalized Laplacian.	
<code>cluster.SpectralBiclustering([n_clusters, ...])</code>	Spectral biclustering	
<code>cluster.SpectralCoclustering([n_clusters, ...])</code>	Spectral co-clustering	
		Et Spectral Clustering aussi, super utile !

Sklearn: Isolation Forest (1)

<https://machinelearnia.com>

Isolation Forest : Cette technique permet d'isoler les anomalies car elles sont rares et différentes des autres points. On peut construire une forêt de plusieurs arbres en divisant **récursivement** les données jusqu'à ce que chaque **point** de données soit **isolé**.

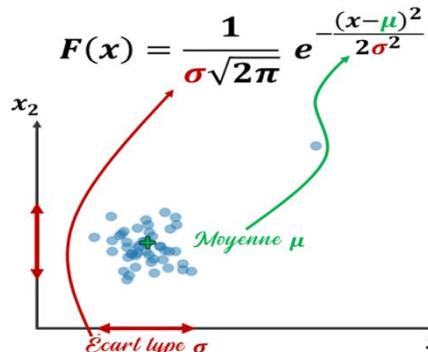
IsolationForest(contamination=0.02) Pourcentage d'anomalies estimées

`fit(X) → entraîne le modèle`

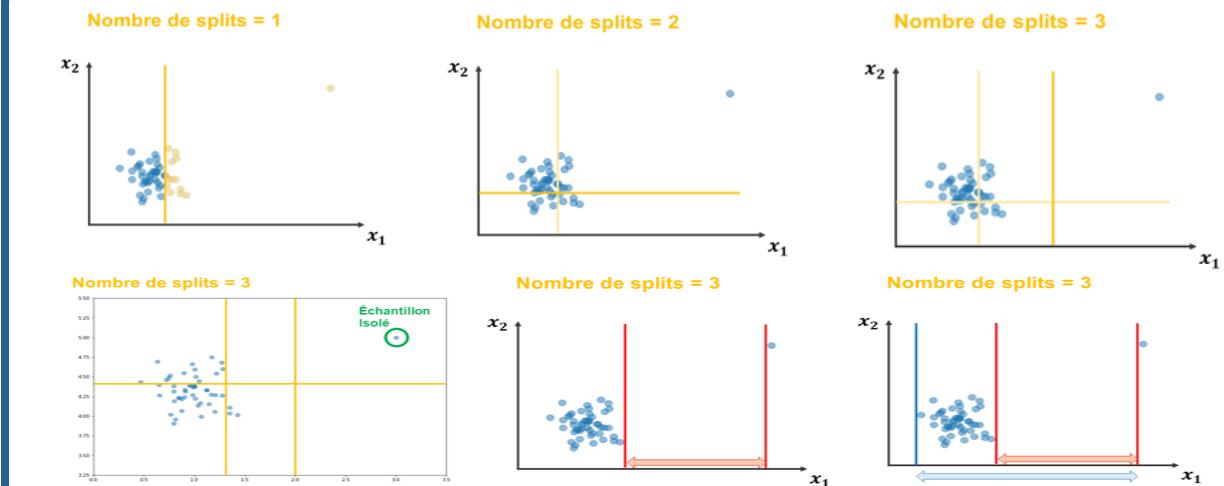
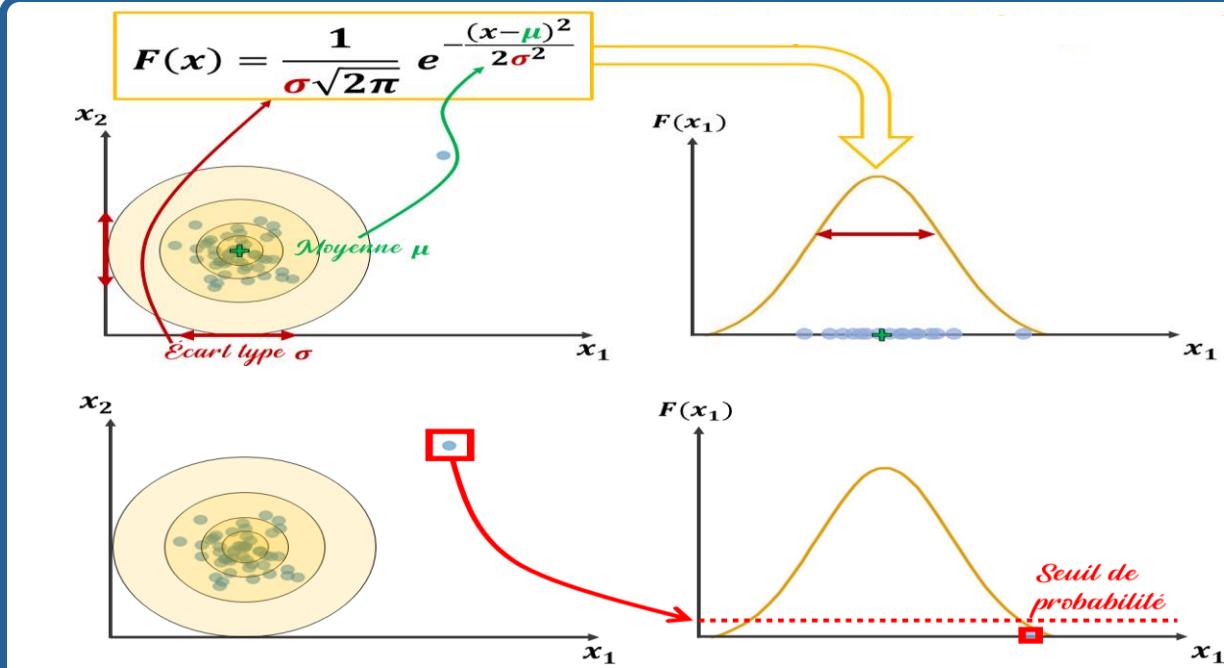
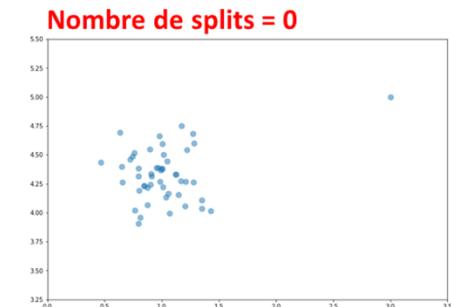
`predict(X) → { +1 : normale
-1 : anomalie }`

`decision_function →`

On effectue une série de **splits** aléatoires,
Et on **compte** le nombre de splits qu'il faut effectuer pour pouvoir isoler nos échantillons.

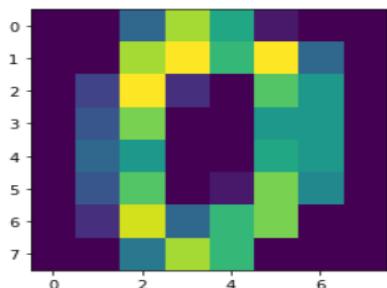
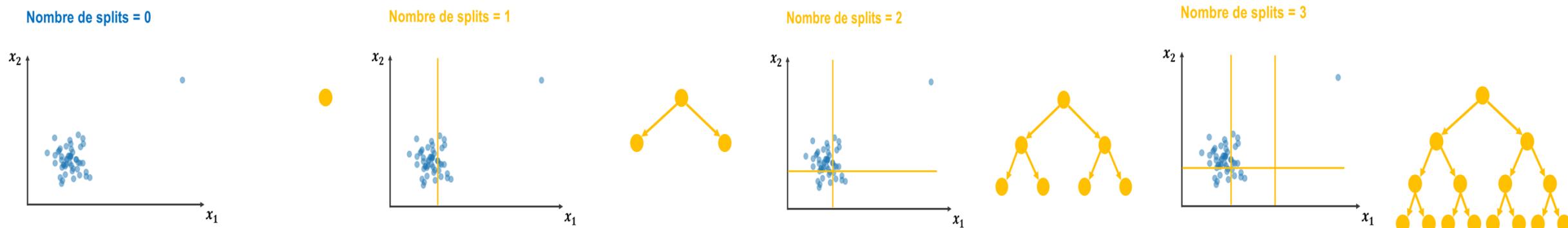
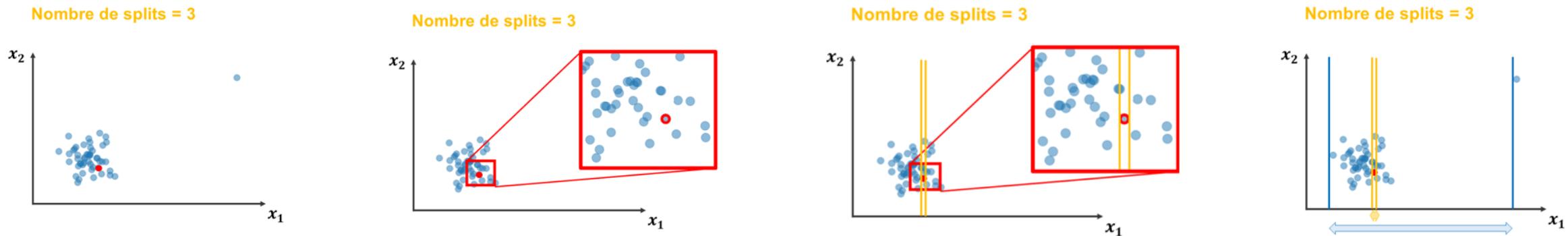


Densité de probabilité (loi Normale)



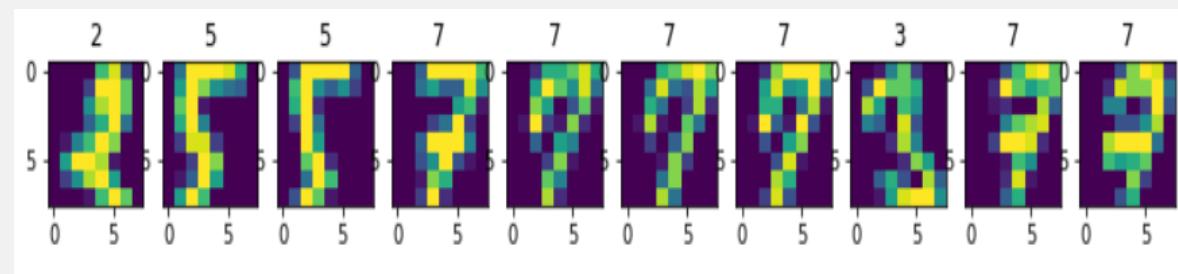
Sklearn: Isolation Forest (2)

<https://machinelearnia.com>



Solution :

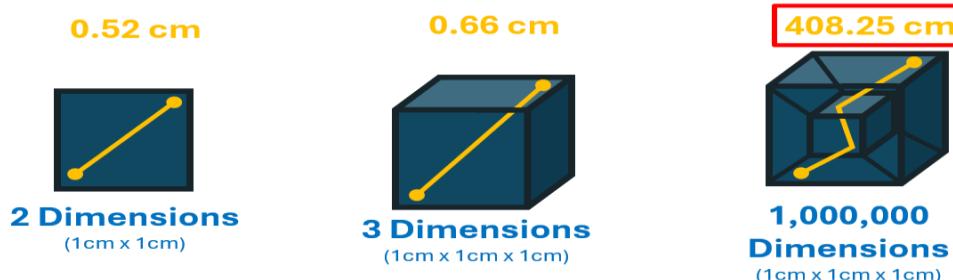
```
model = IsolationForest(random_state = 0,  
                        contamination = 0.02)  
  
model.fit(X)  
outliers = model.predict(X) == -1
```



Sklearn: Analyse en Composante Principale

<https://machinelearnia.com>

Lorsque la dimension du Dataset augmente, la **distance moyenne** entre 2 points aussi. Cela peut poser un problème :



Solution : on transforme les variables d'origine en un nouveau jeu de variables non corrélées (**composante principales**), en les ordonnant de manière à capturer la plus grande variance possible.

Il faut **Standardiser** les données avant d'utiliser PCA.
PCA est normalement conçu pour les **variables continues**.
PCA n'est **pas efficace** sur les Dataset **non-linéaires**.

Pour trouver **les axes** de projection (**xpc**) :

1. On calcule la **matrice de covariance** des données
2. On détermine les **vecteurs propres** de cette matrice : ce sont les **Composantes Principales**
3. On **projette** les données sur ces **axes**

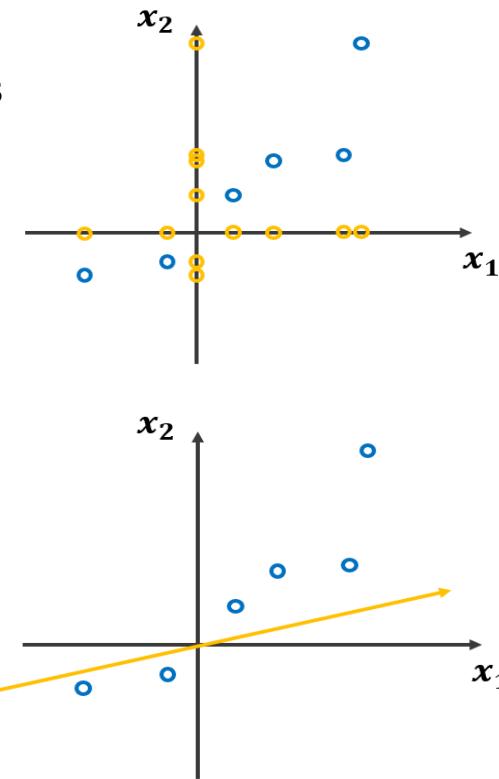
PCA est un **transformer** !

1. Définir le nombre de **composantes**
2. Transformer les données avec **fit_transform()**

Solution :

```
model = PCA(n_components=2)  
model.fit(X)
```

```
PCA(copy=True,  
iterated_power ='auto',  
n_components=2,  
random_state=None,  
svd_solver='auto',  
tol=0.0,  
whiten=False)
```

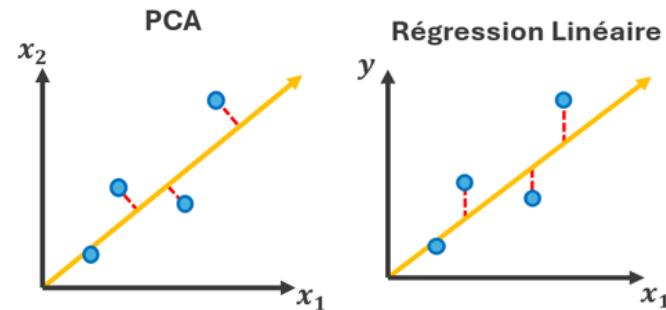


ATTENTION :

PCA \neq Régression linéaire

Pour trouver **l'axe de projection optimale**, il faut :

1. **Calculer la matrice de covariance des données**
2. **Déterminer les vecteurs propres de cette matrice**



Formation Machine Learning

25. Ensemble Learning

The Scikit-learn logo is located in the bottom right corner. It consists of the word "scikit" in a small, sans-serif font above the word "learn" in a larger, stylized, italicized font. The entire logo is contained within a yellow circular graphic.

scikit
learn

```
ensemble.AdaBoostClassifier([...])  
ensemble.AdaBoostRegressor([base_estimator, ...])  
ensemble.BaggingClassifier([base_estimator, ...])  
ensemble.BaggingRegressor([base_estimator, ...])  
ensemble.ExtraTreesClassifier([...])  
ensemble.ExtraTreesRegressor([n_estimators, ...])  
ensemble.GradientBoostingClassifier([loss, ...])  
ensemble.GradientBoostingRegressor([loss, ...])  
ensemble.IsolationForest([n_estimators, ...])  
ensemble.RandomForestClassifier([...])  
ensemble.RandomForestRegressor([...])  
ensemble.RandomTreesEmbedding([...])  
ensemble.StackingClassifier(estimators[, ...])  
ensemble.StackingRegressor(estimators[, ...])  
ensemble.VotingClassifier(estimators[, ...])  
ensemble.VotingRegressor(estimators[, ...])  
ensemble.HistGradientBoostingRegressor([...])  
ensemble.HistGradientBoostingClassifier([...])
```

Boosting estimator

Bagging estimator

Stacking estimator

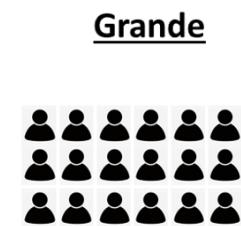
Voting estimator

Sklearn: Wisdom of the Crowd

<https://machinelearnia.com>

On peut regrouper les prédictions d'un groupe de prédicteurs (tels que des classificateurs ou des régresseurs), ce qui produit souvent de meilleures prédictions qu'avec un prédicteur individuel.

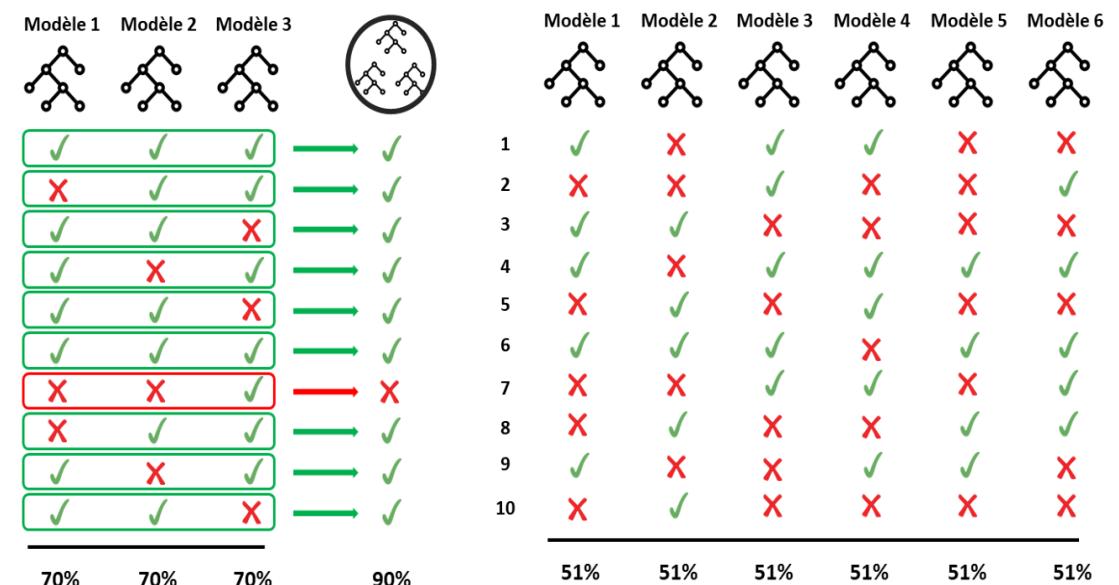
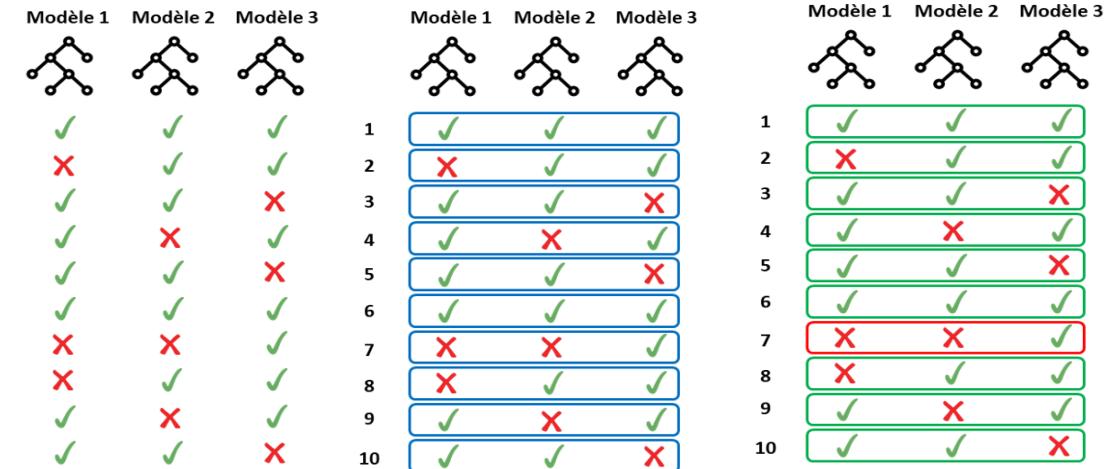
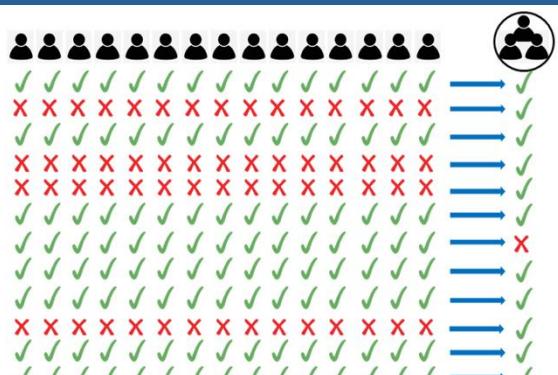
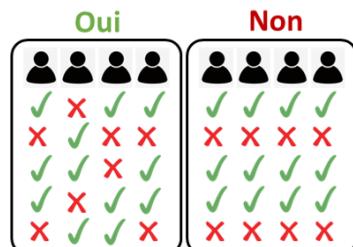
Une foule d'individu a plus souvent raison qu'un expert tout seul, à condition que la foule soit :



→ +50%

Compétente

Diversifiée

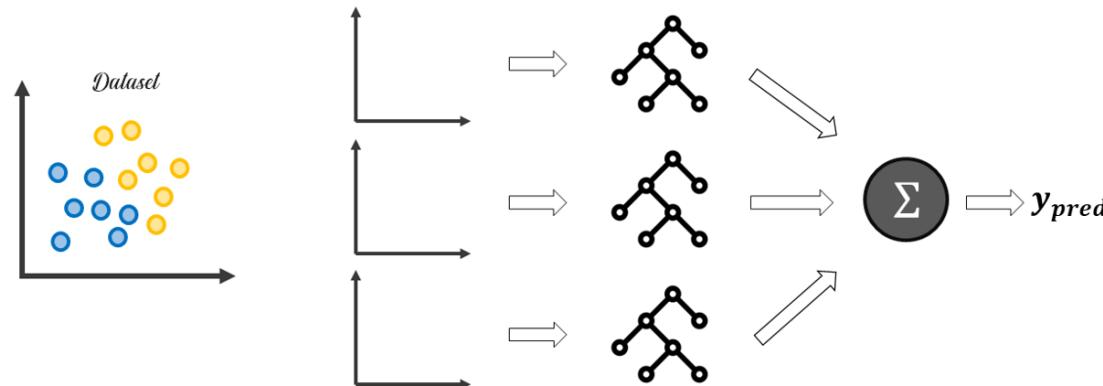


Sklearn: Bagging and Boosting

<https://machinelearnia.com>

Bagging : Crée plusieurs copies d'un même modèle, en entrainant chaque copie sur une partie aléatoire du dataset.

```
from sklearn.ensemble import BaggingClassifier
```



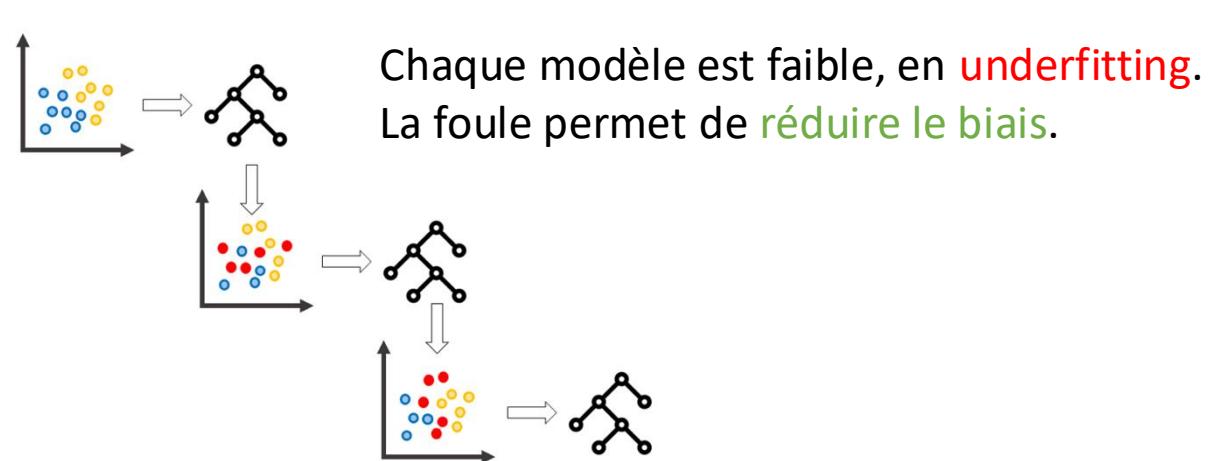
Chaque modèle est fort mais **overfit** son sub-set.
La foule permet de **réduire la variance**

Solution :

```
model=BaggingClassifier(base_estimator=KNeighborsClassifier())
model.fit(X_train, y_train)
model.score(X_test, y_test)
```

Boosting : Crée plusieurs copies d'un même modèle, en entrainant chaque copie sur une partie aléatoire du dataset.

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
```



Solution :

```
model = AdaBoostClassifier(n_estimators =100)
model.fit(X_train, y_train)
model.score(X_test, y_test)
```

Sklearn: Voting and Stacking

<https://machinelearnia.com>

VotingClassifier est utilisée pour les problèmes de classification. Elle prend en compte une liste d'estimateurs et un paramètre de vote, qui peut être défini comme dur ou doux.

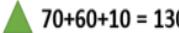
```
from sklearn.ensemble import VotingClassifier  
+ des modèles pour réaliser le vote
```

VotingRegressor : La prédition finale correspond à la **moyenne** des prédictions.



1

VotingClassifier :

- **Hard Voting** : vote sur les prédictions.  $70+60+10 = 130$
- **Soft Voting** : vote sur les probabilités de chaque classe. 

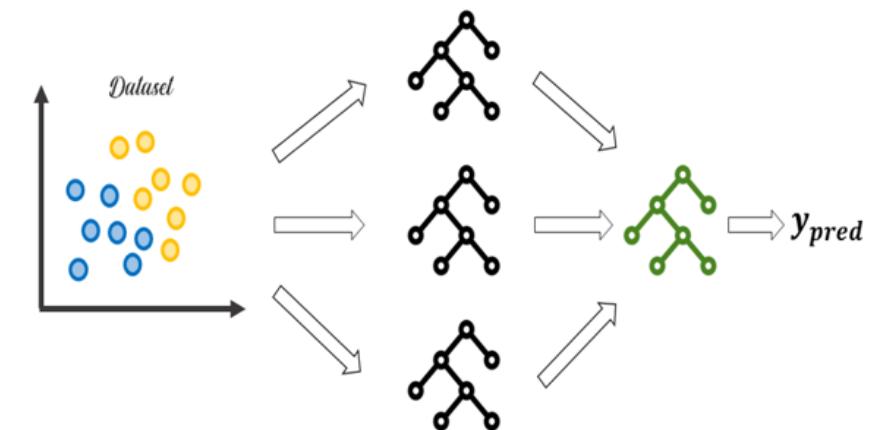
70%	30%
60%	40%
10%	90%

Solution :

```
model_1 = SGDClassifier(random_state=0)  
model_2 = DecisionTreeClassifier(random_state=0)  
model_3 = KNeighborsClassifier(n_neighbors=2)  
model_4 = VotingClassifier([('SGD', model_1),  
                           ('Tree', model_2),  
                           ('KNN', model_3)],  
                           voting = 'hard')
```

Stacking : Au lieu de rassembler les prédictions de chaque modèle... On demande à un dernier estimateur d'apprendre à prédire le résultat final en fonction de ces prédictions

```
from sklearn.ensemble import StackingClassifier
```



Solution :

```
model = StackingClassifier([('SGD', model_1),  
                           ('Tree', model_2),  
                           ('KNN', model_3)],  
                           final_estimator = KNeighborsClassifier())  
  
model.fit(X_train, y_train)  
model.score(X_test, y_test)
```