

# Python for machine learning

Ph.D. HOSNI Adil Imad Eddine

September 13, 2022



# Contents

<b>1 Bases de Python</b>	<b>7</b>
1.1 Variables et Fonctions . . . . .	7
1.1.1 Variables et Operations . . . . .	7
1.1.2 Fonctions . . . . .	8
1.1.3 Exercice . . . . .	9
1.2 Structures de Controle . . . . .	9
1.2.1 Instructions conditionnelles : . . . . .	10
1.2.2 Instructions itératives For, while . . . . .	10
1.2.3 Exercices . . . . .	12
1.3 Structure de données (Listes et Tuples) . . . . .	13
1.3.1 Création de Listes et de Tuples . . . . .	13
1.3.2 Indexing et Slicing . . . . .	13
1.3.3 Actions utiles sur les listes . . . . .	14
1.3.4 Exercice . . . . .	16
1.4 Dictionnaires . . . . .	16
1.4.1 Définition . . . . .	16
1.4.2 fonction de manipulation . . . . .	17
1.4.3 List Comprehension . . . . .	18
1.4.4 Exercice . . . . .	19
1.5 Built-in Functions . . . . .	20
1.5.1 Fonction de bases . . . . .	20
1.5.2 Fonction de conversion . . . . .	20
1.5.3 Fonction <b>input()</b> . . . . .	21
1.5.4 Fonction <b>format()</b> . . . . .	22
1.5.5 Fonction <b>open()</b> . . . . .	22
1.5.6 Exercice . . . . .	23
1.6 Modules de bases de Python . . . . .	23
1.6.1 Modules math et statistics . . . . .	23
1.6.2 Module Random . . . . .	24
1.6.3 Modules OS et Glob . . . . .	25
1.6.4 Créer notre propre module . . . . .	25
1.7 Programmation Orientée Objet avec Python . . . . .	25
1.7.1 Créer des classes . . . . .	25
1.7.2 Créer des sous-classes . . . . .	26
1.8 Solutions aux exercices . . . . .	27
1.8.1 Variables et Fonctions . . . . .	27

1.8.2	Structures de Contrôle . . . . .	28
1.8.3	Structure de données (Listes et Tuples) . . . . .	28
1.8.4	Dictionnaires . . . . .	28
1.8.5	Built-in Functions . . . . .	29
<b>2</b>	<b>Modules scientifiques</b>	<b>31</b>
2.1	Numpy : . . . . .	31
2.1.1	Fonctions de base . . . . .	31
2.1.2	Slicing et Indexing . . . . .	35
2.1.3	Mathématiques . . . . .	37
2.1.4	Solutions aux exercices . . . . .	42
2.2	Matplotlib . . . . .	45
2.2.1	Pyplot . . . . .	45
2.2.2	Graphiques simples . . . . .	45
2.2.3	Styles Graphiques . . . . .	46
2.2.4	Cycle de vie d'une figure . . . . .	47
2.2.5	Subplots . . . . .	48
2.2.6	Méthode orientée objet . . . . .	49
2.2.7	Matplotlib Graphiques importants . . . . .	50
2.2.8	Exercices . . . . .	57
2.2.9	Solution au exercices . . . . .	58
2.3	Scipy . . . . .	61
2.3.1	Interpolation . . . . .	61
2.3.2	Optimisation . . . . .	62
2.3.3	Traitemet du signal . . . . .	66
2.3.4	Transformation de Fourier (FFT) . . . . .	67
2.3.5	Traitemet d'image . . . . .	69
2.3.6	Application (cas réel) . . . . .	70
<b>3</b>	<b>Analyse de données et visualisation</b>	<b>75</b>
3.1	Pandas . . . . .	75
3.1.1	DataFrame Pandas . . . . .	75
3.1.2	Statistics avec Groupby() et value_counts() . . . . .	78
3.1.3	Opération sur les série . . . . .	80
3.1.4	Méthodes loc et iloc . . . . .	81
3.1.5	Codefication des données . . . . .	81
3.1.6	Séries temporelles . . . . .	84
3.1.7	Moving Average et EWM . . . . .	90
3.1.8	Exercice et Solution . . . . .	93
3.2	Seaborn . . . . .	94
3.2.1	La vue d'ensemble Pairplot() . . . . .	95
3.2.2	Visualiser de catégories . . . . .	97
3.2.3	Visualisation de Distributions . . . . .	99
<b>4</b>	<b>Machine learning avec Sklearn</b>	<b>101</b>
4.1	Introduction sur Sklearn . . . . .	101
4.1.1	Régression . . . . .	101
4.1.2	Classification . . . . .	105

4.1.3	Exercice et Solution . . . . .	106
4.2	Sélection de Modele . . . . .	107
4.2.1	Train Test Split . . . . .	108
4.2.2	Validation Set et Cross Validation . . . . .	109
4.2.3	Validation Curve . . . . .	111
4.2.4	GridSearchCV . . . . .	112
4.2.5	Learning Curve . . . . .	113
4.2.6	Metrics d'évaluation . . . . .	114
4.2.7	Exercice et solution . . . . .	116
4.3	Pre-processing . . . . .	118
4.3.1	Encodage . . . . .	118
4.3.2	Normalisation . . . . .	120
4.3.3	Polynomial Features . . . . .	124
4.3.4	Discretisation . . . . .	126
4.4	Pipelines . . . . .	127
4.4.1	Cree une pipeline . . . . .	127
4.4.2	Imputers . . . . .	128
4.4.3	Make Column transformer . . . . .	129
4.4.4	Make column selector . . . . .	131
4.4.5	Exercise . . . . .	132
4.5	Feature Selection . . . . .	135
4.5.1	Variance Threshold . . . . .	135
4.5.2	SelectKBest . . . . .	136
4.5.3	Recursive feature Elimination . . . . .	136
4.5.4	SelectFromModel . . . . .	137
4.5.5	Exercice . . . . .	137
4.6	Apprentissage Non-Supervisé . . . . .	139
4.6.1	Clustering . . . . .	139
4.6.2	Detection d'anomalies avec Isolation Forest . . . . .	145
4.6.3	Reduction de dimension . . . . .	149
<b>5</b>	<b>Demarches pour un projet de science de donnees</b>	<b>155</b>
5.1	Exploratory Data Analysis . . . . .	155
5.1.1	Objectif : . . . . .	155
5.1.2	Analyse de la forme des données . . . . .	155
5.1.3	Analyse du Fond . . . . .	159
5.1.4	Relation Target / Variables . . . . .	165
5.1.5	Analyse Avancée . . . . .	169
5.1.6	Test des hypotheses . . . . .	180
5.2	PRE-PROCESSING . . . . .	182
5.2.1	Création des sous-ensembles . . . . .	182
5.2.2	Data split - Nettoyage - Encodage . . . . .	182
5.2.3	Modellisation . . . . .	184
5.2.4	Procédure d'évaluation . . . . .	184
5.3	Modelisation . . . . .	187
5.3.1	Création des sous-ensembles . . . . .	187
5.3.2	TrainTest - Nettoyage - Encodage . . . . .	187
5.3.3	Procédure d'évaluation . . . . .	189

5.3.4	Modellisation . . . . .	189
5.3.5	OPTIMISATION . . . . .	193
5.3.6	Precision Recall Curve . . . . .	195
<b>6</b>	<b>Projets Global</b>	<b>197</b>
6.1	Description du problème : . . . . .	197
6.2	Travail à faire . . . . .	197
6.3	Exploratory Data Analysis . . . . .	198
6.3.1	Analyse de la forme des données . . . . .	198
6.3.2	Analyse du Fond . . . . .	199
6.3.3	Preprocessing . . . . .	210
6.4	Modelisation du probleme . . . . .	214
6.4.1	test des Hypothese . . . . .	214
6.4.2	Construction des modeles de prediction . . . . .	219
6.4.3	Evaluation des modeles . . . . .	234
6.5	Conclusion . . . . .	240

# Chapter 1

## Bases de Python

### 1.1 Variables et Fonctions

#### 1.1.1 Variables et Opérations

Il existe 4 grands types de variables - input / output - int (nombre entier) - float (nombre décimal) - string (chaine de caractères) - bool (booléen)

```
[11]: x=input() # Lire les entrées clavier  
       print(x) # afficher les résultats
```

```
5  
5
```

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

```
[12]: # Opérations arithmétiques  
      print('x + y =', x + y)  
      print('x - y =', x - y)  
      print('x / y =', x / y)  
      print('x // y =', x // y) # division entière (très utile pour les tableaux Numpy)  
      print('x * y =', x * y)  
      print('x ^ y =', x ** y) # x puissance y
```

```
x + y = 5.5  
x - y = 0.5  
x / y = 1.2  
x // y = 1.0  
x * y = 7.5  
x ^ y = 15.588457268119896
```

```
[13]: # Opérations de comparaison
print('égalité :', x == y)
print('inégalité :', x != y)
print('inférieur ou égal :', x <= y)
print('supérieur ou égal :', x >= y)
```

égalité : False  
 inégalité : True  
 inférieur ou égal : False  
 supérieur ou égal : True

```
[14]: # Opérations Logiques
print('ET :', False and True)
print('OU :', False or True)
print('OU exclusif :', False ^ True)
```

ET : False  
 OU : True  
 OU exclusif : True

Note : Les opérations de comparaison et de logique utilisées ensemble permettent de construire des structures algorithmiques de bases (if/esle, while, ...)

```
[6]: # Table Logiques de AND
print(' A     B    ||  A and B ')
print('-----')
print('False True  ||  ', False and True)
print('False False ||  ', False and False)
print('True  True  ||  ', True and True)
print('True  False ||  ', True and False)
```

A	B		A and B
<hr/>			
False	True		False
False	False		False
True	True		True
True	False		False

## 1.1.2 Fonctions

Une fonction anonyme est une fonction créée avec **lambda**. Ce type de fonction est basique et est utile pour être intégrée au milieu de structures de contrôles ou bien d'autres fonctions. On l'utilise rarement.

```
[7]: # Exemple d'une fonction f(x) = x^2
f = lambda x : x**2

print(f(3))
```

9

```
[8]: # Exemple d'une fonction  $g(x, y) = x^2 - y^2$ 
g = lambda x, y : x**2 - y**2

print(g(4, 2))
```

12

```
[10]: print(g(f(2), 2))
```

12

La meilleure façon de créer une fonction est d'utiliser la structure suivante : **def**

```
[21]: # une fonction a un nom, prend des entrées (arguments) et les transforme pour
      ↳ retourner un résultat

def nom_de_la_fonction(argument_1, argument_2):
    resultat = argument_1 + argument_2
    return resultat

nom_de_la_fonction(3, 2)
```

[21]: 5

```
[22]: # Exemple concret : fonction qui calcul l'énergie potentielle d'un corps

def e_potentielle(masse, hauteur, g=9.81):
    energie = masse * hauteur * g
    return energie

# ici g a une valeur par défaut donc nous ne sommes pas obligé de lui donner une
      ↳ valeur
e_potentielle(masse=10, hauteur=10)
```

[22]: 981.0

### 1.1.3 Exercice

Modifiez la fonction `e_potentielle` définie plus haut pour retourner une valeur indiquant si l'énergie calculée est supérieure ou inférieure à une `energie_limite` passée en tant que 4ème argument

## 1.2 Structures de Contrôle

Dans le monde de la programmation, il existe 3 principales **structures de contrôle** pour créer des algorithmes. - Les Blocks d'instructions - Les instructions conditionnelles **If / Else** - Les instruc-

tions itératives **For, while**

### 1.2.1 Instructions conditionnelles :

Cette structure permet de tester une séquence d'alternatives. Si une condition est respectée, alors les instructions qui la suivent sont exécutées et la structure de contrôle est stoppée, sinon la condition suivante est testée.

```
[0]: def test_du_signe(valeur):
    if valeur < 0:
        print('négatif')
    elif valeur == 0:
        print('nul')
    else:
        print('positif')
```

```
[6]: test_du_signe(-2)
```

négatif

Note importante : Une condition est respectée si et seulement si elle correspond au résultat **booléen True**.

```
[7]: valeur = -2
print(valeur < 0) # le résultat de cette comparaison est True

if valeur < 0:
    print('négatif')
```

True

négatif

Cela permet de développer des algorithmes avec des mélanges d'opérations Logiques et d'opérations de comparaisons. Par exemple : *si il fait beau et qu'il faut chaud, alors j'irai me baigner*

```
[10]: x = 3
y = -1
if (x>0) and (y>0):
    print('x et y sont positifs')
else:
    print('x et y ne sont pas tous les 2 positifs')
```

x et y ne sont pas tous les 2 positifs

### 1.2.2 Instructions itératives For, while

#### Boucle for

Une boucle for permet de créer des algorithmes itératifs (qui effectuent une certaine tâche plusieurs fois de suite). Pour ça, la boucle parcourt tous les éléments d'un objet dit **itérable**. Il

peut s'agir d'une liste, d'un dictionnaire, d'un range, d'un tableau numpy, ou de bien d'autres objets...

[13]: # range(début, fin, pas) est une built-in fonction tres utile de python qui ↳ retourne un itérable.

```
for i in range(0, 10):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

[16]: # on peut s'amuser à combiner cette boucle avec notre fonction de tout à l'heure.

```
for i in range(-10, 10, 2):
    print(i)
    test_du_signe(i)
```

```
-10
négatif
-8
négatif
-6
négatif
-4
négatif
-2
négatif
0
nul
2
positif
4
positif
6
positif
8
positif
```

## Boucle While

Une boucle While permet d'effectuer en boucle une action, tant que la condition d'exécution est validée (tant que la condition est **True**)

```
[17]: x = 0
while x < 10:
    print(x)
    x += 1 # incrémente x de 1 (équivalent de x = x+1)
```

0  
1  
2  
3  
4  
5  
6  
7  
8  
9

### 1.2.3 Exercices

#### Exercice 1

Implémentez la **suite de Fibonacci** [0, 1, 1, 2, 3, 5, 8, 13, 21, ...] qui part de 2 nombres a=0 et b=1, et qui calcule le nombre suivant en additionnant les 2 nombres précédents.

Indices : - Pour cet exercice vous aurez besoin d'une boucle **While** - Vous pouvez imprimer cette suite jusqu'à atteindre un nombre **n** que vous aurez choisi - dans python il est possible de mettre à jour 2 variables simultanément sur la même ligne : **a, b = b, a+b**

```
[1]: # Votre code ici
```

#### Exercice 2

Implémentez l'algorithme du jeu "trouver le nombre" qui est le suivant : Un nombre aléatoire sera généré et le but est de le trouver. - À chaque itération, l'utilisateur essaie de deviner le nombre. - si l'utilisateur devine le numéro correctement, le jeu se terminera et un message de félicitations apparaîtra - si l'utilisateur donne un nombre supérieur, un message "inférieur" apparaîtra - si l'utilisateur donne un nombre bas, un message "supérieur" apparaîtra

```
[13]: #vous pouvez utiliser la fonction random
import random
x= random.randint(0,1000)
print(x)
```

[14] : # Votre code ici

```
901
congratulation you found the number
```

## 1.3 Structure de données (Listes et Tuples)

Une structure de données est une variable capable de contenir plusieurs valeurs à la fois. Une structure de données peut former une **séquence** si les valeurs qui la composent sont rangées dans un certains ordre. C'est le cas des **listes** et des **tuples**. A l'inverse un **Dictionnaire** ne forme pas une séquence.

### 1.3.1 Crédation de Listes et de Tuples

une liste ou un tuple peuvent contenir tout types de valeurs (int, float, bool, string). On dit que ce sont des structures hétérogènes.

La différence entre les 2 est qu'une liste est **mutable** alors qu'un Tuple ne l'est pas (on ne peut pas le changer après qu'il soit créé)

[0] : # Listes

```
liste_1 = [1, 4, 2, 7, 35, 84]
villes = ['Paris', 'Berlin', 'Londres', 'Bruxelles']
nested_list = [liste_1, villes] # une liste peut même contenir des listes ! On appelle cela une nested list
```

#Tuples

```
tuple_1 = (1, 2, 6, 2)
```

[2] : print(villes)

```
['Paris', 'Berlin', 'Londres', 'Bruxelles']
```

### 1.3.2 Indexing et Slicing

Dans une séquence, chaque élément est rangé selon un **index** (le premier index étant l'index 0)

Pour accéder à un élément d'une liste ou d'un tuple, on utilise une technique appelée **Indexing**

Pour accéder à plusieurs éléments d'une liste ou d'un tuple, on utilise une technique appelée **Slicing**

[7] : # INDEXING

```
print('séquence complète:', villes)
print('index 0:', villes[0])
print('index 1:', villes[1])
print('dernier index (-1):', villes[-1])
```

```
séquence complète: ['Paris', 'Berlin', 'Londres', 'Bruxelles']
index 0: Paris
index 1: Berlin
dernier index (-1): Bruxelles
```

[9]: # SLICING [début (inclus) : fin (exclus) : pas]

```
print('séquence complète:', villes)
print('index 0-2:', villes[0:3])
print('index 1-2:', villes[1:3])
print('ordre inverse:', villes[::-1])
```

```
séquence complète: ['Paris', 'Berlin', 'Londres', 'Bruxelles']
index 0-2: ['Paris', 'Berlin', 'Londres']
index 1-2: ['Berlin', 'Londres']
ordre inverse: ['Bruxelles', 'Londres', 'Berlin', 'Paris']
```

### 1.3.3 Actions utiles sur les listes

[3]: villes = ['Paris', 'Berlin', 'Londres', 'Bruxelles'] # liste initiale  
`print(villes)`

`villes.append('Dublin') # Rajoute un élément à la fin de la liste`  
`print(villes)`

`villes.insert(2, 'Madrid') # Rajoute un élément à l'index indiqué`  
`print(villes)`

`villes.extend(['Amsterdam', 'Rome']) # Rajoute une liste à la fin de notre liste`  
`print(villes)`

`print('longeur de la liste:', len(villes)) # affiche la longueur de la liste`

`villes.sort(reverse=False) # trie la liste par ordre alphabétique / numérique`  
`print(villes)`

`print(villes.count('Paris')) # compte le nombre de fois qu'un élément apparaît dans la liste`

```
['Paris', 'Berlin', 'Londres', 'Bruxelles']
['Paris', 'Berlin', 'Londres', 'Bruxelles', 'Dublin']
['Paris', 'Berlin', 'Madrid', 'Londres', 'Bruxelles', 'Dublin']
['Paris', 'Berlin', 'Madrid', 'Londres', 'Bruxelles', 'Dublin', 'Amsterdam',
'Rome']
longeur de la liste: 8
['Amsterdam', 'Berlin', 'Bruxelles', 'Dublin', 'Londres', 'Madrid', 'Paris',
'Rome']
```

Les listes et les tuples fonctionnent en harmonie avec les structures de contrôle **if/else** et **For**

```
[23]: if 'Paris' in villes:
    print('oui')
else:
    print('non')
```

oui

```
[24]: for element in villes:
    print(element)
```

Amsterdam  
Berlin  
Bruxelles  
Dublin  
Londres  
Madrid  
Paris  
Rome

La fonction **enumerate** est très utile pour sortir à la fois les éléments d'une liste et leurs **index**. C'est une fonction très utilisée en datascience

```
[4]: for index, element in enumerate(villes):
    print(index, element)
```

0 Amsterdam  
1 Berlin  
2 Bruxelles  
3 Dublin  
4 Londres  
5 Madrid  
6 Paris  
7 Rome

La fonction **zip** est aussi très utile pour itérer à travers 2 listes en parallèle. Si une liste est plus courte que l'autre, la boucle for s'arrête à la liste la plus courte

```
[7]: liste_2 = [312, 52, 654, 23, 65, 12, 678]
liste_3 = [1, 2, 3, 4, 5, 6, 7]
for element_1, element_2, element_3 in zip(villes, liste_2, liste_3):
    print(element_3, " - ", element_1, element_2)
```

1 - Amsterdam 312  
2 - Berlin 52  
3 - Bruxelles 654  
4 - Dublin 23  
5 - Londres 65

```
6 - Madrid 12
7 - Paris 678
```

### 1.3.4 Exercice

Transformer le code suivant qui donne la **suite de Fibonacci** pour enregistrer les résultats dans une liste et retourner cette liste à la fin de la fonction

```
[1]: # Exercice :
def fibonacci(n):
    a = 0
    b = 1
    while b < n:
        a, b = b, a+b
    print(a)
```

## 1.4 Dictionnaires

Les dictionnaires sont des structures de contrôle **non-ordonnées**, c'est-à-dire que les valeurs qu'ils contiennent ne sont pas rangées selon un index, mais suivant une **clé unique**.

Une utilisation parfaite des dictionnaires est pour regrouper ensemble des "variables" dans un même conteneur. (ces variables ne sont pas de vraies variables, mais des **keys**).

### 1.4.1 Définition

*On peut par exemple créer un dictionnaire inventaire qui regroupe plusieurs produits (les clefs) et leur quantités (les valeurs)*

```
[1]: inventaire = {'pommes': 100,
                  'bananes': 80,
                  'poires': 120}
```

```
[2]: inventaire.values()
```

```
[2]: dict_values([100, 80, 120])
```

```
[3]: inventaire.keys()
```

```
[3]: dict_keys(['pommes', 'bananes', 'poires'])
```

```
[4]: len(inventaire)
```

```
[4]: 3
```

### 1.4.2 fonction de manipulation

Voici comment ajouter une association key/value dans notre dictionnaire (attention si la clef existe déjà elle est remplacée)

```
[5]: inventaire['abricots'] = 30
print(inventaire)
```

```
{'pommes': 100, 'bananes': 80, 'poires': 120, 'abricots': 30}
```

Attention : si vous cherchez une clef qui n'existe pas dans un dictionnaire, python vous retourne une erreur. Pour éviter cela, vous pouvez utiliser la méthode `get()`

```
[6]: inventaire.get('peches') # n'existe pas
```

```
[7]: inventaire.get('pommes') # pomme existe
```

```
[7]: 100
```

```
[22]: Animal_list = ('dog', 'cat', 'bird', 'fish')
Animal_number = 0
MyAnimals={

}
MyAnimals.fromkeys(Animal_list,Animal_number)
```

```
[22]: {'dog': 0, 'cat': 0, 'bird': 0, 'fish': 0}
```

la méthode `pop()` permet de retirer une clef d'un dictionnaire tout en retournant la valeur associée à la clef.

```
[8]: abricots = inventaire.pop("abricots")
print(inventaire) # ne contient plus de clef abricots
print(abricots) # abricots contient la valeur du dictionnaire
```

```
{'pommes': 100, 'bananes': 80, 'poires': 120}
30
```

Pour utiliser une boucle for avec un dictionnaire, il est utile d'utiliser la méthode `items()` qui retourne à la fois les clefs et les valeurs

```
[25]: for key in inventaire.keys():
    print(key)

for value in inventaire.values():
    print(value)

for key, value in inventaire.items():
    print(key, value)
```

```
pommes
bananes
poires
100
80
120
pommes 100
bananes 80
poires 120
```

### 1.4.3 List Comprehension

Les listes comprehension sont une façon habile de créer des listes sur une seule ligne de code, ce qui rend le code beaucoup plus rapide (car python est un langage assez lent)

Les deux codes ci-dessous effectuent chacun la même opération. On peut voir (grâce à la commande %time) que le temps d'exécution avec liste comprehension est bien inférieur au temps d'exécution avec la méthode append()

```
[1]: %%time
liste = []
for i in range(100000):
    liste.append(i**2)
```

```
CPU times: user 44.2 ms, sys: 5.21 ms, total: 49.4 ms
Wall time: 56.5 ms
```

```
[2]: %%time
liste = [i**2 for i in range(100000)]
```

```
CPU times: user 35.1 ms, sys: 3.83 ms, total: 38.9 ms
Wall time: 39.6 ms
```

```
[8]: liste = [[i**2 for i in range(0,10,2)] for i in range(3)]
print(liste)
```

```
[[0, 4, 16, 36, 64], [0, 4, 16, 36, 64], [0, 4, 16, 36, 64]]
```

On peut rajouter des conditions **if** dans les listes comprehension, par exemple :

```
[3]: liste = [i**2 for i in range(100000) if (i % 2) == 0] # calcule i**2 seulement
→pour les nombres pairs.

print(liste[:10]) #affiche les 10 premiers éléments de la liste
```

```
[0, 4, 16, 36, 64, 100, 144, 196, 256, 324]
```

```
[13]: names=['adel','omar','moktar','ali']
dico1={k:v for k,v in enumerate(names)}
```

```
print((dico1))
print(dico1.keys())
print(dico1.values())
```

```
{0: 'adel', 1: 'omar', 2: 'moktar', 3: 'ali'}
dict_keys([0, 1, 2, 3])
dict_values(['adel', 'omar', 'moktar', 'ali'])
```

[15]:

```
ages=[30,23,25,26]
dico2={names:ages for names,ages in zip(names,ages)}
print((dico2))
print(dico2.keys())
print(dico2.values())
```

```
{'adel': 30, 'omar': 23, 'moktar': 25, 'ali': 26}
dict_keys(['adel', 'omar', 'moktar', 'ali'])
dict_values([30, 23, 25, 26])
```

[16]:

```
ages=[30,23,25,26]
dico2={names:ages for names,ages in zip(names,ages) if ages< 30}
print((dico2))
print(dico2.keys())
print(dico2.values())
```

```
{'omar': 23, 'moktar': 25, 'ali': 26}
dict_keys(['omar', 'moktar', 'ali'])
dict_values([23, 25, 26])
```

#### 1.4.4 Exercice

##### Exercice 1

Implémentez une fonction *trier(classeur, valeur)* qui place une valeur dans un dictionnaire en fonction de son signe

[10]:

```
classeur = {'négatifs':[], 'positifs':[]}
```

[11]:

```
def trier(classeur, valeur):
    #Code ici
    return classeur
```

##### Exercice 2

Implémentez un code qui permet de lire les valeur du classeur parameters

```
[15]: parameters = {'Ws':[1,2,3,4,8],
                   'Bs':[1,2,3,5,6]
                  }
```

## 1.5 Built-in Functions

Python contient un grand nombre de fonctions intégrées très utiles à connaître. Cela vous permet de construire des codes plus rapidement, sans avoir à développer vos propres fonctions pour les tâches les plus basiques. Dans ce notebook, je vous montre les plus importantes :

link:<https://docs.python.org/3/library/functions.html>

### 1.5.1 Fonction de bases

Utiles en toute circonstance !

```
[1]: x = -3.14
      print(abs(x)) # valeur absolue
      print(round(x)) # arrondi
```

3.14  
-3

```
[2]: liste = [-2, 3, 1, 0, -4]

      print(min(liste)) # minimum
      print(max(liste)) # maximum
      print(len(liste)) # longueur
      print(sum(liste)) # somme des éléments
```

-4  
3  
5  
-2

```
[4]: liste = [False, False, True]

      print(any(liste)) # y-a-t'il au moins un élément True ?
      print(all(liste)) # est-ce que tous les éléments sont True ?
```

True  
False

### 1.5.2 Fonction de conversion

Il peut être très utile de convertir une variable d'un type à un autre (par exemple pour faire des calculs). Pour cela, on dispose des fonctions int(), str() et float().

La fonction type() est très utile pour inspecter les types de nos variables

```
[6]: age = '32'
type(age)
```

[6]: str

```
[8]: age = int(age)
type(age)
```

[8]: int

```
[9]: age + 10
```

[9]: 42

```
[14]: float(x)
type(x)
```

[14]: float

On peut également convertir des listes en tuples, ou des tableaux Numpy (que l'on verra par la suite) en liste...

```
[15]: tuple_1 = (1, 2, 3, 4)

liste_1 = list(tuple_1) # convertir un tuple en liste
print(liste_1)
type(liste_1)
```

[1, 2, 3, 4]

[15]: list

```
[18]: print(bin(15),bin(32))
print(oct(15),oct(32))
print(hex(15),hex(32))
```

0b1111 0b100000

0o17 0o40

0xf 0x20

### 1.5.3 Fonction input()

Cette fonction est très utile pour demander à l'utilisateur du programme d'entrer une valeur dans votre programme

```
[12]: age = input('quel age avez-vous ?')
```

quel age avez-vous ?30 ans

[14]: `type(age) # age est de type string. il faut penser à le convertir si on désire faire un calcul avec`

[14]: `str`

### 1.5.4 Fonction format()

cette fonction permet d'insérer la valeur d'une variable au sein d'une chaîne de caractères (string).

Une méthode plus rapide pour utiliser cette fonction est de faire appel au **f-string**

[15]: `x = 25  
ville = 'Paris'  
  
message = 'il fait {} degrés à {}'.format(x, ville)  
print(message)`

il faut 25 degrés à Paris

[17]: `message = f'il fait {x} degrées à {ville}'  
print(message)`

il fait 25 degrées à Paris

### 1.5.5 Fonction open()

Cette fonction est l'une des plus utiles de Python. Elle permet d'ouvrir n'importe quel fichier de votre ordinateur et de l'utiliser dans Python. Différents modes existent : - le mode 'r' : lire un fichier de votre ordinateur - le mode 'w' : écrire un fichier sur votre ordinateur - le mode 'a' : (append) ajouter du contenu dans un fichier existant

[5]: `f = open('./assets/text.txt', 'w') # ouverture d'un objet fichier f  
f.write('hello')  
f.close() # il faut fermer notre fichier une fois le travail terminé`

[6]: `f = open('./assets/text.txt', 'r')  
print(f.read())  
f.close()`

hello

Dans la pratique, on écrit plus souvent **with open() as f** pour ne pas avoir à fermer le fichier une fois le travail effectué :

[7]: `with open('./assets/text.txt', 'r') as f:  
 print(f.read())`

hello

### 1.5.6 Exercice

Le code ci-dessous permet de créer un fichier qui contient les nombres carrée de 0 jusqu'à 19. L'exercice est d'implémenter un code qui permet de lire ce fichier et d'écrire chaque ligne dans une liste.

Note\_1 : la fonction `read().splitlines()` sera très utile

Note\_2 : Pour un meilleur résultat, essayer d'utiliser une liste comprehension !

```
[8]: # Ce bout de code permet d'écrire le fichier
with open('./assets/fichier.txt', 'w') as f:
    for i in range(0, 20):
        f.write(f'{i}: {i**2}\n')
    f.close()

# Écrivez ici le code pour lire le fichier et enregistrer chaque lignes dans une liste.
```

```
[12]: # SOLUTION (non optimale)
```

```
[1]: # SOLUTION (Améliorée)
# Une meilleure façon de procéder est d'utiliser une liste comprehension !
```

```
[ ]:
```

## 1.6 Modules de bases de Python

Python contient un certain nombre de modules intégrés, qui offrent de nombreuses fonctions mathématiques, statistiques, aléatoires et très utiles. Pour importer un module, il faut procéder comme-ci dessous : - **import module** (importe tout le module) - **import module as md** (donne un surnom au module) - **from module import fonction** (importe une fonction du module)

```
[5]: import math
import statistics
import random
import os
import glob
```

### 1.6.1 Modules math et statistics

les modules math et statistics sont en apparence très utiles, mais en data science, nous utiliserons leurs équivalents dans le package **NUMPY**. Il peut néanmoins être intéressant de voir les fonctions de bases.

```
[8]: print(math.pi)
print(math.cos(2*math.pi))
print(math.exp(1))
```

```
3.141592653589793
1.0
2.718281828459045
```

```
[13]: liste = [1, 4, 6, 2, 5, 3, 9, 6, 2, 1, 8, 8, 10, 9, 9]

print(statistics.mean(liste)) # moyenne de la liste
print(statistics.variance(liste)) # variance de la liste
print(statistics.median(liste)) # median de la liste
```

```
5.533333333333333
10.266666666666666
6
```

## 1.6.2 Module Random

Le module random est l'un des plus utile de Python. En datascience, nous utiliserons surtout sont équivalent **NUMPY**

```
[14]: random.seed(0) # fixe le générateur aléatoire pour produire toujours le même résultat

print(random.choice(liste)) # choisit un élément au hasard dans la liste
print(random.random()) # génère un nombre aléatoire entre 0 et 1
print(random.randint(5, 10)) # génère un nombre entier aléatoire entre 5 et 10
```

```
9
0.3852453064766108
8
```

```
[15]: random.sample(range(100), 10) # retourne une liste de 10 nombres aléatoires entre 0 et 100
```

```
[15]: [5, 33, 65, 62, 51, 38, 61, 45, 74, 27]
```

```
[16]: print('liste de départ', liste)

random.shuffle(liste) #mélange les éléments d'une liste

print('liste mélangée', liste)
```

```
liste de départ [1, 4, 6, 2, 5, 3, 9, 6, 2, 1, 8, 8, 10, 9, 9]
liste mélangée [2, 9, 8, 9, 3, 1, 6, 8, 10, 1, 4, 9, 5, 6, 2]
```

### 1.6.3 Modules OS et Glob

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

```
[17]: os.getcwd() # affiche le répertoire de travail actuel
```

```
[17]: '/Users/mac/Documents/Teaching/Machine learning/Python-Machine-Learning/Formations python for machine learning/Introduction to python'
```

```
[18]: print(glob.glob('*')) # contenu du repertoire de travail actuel
```

### 1.6.4 Créer notre propre module

Vous pouvez également créer vos propres modules et les importer dans d'autres projets. Un module n'est en fait qu'un simple fichier.py qui contient des fonctions et des classes

```
[1]: import MyModule as a
```

```
[3]: a.somme(4,5)  
a.substraction(5,9)
```

```
[3]: -4
```

```
[4]: from MyModule import somme  
a.somme(4,5)
```

```
[4]: 9
```

## 1.7 Programmation Orientée Objet avec Python

La programmation orientée objet est un **paradigme**, c'est à dire une façon de procéder pour écrire des programmes clairs et simples. Le principe est de modéliser les éléments de notre programme (comme les tableaux et les listes) comme étant des **objets** caractérisés par des **attributs** et capables d'effectuer des **actions**. Ces objets sont construits à partir de **classes** qui contiennent leur plan de fabrication.

Dans le langage Python, presque tout est construit pour être un objet : les listes, les dictionnaires, les tableaux numpy, etc. Par exemple quand on écrit : `list.append()`, on utilise en fait la méthode `append()` sur un objet liste.

La documentation Python, Numpy, Pandas, Matplotlib, Sklearn, est donc en vaste majorité constituée de classes qu'il est important de savoir déchiffrer pour pouvoir apprendre soi-même de nouvelles choses grâce aux documentations.

### 1.7.1 Créer des classes

Voici donc comment créer simplement et efficacement des classes :

```
[0]: 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}'')
```

```
[0]: # création d'un objet de la classe voiture
voiture_1 = vehicule(couleur='rouge')
```

```
[0]: voiture_1.accelerer(100)
```

```
[47]: voiture_1.afficher()
```

```
couleur:rouge
roues:4
vitesse:100
```

## 1.7.2 Créer des sous-classes

```
[0]: class voiture_electrique(vehicule):
    """
    La classe moto hérite des méthodes et des attributs de la classe véhicule
    """

    def __init__(self, couleur='black', vitesse=0, roues=4, autonomie=100):
        super().__init__(couleur, vitesse, roues) # super() permet d'utiliser la
        #fonction de la classe "parent"
```

```
    self.autonomie = autonomie

    # Ré-écriture de certaines méthodes
    def accelerer(self, vitesse):
        super().accelerer(vitesse)
        self.autonomie -= 0.1 *self.vitesse

    def afficher(self):
        super().afficher()
        print(f'autonomie: {self.autonomie}' )
```

```
[0]: voiture_2 = voiture_electrique()
```

```
[54]: voiture_2.afficher()

voiture_2.accelerer(10)

voiture_2.afficher()
```

```
couleur:black
roues:4
vitesse:0
autonomie: 100
couleur:black
roues:4
vitesse:10
autonomie: 99.0
```

## 1.8 Solutions aux exercices

### 1.8.1 Variables et Fonctions

```
[3]: def e_potentielle(masse, hauteur, e_limite, g=9.81):

    energie = masse * hauteur * g
    return energie > e_limite

e_potentielle(masse=10, hauteur=8, e_limite=800)
```

```
[3]: False
```

### 1.8.2 Structures de Contrôle

```
[ ]: #Solution 1
def fibonacci(n):
    # retourne une liste contenant la suite de fibonacci jusqu'a n
    a = 0
    b = 1
    while b < n:
        a, b = b, a+b
        print(a)

fibonacci(1000)
```

```
[ ]: #Solution 2
find= False
while not find:
    a = int(input())
    if a == x:
        print('congratulation you found the number')
        find= True
    elif x> a:
        print('up')
    else:
        print('down')
```

### 1.8.3 Structure de données (Listes et Tuples)

```
[ ]: def fibonacci(n):
    a = 0
    b = 1
    fib = [a] # Création d'une liste fib
    while b < n:
        a, b = b, a+b
        fib.append(a) # ajoute la nouvelle valeur de a a la fin de fib
    return fib

print(fibonacci(1000))
```

### 1.8.4 Dictionnaires

```
[5]: def trier(classeur, valeur):
    if valeur >=0:
        classeur['positifs'].append(valeur)
    else:
        classeur['négatifs'].append(valeur)
    return classeur
```

```
trier(classeur, 9)
trier(classeur, -6)
trier(classeur, 8)
```

[5]: {'négatifs': [-6], 'positifs': [9, 8]}

### 1.8.5 Built-in Functions

```
[13]: # Ce bout de code permet d'écrire le fichier
with open('./assets/fichier.txt', 'w') as f:
    for i in range(0, 20):
        f.write(f'{i}: {i**2}\n')
    f.close()

# Écrivez ici le code pour lire le fichier et enregistrer chaque lignes dans une liste.
# SOLUTION (non optimale)
with open('./assets/fichier.txt', 'r') as f:
    liste = f.read().splitlines()

liste

# SOLUTION (Améliorée)
# Une meilleure façon de procéder est d'utiliser une liste comprehension !

liste2 = [row.strip() for row in open('./assets/fichier.txt', 'r')]
liste2
```

[13]: ['0: 0',
 '1: 1',
 '2: 4',
 '3: 9',
 '4: 16',
 '5: 25',
 '6: 36',
 '7: 49',
 '8: 64',
 '9: 81',
 '10: 100',
 '11: 121',
 '12: 144',
 '13: 169',
 '14: 196',
 '15: 225',
 '16: 256',
 '17: 289',

```
'18: 324',  
'19: 361']
```

[ ]:

## Chapter 2

# Modules scientifiques

## 2.1 Numpy :

### 2.1.1 Fonctions de base

NumPy est le package fondamental pour le calcul scientifique en Python. Il s'agit d'une bibliothèque Python qui fournit un objet tableau multidimensionnel, divers objets dérivés (tels que des tableaux masqués et des matrices) et un assortiment de routines pour des opérations rapides sur des tableaux, y compris mathématiques, logiques, manipulation de forme, tri, sélection, E/S , transformées de Fourier discrètes, algèbre linéaire de base, opérations statistiques de base, simulation aléatoire et bien plus encore.

Pour accéder à NumPy et à ses fonctions, importez-le dans votre code Python comme ceci :

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

Nous raccourcissons le nom importé en np pour une meilleure lisibilité du code à l'aide de NumPy. Il s'agit d'une convention largement adoptée que vous devez suivre afin que toute personne travaillant avec votre code puisse facilement la comprendre.

### Générateurs de tableaux ndarray

```
[2]: A = np.array([1, 2, 3]) # générateur par défaut, qui permet de convertir des listes (ou autres objets) en tableau ndarray
      A = np.zeros((2, 3)) # tableau de 0 aux dimensions 2x3
      B = np.ones((2, 3)) # tableau de 1 aux dimensions 2x3
```

```
[3]: np.random.seed(0)
      C = np.random.randn(2, 3) # tableau aléatoire (distribution normale) aux dimensions 2x3
      D = np.random.rand(2, 3) # tableau aléatoire (distribution uniforme)
```

```
[4]: size = (2, 3)
```

```
E = np.random.randint(0, 10, size) # tableau d'entiers aléatoires de 0 à 10 et
    ↪de dimension 2x3
print(E)
```

```
[[8 1 5]
 [9 8 9]]
```

[5]: B = np.eye(4, dtype=bool) # créer une matrice identité et convertit les éléments en type bool.

```
print(B)
```

```
[[ True False False False]
 [False  True False False]
 [False False  True False]
 [False False False  True]]
```

[6]: A = np.linspace(0,10, 11) # np.linspace(start, end, number of elements)
B = np.arange(0, 10, 1) # np.linspace(start, end, step)

```
print(A)
print(B)
```

```
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
[0 1 2 3 4 5 6 7 8 9]
```

[7]: A = np.linspace(5, 10, 7, dtype=np.float16) # définit le type et la place a
 ↪occuper sur la mémoire

```
print("float16: ",A)
A = np.linspace(5,10,7, dtype=np.float32)
print("float32: ",A)
A = np.linspace(5,10,7, dtype=np.float64)
print("float64: ",A)
```

```
float16:  [ 5.      5.832  6.668  7.5     8.336  9.164 10.     ]
float32:  [ 5.      5.8333335  6.6666665  7.5     8.333333  9.166667
 10.      ]
float64:  [ 5.      5.83333333  6.66666667  7.5     8.33333333
 9.16666667
 10.      ]
```

### Attributs importants

[8]: A = np.zeros((2, 3)) # création d'un tableau de shape (2, 3)

```
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
```

```
6
(2, 3)
<class 'tuple'>
2
```

### Méthodes importantes

- **reshape()** : pour redimensionner un tableau
- **ravel()** : pour aplatisir un tableau (qu'il ne fasse plus qu'une dimension)
- **squeeze()** : quand une dimension est égale à 1, cette dimension disparaît
- **concatenate()** : assemble 2 tableaux ensemble selon un axes (existe aussi en hstack et vstack)

[9]:

```
A = np.zeros((2, 3)) # création d'un tableau de shape (2, 3)

A = A.reshape((3, 2)) # redimensionne le tableau A (3 lignes, 2 colonnes)
print("reshape: ",A)
B = A.ravel() # Aplatit le tableau A (une seule dimension)
print("ravel: ",B)
```

```
reshape:  [[0. 0.]
 [0. 0.]
 [0. 0.]]
ravel:  [0. 0. 0. 0. 0. 0.]
```

[10]:

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

A= A.reshape((A.shape[0],1))
print(A.shape)

A=A.squeeze()
print(A.shape)
```

```
[1 2 5 3]
(4,)
(4, 1)
(4,)
```

[11]:

```
A = np.zeros((2, 3)) # création d'un tableau de shape (2, 3)
B = np.ones((2, 3)) # création d'un tableau de shape (2, 3)

np.concatenate((A, B), axis=0) # axe 0 : équivalent de np.vstack((A, B))
```

[11]:

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

```
[12]: np.concatenate((A, B), axis=1) # axe 1 : équivalent de np.hstack((A, B))
```

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

```
[13]: A = np.zeros((2, 3)) # création d'un tableau de shape (2, 3)
B = np.ones((2, 3)) # création d'un tableau de shape (2, 3)
C = np.hstack((A,B))
print(C)
print(C.shape)
```

```
[[0. 0. 0. 1. 1. 1.]
 [0. 0. 0. 1. 1. 1.]]
(2, 6)
```

```
[14]: C = np.vstack((A,B))
print(C)
print(C.shape)
```

```
[[0. 0. 0.]
 [0. 0. 0.]
 [1. 1. 1.]
 [1. 1. 1.]]
(4, 3)
```

## Exercice

```
[15]: def initialisation(m, n):
    # m : nombre de lignes
    # n : nombre de colonnes
    # retourne une matrice aléatoire (m, n+1)
    # avec une colonne biais (remplie de "1") tout à droite
    return X
```

```
[16]: def initialisation(m, n):
    # m : nombre de lignes
    # n : nombre de colonnes
    # retourne une matrice aléatoire (m, n+1)
    # avec une colonne biais (remplie de "1") tout à droite
    #-----> votre code ici <-----#
    return X
initialisation(3, 3)
```

## 2.1.2 Slicing et Indexing

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

### Indexing et Slicing

Le fonctionnement est le même que pour les listes

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

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

```
[14]: # Pour accéder à la ligne 0, colonne 1
A[0, 1]
```

```
[14]: 10
```

```
[16]: # Pour sélectionner les blocs de la ligne (0-1) colonne (0-1)
A[0:2, 0:2]
```

```
[16]: array([[10, 10],
 [10, 10]])
```

```
[17]: A[0:2, 0:2] = 10
print(A)
```

```
[[10 10  3]
 [10 10  6]]
```

### Boolean Indexing

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

print(A<5) # masque booléen

print(A[A < 5]) # sous-ensemble filtré par le masque booléen

A[A<5] = 4 # convertit les valeurs sélectionnées.
print(A)
```

```
[[ True  True  True]
 [ True False False]]
[1 2 3 4]
[[4 4 4]
 [4 5 6]]
```

## Exercices

### Exercice 1

Remplir les 4 blocs du milieux par des "1"

```
[10]: B = np.zeros((4, 4))
B
```

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

```
[11]: # SOLUTION
B[1:3 , 1:3] = 1
B
```

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

### Exercice 2

Remplir le tableau de "1" (une ligne sur deux, une colonne sur deux depuis la première ligne/colonne jusqu'à l'avant dernier)

```
[12]: C = np.zeros((10, 10))
C
```

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

```
[14]: # SOLUTION
```

### Exercice 3

Sur l'image ci dessous, effectuer un slicing pour ne garder que la moitié de l'image (en son centre) et remplacer tous les pixels > 150 par des pixels = 255

```
[23]: from scipy import misc
import matplotlib.pyplot as plt
face = misc.face(gray=True)
plt.imshow(face, cmap=plt.cm.gray)
plt.show()
face.shape
```



```
[15]: # SOLUTION
```

### 2.1.3 Mathématiques

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

#### Méthodes de bases

```
[34]: A = np.array([[1, 2, 3], [4, 5, 6]])
print(A)
print("la somme des éléments",A.sum()) # effectue la somme de tous les éléments
→ du tableau
print("la somme des éléments colonne",A.sum(axis=0)) # effectue la somme des
→ colonnes (somme sur éléments des les lignes)
print("la somme des éléments ligne",A.sum(axis=1)) # effectue la somme des
→ lignes (somme sur les éléments des colonnes)
print("la somme cumulative",A.cumsum(axis=0)) # effectue la somme cumulée
```

```

print(A.prod()) # effectue le produit
print(A.cumprod()) # effectue le produit cumulé

print(A.min()) # trouve le minimum du tableau
print(A.max()) # trouve le maximum du tableau

print(A.mean()) # calcule la moyenne
print(A.std()) # calcule l'écart type,
print(A.var()) # calcule la variance

```

```

[[1 2 3]
 [4 5 6]]
la somme des éléments 21
la somme des éléments colonne [5 7 9]
la somme des éléments ligne [ 6 15]
la somme cumulative [[1 2 3]
 [5 7 9]]
720
[ 1    2    6   24  120 720]
1
6
3.5
1.707825127659933
2.9166666666666665

```

Une méthode très importante : la méthode **argsort()**

```
[19]: A = np.random.randint(0, 10, [5, 5]) # tableau aléatoire
print(A)
```

```

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

```
[20]: print(A.argsort()) # retourne les index pour trier chaque ligne du tableau
```

```

[[1 3 0 4 2]
 [3 4 2 0 1]
 [3 4 0 1 2]
 [1 0 4 2 3]
 [2 0 1 4 3]]
```

```
[21]: print(A[:,0].argsort()) # retourne les index pour trier la colonne 0 de A
```

```
[3 0 4 2 1]
```

```
[22]: A = A[A[:,0].argsort(), :] # trie les colonnes du tableau selon la colonne 0.
A
```

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

## Numpy Statistics

Correlation de Pearson :

```
[23]: B = np.random.randn(3, 3) # nombres aléatoires 3x3

# retourne la matrice de corrélation de B
print(np.corrcoef(B))
```

```
[[ 1.          -0.63427277  0.99937797]
 [-0.63427277  1.          -0.66114251]
 [ 0.99937797 -0.66114251  1.        ]]
```

```
[24]: # retourne la matrice de corrélation entre les lignes 0 et 1 de B
print(np.corrcoef(B[:,0], B[:, 1]))
```

```
[[1.          0.81847981]
 [0.81847981 1.        ]]
```

np.unique() :

```
[15]: np.random.seed(0)
A = np.random.randint(0, 5, [5,5])
A
```

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

```
[16]: np.unique(A)
```

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

```
[17]: values, counts = np.unique(A, return_counts=True)

for i, j in zip(values[counts.argsort()], counts[counts.argsort()]):
    print(f'valeur {i} apparait {j}'')
```

```
valeur 2 apparait 3
valeur 4 apparait 4
valeur 1 apparait 5
valeur 3 apparait 6
valeur 0 apparait 7
```

Calculs statistiques en présence de données manquantes (NaN)

```
[36]: A = np.random.randn(5, 5)
A[0, 2] = np.nan # insere un NaN dans la matrice A

print('ratio NaN/zise:', (np.isnan(A).sum()/A.size)) # calcule la proportion de NaN dans A

print('moyenne sans NaN:', np.nanmean(A)) # calcule la moyenne de A en ignorant les NaN
```

```
ratio NaN/zise: 0.04
moyenne sans NaN: 0.1832816316588837
```

## Algebre Linéaire

```
[38]: A = np.ones((2,3))
B = np.ones((3,3))

print(A.T) # transposé de la matrice A (c'est un attribut de ndarray)
```

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

```
[39]: print(A.dot(B)) # produit matriciel A.B
```

```
[[3. 3. 3.]
 [3. 3. 3.]]
```

```
[45]: A = np.random.randint(0, 10, [3, 3])

print('det=', np.linalg.det(A)) # calcule le determinant de A
print('inv A:\n', np.linalg.inv(A)) # calcul l'inverse de A
```

```
det= 61.00000000000001
inv A:
[[ 0.08196721  0.63934426 -0.60655738]
 [-0.13114754 -0.62295082  0.7704918 ]
 [ 0.21311475  0.26229508 -0.37704918]]
```

```
[44]: val, vec = np.linalg.eig(A)
print('valeur propre:\n', val) # valeur propre
print('vecteur propre:\n', vec) # vecteur propre
```

```
valeur propre:
[ 8.91371956 -0.86320273  1.94948316]
vecteur propre:
[[-0.27183844 -0.6838339   0.39494311]
 [-0.4097407  -0.15279739 -0.73291062]
 [-0.87075623  0.71345929  0.55395123]]
```

## Fonctions mathématiques

```
[13]: A = np.linspace(1,10,10)
print(np.exp(A)) # calcule l'exponentielle pour tous les element d'un tableau
print(np.log(A)) # calcule le logarithme pour tous les element d'un tableau
print(np.cos(A)) # calcule le cosinus pour tous les element d'un tableau
print(np.sin(A)) # calcule le sinus pour tous les element d'un tableau
```

```
[2.71828183e+00 7.38905610e+00 2.00855369e+01 5.45981500e+01
 1.48413159e+02 4.03428793e+02 1.09663316e+03 2.98095799e+03
 8.10308393e+03 2.20264658e+04]
[0.          0.69314718 1.09861229 1.38629436 1.60943791 1.79175947
 1.94591015 2.07944154 2.19722458 2.30258509]
[ 0.54030231 -0.41614684 -0.9899925  -0.65364362  0.28366219  0.96017029
  0.75390225 -0.14550003 -0.91113026 -0.83907153]
[ 0.84147098  0.90929743  0.14112001 -0.7568025  -0.95892427 -0.2794155
  0.6569866   0.98935825  0.41211849 -0.54402111]
```

## Exercice

Standardisez la matrice suivante, c'est à dire effectuez le calcul suivant :  $A = \frac{A - \text{mean}(A_{\text{colonne}})}{\text{std}(A_{\text{colonne}})}$

```
[30]: np.random.seed(0)
A = np.random.randint(0, 100, [10, 5])
A
```

```
[30]: array([[44, 47, 64, 67, 67],
       [ 9, 83, 21, 36, 87],
       [70, 88, 88, 12, 58],
       [65, 39, 87, 46, 88],
       [81, 37, 25, 77, 72],
       [ 9, 20, 80, 69, 79],
       [47, 64, 82, 99, 88],
       [49, 29, 19, 19, 14],
       [39, 32, 65,  9, 57],
       [32, 31, 74, 23, 35]])
```

```
[33]: print(D.mean(axis=0)) # les moyennes sont toutes = 0
print(D.std(axis=0)) # les std sont tous = 1
```

```
[-2.22044605e-17 -4.44089210e-17  0.00000000e+00 -1.22124533e-16
 -4.44089210e-17]
[1. 1. 1. 1. 1.]
```

## 2.1.4 Solutions aux exercices

### numpy

```
[2]: import numpy as np
def initialisation(m, n):
    # m : nombre de lignes
    # n : nombre de colonnes
    # retourne une matrice aléatoire (m, n+1)
    # avec une colonne biais (remplie de "1") tout à droite
    X = np.random.randn(m, n)
    X = np.concatenate((X, np.ones((X.shape[0], 1))), axis = 1)

    return X

initialisation(3, 4)
```

```
[2]: array([[-0.57268788, -2.59114491, -0.88331436,  0.3858638 ,  1.        ],
       [-0.11838144, -0.03973665, -1.62603282,  0.99373931,  1.        ],
       [-0.39381019, -0.58314658, -0.10498704, -0.16970352,  1.        ]])
```

## Numpy : Slicing et Indexing

### Exercice 1

Remplir les 4 blocs du milieux par des “1”

```
[3]: B = np.zeros((4, 4))
B
# SOLUTION
B[1:3 , 1:3] = 1
B
```

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

**Exercice 2**

Remplir le tableau de "1" (une ligne sur deux, une colonne sur deux depuis la première ligne/colonne jusqu'à l'avant dernier)

```
[8]: C = np.zeros((10, 10))
C
# SOLUTION
C[1:-1:2, 1:-1:2] = 1
C
```

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

**Exercice 3**

Sur l'image ci-dessous, effectuer un slicing pour ne garder que la moitié de l'image (en son centre) et remplacer tous les pixels > 150 par des pixels = 255

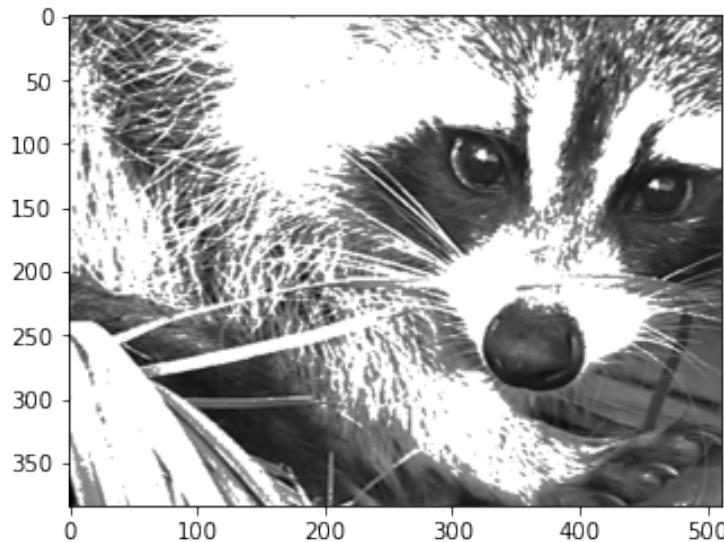
```
[12]: from scipy import misc
import matplotlib.pyplot as plt
face = misc.face(gray=True)
plt.imshow(face, cmap=plt.cm.gray)
plt.show()
face.shape
```



[12]: (768, 1024)

[13] : # SOLUTION

```
x, y = face.shape
zoom_face = face[x//4 : -x//4, y //4: -y//4] # redimensionner en divisant chaque dimension par 4 (division entiere)
zoom_face[zoom_face>150] = 255 # boolean indexing
plt.imshow(zoom_face, cmap=plt.cm.gray)
plt.show()
```



## Numpy : Mathématiques

[36] : np.random.seed(0)

```
A = np.random.randint(0, 100, [10, 5])
print(A)
# SOLUTION
D = (A - A.mean(axis=0)) / A.std(axis=0)
D
```

```
[[44 47 64 67 67]
 [ 9 83 21 36 87]
 [70 88 88 12 58]
 [65 39 87 46 88]
 [81 37 25 77 72]
 [ 9 20 80 69 79]
 [47 64 82 99 88]
 [49 29 19 19 14]
 [39 32 65  9 57]
 [32 31 74 23 35]]
```

```
[36]: array([[-0.02206157,  0.          ,  0.13173823,  0.72539252,  0.10755798] ,  
           [-1.56637126,  1.61579632, -1.48676006, -0.33034307,  0.96802178] ,  
           [ 1.12513992,  1.84021247,  1.03508612, -1.14768676, -0.27965074] ,  
           [ 0.90452425, -0.35906585,  0.99744662,  0.0102168 ,  1.01104497] ,  
           [ 1.6104944 , -0.44883231, -1.33620208,  1.0659524 ,  0.32267393] ,  
           [-1.56637126, -1.21184724,  0.73397016,  0.7935045 ,  0.62383626] ,  
           [ 0.11030784,  0.76301493,  0.80924915,  1.81518411,  1.01104497] ,  
           [ 0.1985541 , -0.80789816, -1.56203905, -0.90929485, -2.17267111] ,  
           [-0.24267724, -0.67324847,  0.16937773, -1.24985473, -0.32267393] ,  
           [-0.55153918, -0.7181317 ,  0.50813319, -0.77307091, -1.26918412]])
```

```
[ ]:
```

## 2.2 Matplotlib

### 2.2.1 Pyplot

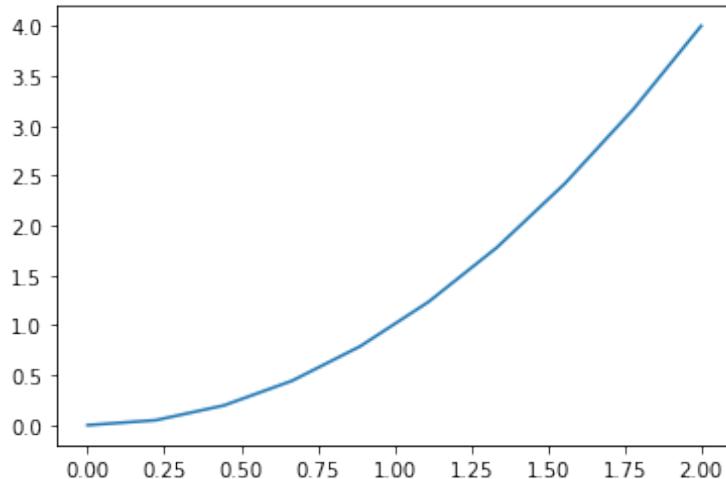
matplotlib.pyplot est une interface basée sur l'état de matplotlib. Il fournit un moyen de traçage implicite, semblable à MATLAB. Il ouvre également les figures sur votre écran et agit en tant que gestionnaire de l'interface graphique des figures. link: [https://matplotlib.org/stable/api/\\_as\\_gen/matplotlib.pyplot.html](https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html)

pyplot est principalement destiné aux tracés interactifs et aux cas simples de génération de tracés programmatiques :

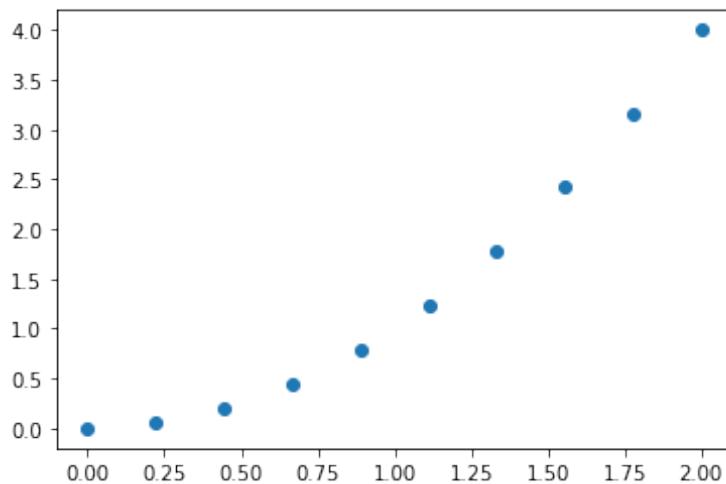
```
[2]: import numpy as np  
import matplotlib.pyplot as plt
```

### 2.2.2 Graphiques simples

```
[5]: X = np.linspace(0, 2, 10)  
y = X**2  
  
plt.plot(X, y)  
plt.show()
```



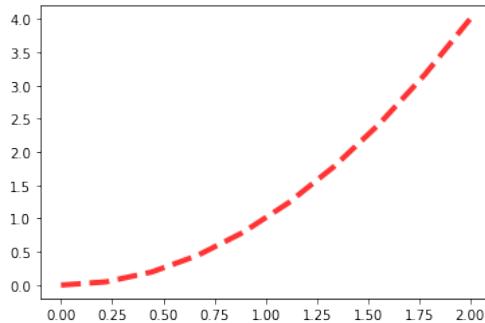
```
[6]: plt.scatter(X, y)
plt.show()
```



### 2.2.3 Styles Graphiques

Il existe beaucoup de styles à ajouter aux graphiques. Voici les plus importants à retenir : - **c** : couleur de la ligne - **lw** : épaisseur de la ligne (pour les graphiques `plot`) - **ls** : style de la ligne (pour les graphiques `plot`) - **size** : taille du point (pour les graphiques `scatter`) - **marker** : style de points (pour les graphiques `scatter`) - **alpha** : transparence du graphique

```
[9]: plt.plot(X, y, c='red', lw=4, ls='--', alpha=0.8)
plt.show()
```



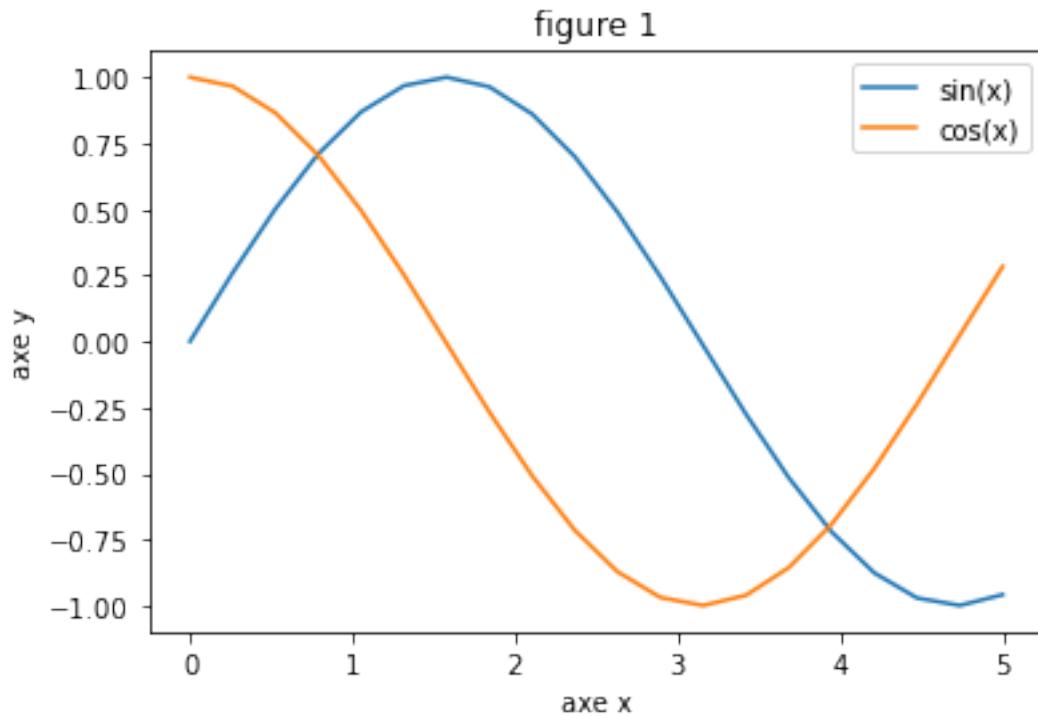
## 2.2.4 Cycle de vie d'une figure

Pour créer des figures proprement, on doit suivre le cycle de vie suivant : 1. `plt.figure(figsize())`  
2. `plt.plot()` 3. Extras (titre, axes, légendes) 4. `plt.show()`

```
[14]: X = np.linspace(0, 5, 20)

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

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

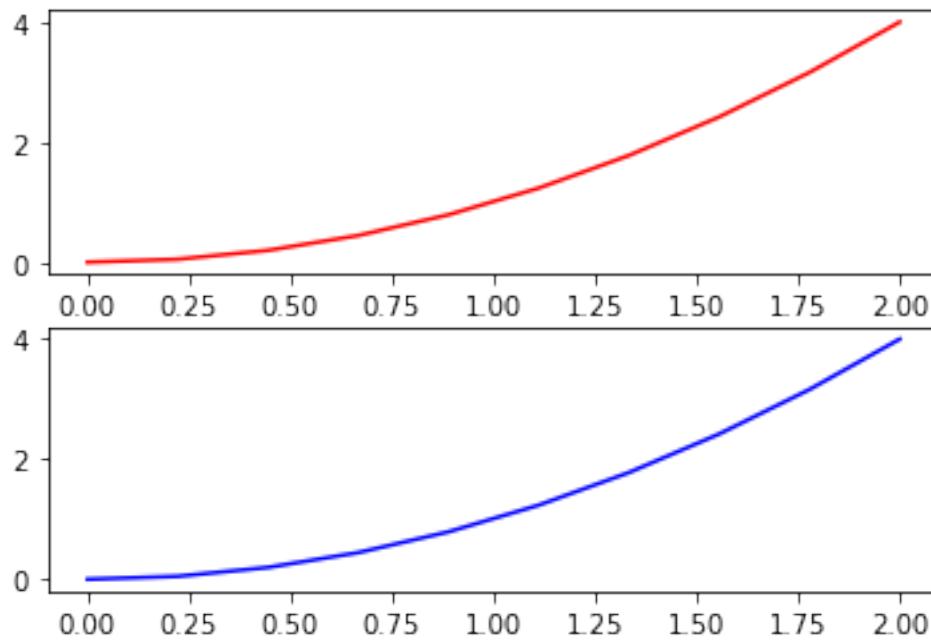


## 2.2.5 Subplots

Les subplots sont un autre éléments a ajouter pour créer plusieurs graphiques sur une même figure

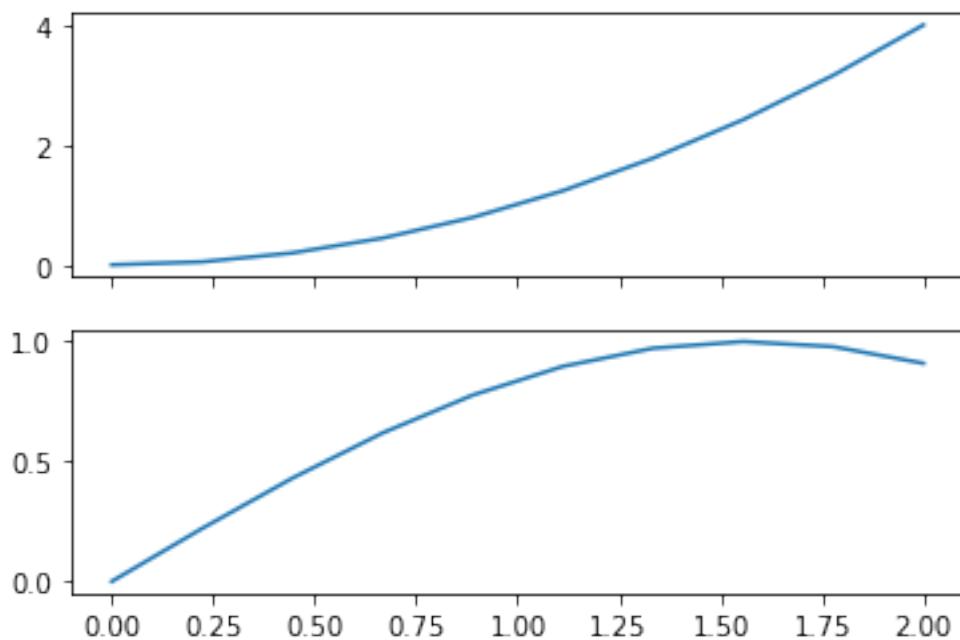
```
[13]: plt.subplot(2, 1, 1)
plt.plot(x, y, c='red')
plt.subplot(2, 1, 2)
plt.plot(x, y, c='blue')
```

```
[13]: [<matplotlib.lines.Line2D at 0x7f4610e2b940>]
```



## 2.2.6 Méthode orientée objet

```
[15]: fig, ax = plt.subplots(2, 1, sharex=True) # partage le même axe pour les subplots
ax[0].plot(x, y)
ax[1].plot(x, np.sin(x))
plt.show()
```



## 2.2.7 Matplotlib Graphiques importants

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

### Graphique de Classification (Scatter())

Les diagrammes de dispersion sont utilisés pour observer les relations entre les variables et utilisent des points pour représenter la relation entre elles. La méthode scatter() de la bibliothèque matplotlib est utilisée pour dessiner un nuage de points. Les diagrammes de dispersion sont largement utilisés pour décrire la relation entre les variables et comment le changement de l'une affecte l'autre.

```
[5]: from sklearn.datasets import load_iris
```

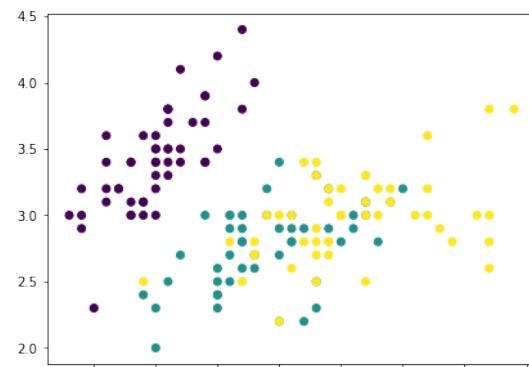
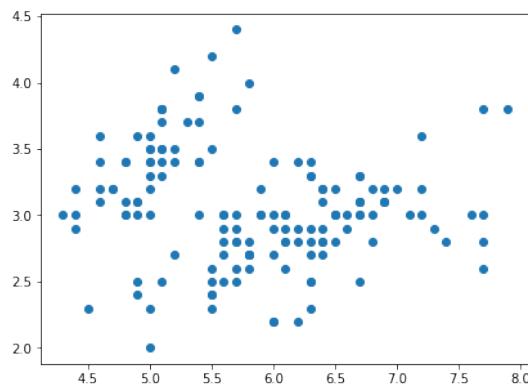
```
[24]: iris = load_iris()
x = iris.data
y = iris.target

print(f'x contient {x.shape[0]} exemples et {x.shape[1]} variables')
print(f'il y a {np.unique(y).size} classes')
```

x contient 150 exemples et 4 variables  
il y a 3 classes

```
[17]: plt.figure(figsize=(15, 5))
plt.subplot(1, 2, 1)
plt.scatter(x[:, 0], x[:, 1])
plt.subplot(1, 2, 2)
plt.scatter(x[:, 0], x[:, 1], c=y)
```

[17]: <matplotlib.collections.PathCollection at 0x7fbde1309220>

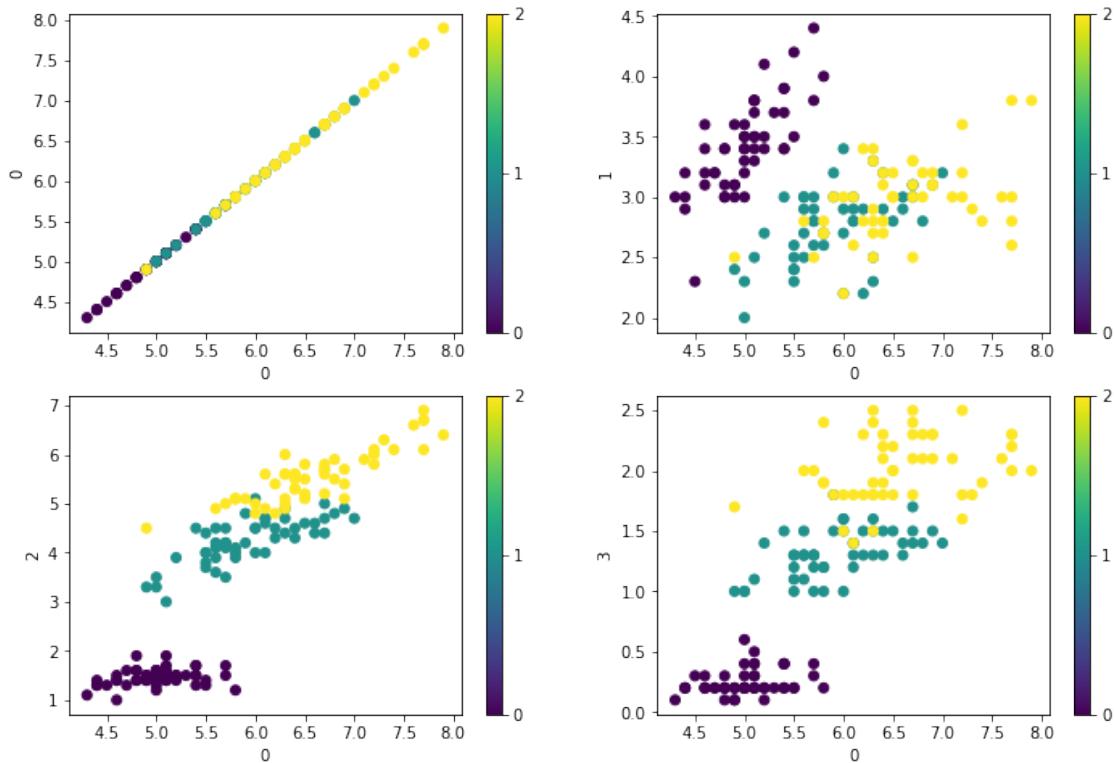


```
[10]: n = x.shape[1]
plt.figure(figsize=(12, 8))
for i in range(n):
```

```

plt.subplot(n//2, n//2, i+1)
plt.scatter(x[:, 0], x[:, i], c=y)
plt.xlabel('0')
plt.ylabel(i)
plt.colorbar(ticks=list(np.unique(y)))
plt.show()

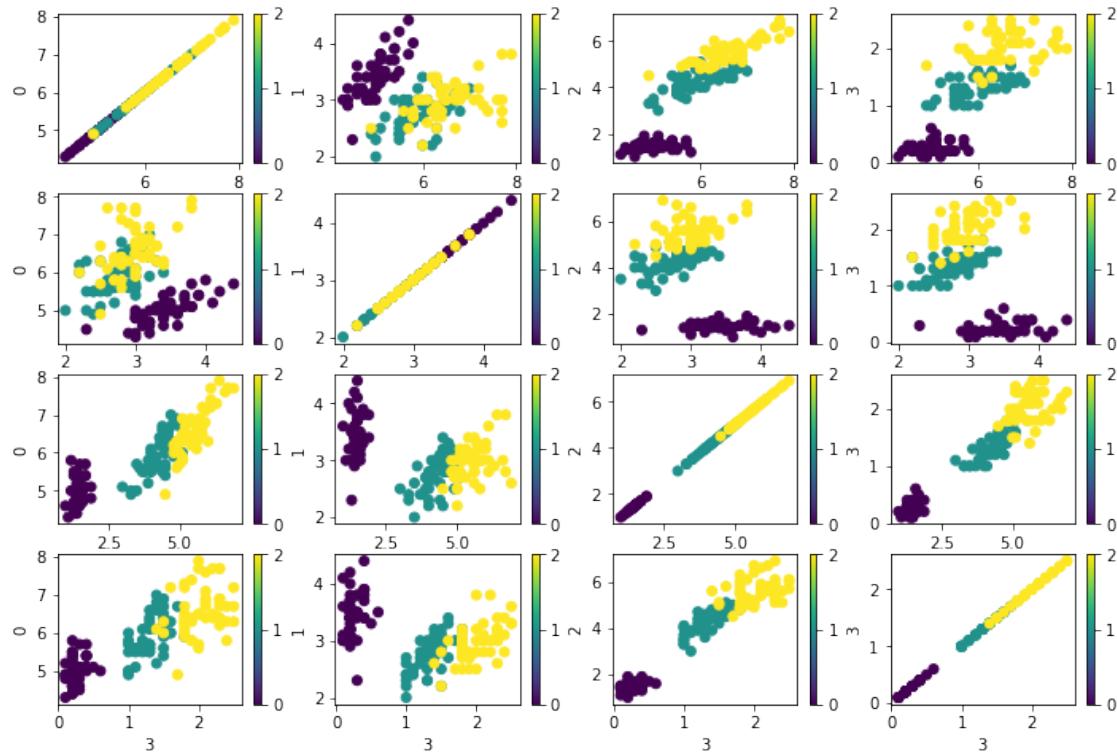
```



```

[11]: n = x.shape[1]
plt.figure(figsize=(12, 8))
for j in range(n):
    for i in range(n):
        plt.subplot(n, n, (n*j)+i+1)
        plt.scatter(x[:, j], x[:, i], c=y)
        plt.xlabel(j)
        plt.ylabel(i)
        plt.colorbar(ticks=list(np.unique(y)))
plt.show()

```

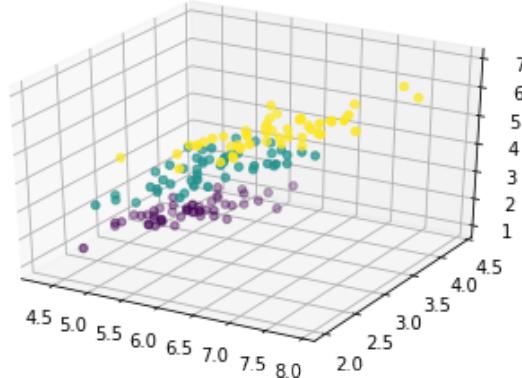


## Graphiques 3D

```
[0]: from mpl_toolkits.mplot3d import Axes3D
```

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

```
[7]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x7ff395f80358>
```

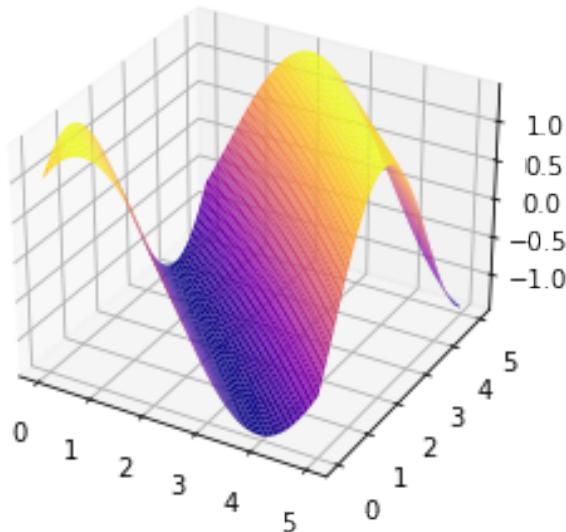


```
[18]: f = lambda x, y: np.sin(x+y) + np.cos(x+y)
```

```
X = np.linspace(0, 5, 50)
Y = np.linspace(0, 5, 50)
```

```
X, Y = np.meshgrid(X, Y)
Z = f(X, Y)

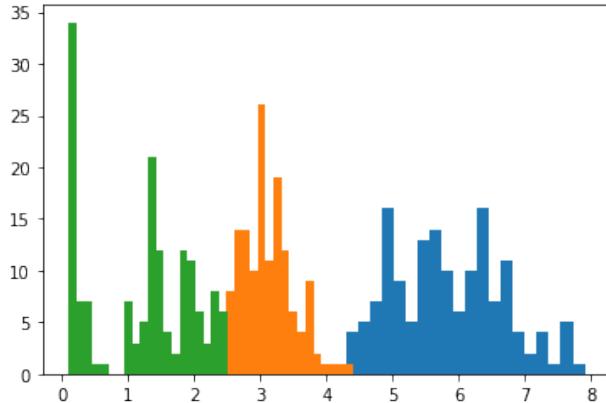
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, Z, cmap='plasma')
plt.show()
```



## Histogrammes

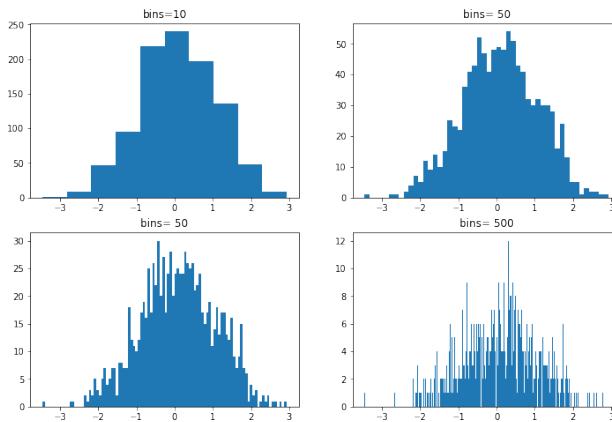
```
[29]: plt.figure()
plt.hist(x[:,0],bins=20)
plt.hist(x[:,1],bins=20)
plt.hist(x[:,3],bins=20)
```

```
[29]: (array([34.,  7.,  7.,  1.,  1.,  0.,  0.,  7.,  3.,  5., 21.,
       12.,  4.,  2., 12., 11.,  6.,  3.,  8.,  6.]),
 array([0.1 , 0.22, 0.34, 0.46, 0.58, 0.7 , 0.82, 0.94, 1.06, 1.18,
        1.3 , 1.42, 1.54, 1.66, 1.78, 1.9 , 2.02, 2.14, 2.26, 2.38, 2.5 ]),
 <BarContainer object of 20 artists>)
```



```
[21]: x = np.random.randn(1000)

plt.figure(figsize=(12, 8))
plt.subplot(221)
plt.hist(x, bins=10)
plt.title('bins=10')
plt.subplot(222)
plt.hist(x, bins=50)
plt.title('bins= 50')
plt.subplot(223)
plt.hist(x, bins=100)
plt.title('bins= 50')
plt.subplot(224)
plt.hist(x, bins=500)
plt.title('bins= 500')
plt.show()
```

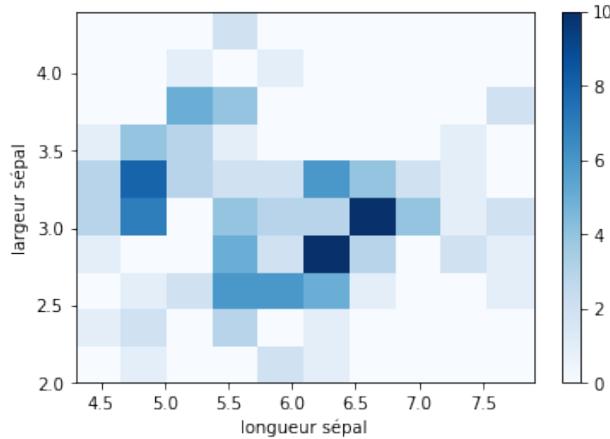


```
[12]: x = iris.data

plt.hist2d(x[:,0], x[:,1], cmap='Blues')
plt.xlabel('longueur sépal')
plt.ylabel('largeur sépal')
```

```
plt.colorbar()
```

[12]: <matplotlib.colorbar.Colorbar at 0x7ff393c2b8d0>

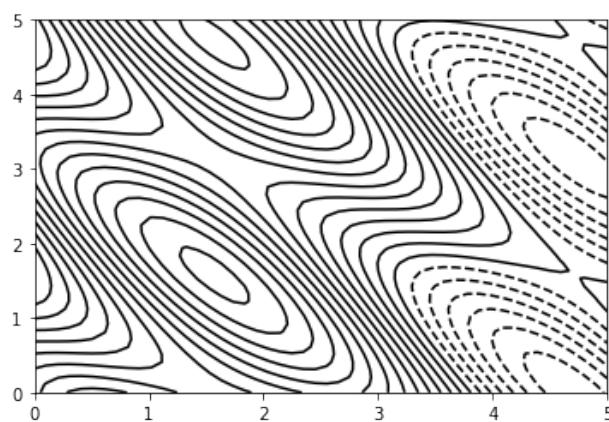


### Graphiques ContourPlot()

```
f = lambda x, y: np.sin(x) + np.cos(x+y)*np.cos(x+y)

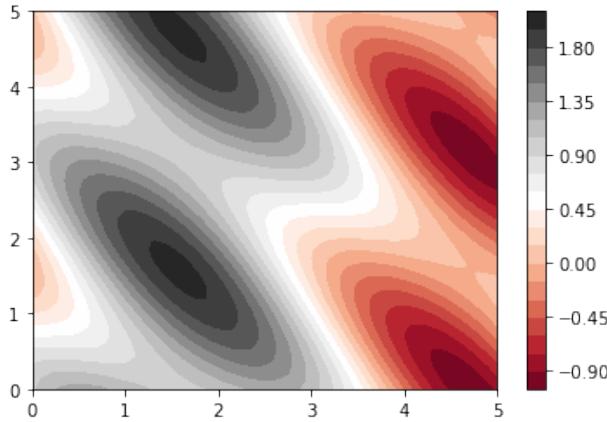
X = np.linspace(0, 5, 50)
Y = np.linspace(0, 5, 50)
X, Y = np.meshgrid(X, Y)
Z = f(X, Y)

plt.contour(X, Y, Z, 20, colors='black')
```



```
plt.contourf(X, Y, Z, 20, cmap='RdGy')
plt.colorbar()
```

[32]: <matplotlib.colorbar.Colorbar at 0x7fbde238a250>



### Imshow()

```
[33]: plt.figure(figsize=(12, 3))

# Simple graphique imshow()
X = np.random.randn(50, 50)

plt.subplot(131)
plt.imshow(X)
plt.title('random normal')

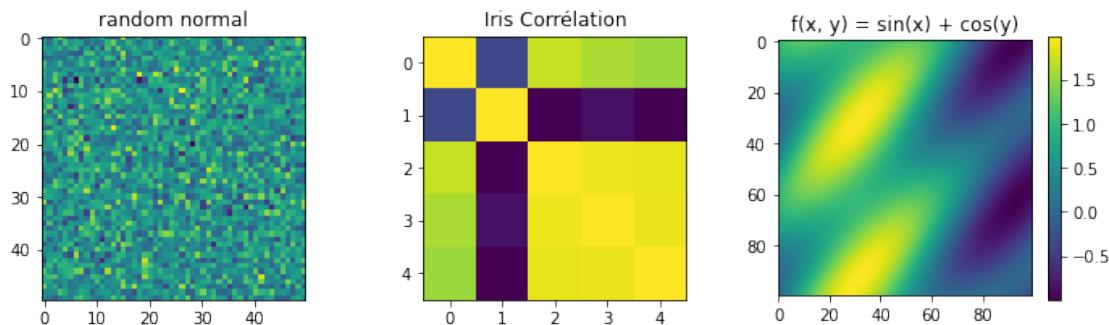
# Matrice de corrélation des iris
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
y = iris.target

plt.subplot(132)
plt.imshow(np.corrcoef(X.T, y))
plt.title('Iris Corrélation')

# Matrice  $f(X, Y) = \sin(X) + \cos(Y)$ 
X = np.linspace(0, 5, 100)
Y = np.linspace(0, 5, 100)
X, Y = np.meshgrid(X, Y)

plt.subplot(133)
plt.imshow(f(X, Y))
plt.colorbar()
plt.title('f(x, y) = sin(x) + cos(y)')
```

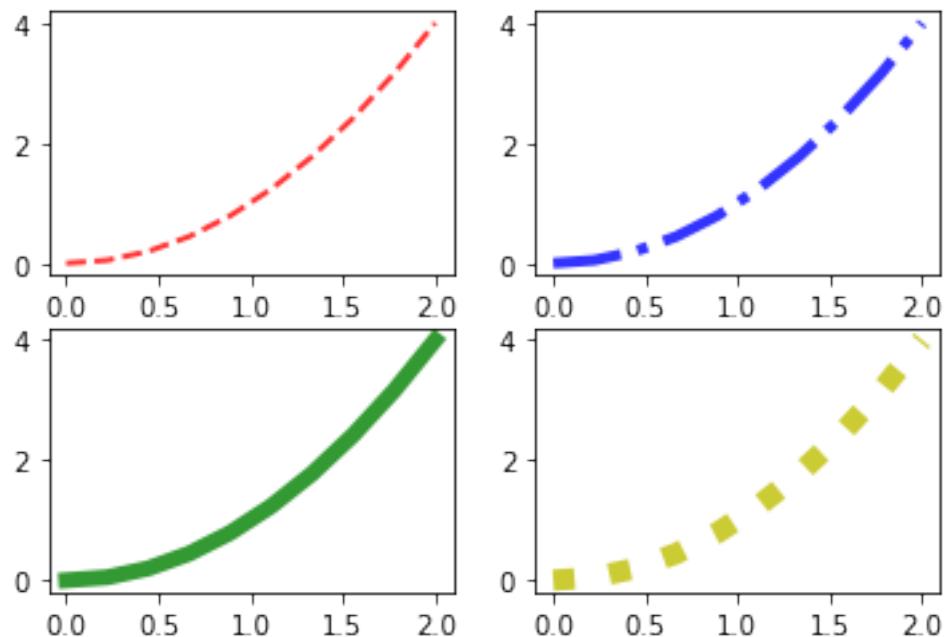
[33]: Text(0.5, 1.0, 'f(x, y) = sin(x) + cos(y)')



## 2.2.8 Exercices

**Exercice 1** Ecrire un programme qui dessine une grille de graphiques 2X2 avec différentes largeurs de lignes, couleurs et styles.

```
[15]: X = np.linspace(0, 2, 10)
y = X**2
```



**Exercice 2** Créez une fonction “graphique” qui permet de tracer sur une seule et même figure une série de graphiques issue d'un dictionnaire contenant plusieurs datasets :

```
[19]: def graphique(dataset):
    # Votre code ici...
    return

    # Voici le dataset utilisé
dataset = {f"experience{i}": np.random.randn(100) for i in range(4)}
```

```
[20]: # SOLUTION
def graphique(data):

    #<-----Votre code ici ----->

    plt.show()
```

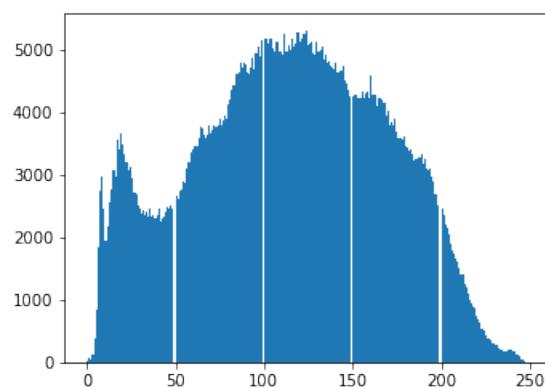
```
[21]: graphique(dataset)
```

### Exercice

Dans cet exercice, nous voulons afficher l'image ainsi que l'histogramme de l'échelle de gris comme indiqué sur la figure.

```
[22]: # histogramme d'une image
from scipy import misc
face = misc.face(gray=True)

plt.figure(figsize=(12, 4))
plt.subplot(121)
# votre code ici
plt.subplot(122)
# votre code ici NB: utiliser la fonction reveal()
plt.show()
```



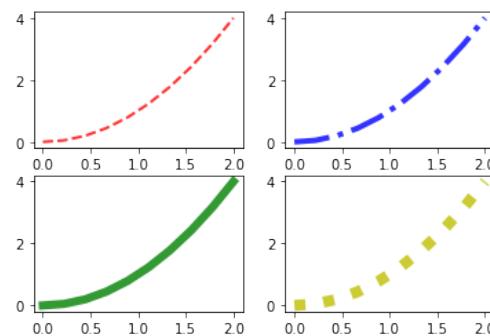
### 2.2.9 Solution au exercices

**Exercices 1** Ecrire un programme qui dessine une grille de graphiques 2X2 avec différentes largeurs de lignes, couleurs et styles.

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

```
[15]: X = np.linspace(0, 2, 10)
y = X**2
```

```
plt.figure()
plt.subplot(2, 2, 1)
plt.plot(X, y, c='r', lw=2, ls='--', alpha=0.8)
plt.subplot(2, 2, 2)
plt.plot(X, y, c='b', lw=4, ls='dashdot', alpha=0.8)
plt.subplot(2, 2, 3)
plt.plot(X, y, c='g', lw=6, ls='-', alpha=0.8)
plt.subplot(2, 2, 4)
plt.plot(X, y, c='y', lw=8, ls=':', alpha=0.8)
plt.show()
```

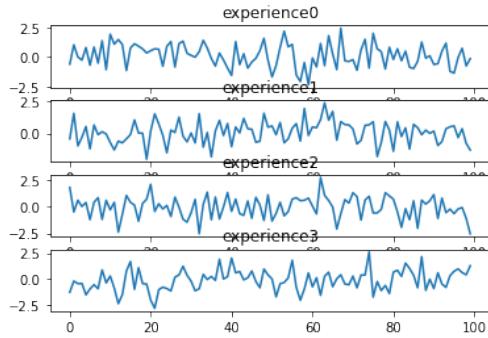


**Exercice 2** Créez une fonction “graphique” qui permet de tracer sur une seule et même figure une série de graphiques issue d’un dictionnaire contenant plusieurs datasets :

```
[22]: def graphique(dataset):
    plt.figure()
    n = len(dataset)
    for k, i in zip(dataset.keys(), range(1, n+1)):
        plt.subplot(n, 1, i)
        plt.plot(dataset[k])
        plt.title(k)
    return

# Voici le dataset utilisé
dataset = {f"experience{i}": np.random.randn(100) for i in range(4)}

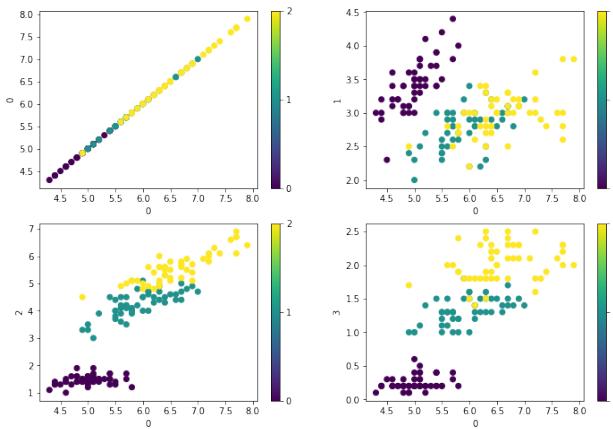
len(dataset)
graphique(dataset)
```



```
[26]: import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
iris = load_iris()
x = iris.data
y = iris.target
print(x.shape[0],x.shape[1],np.unique(y).size)
```

150 4 3

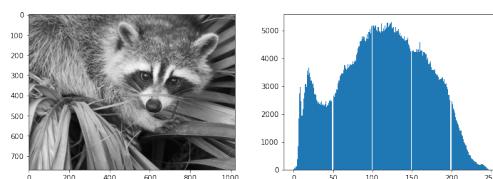
```
[31]: plt.figure(figsize=(12, 8))
n=x.shape[1]
for i in range(n):
    plt.subplot(n//2,n//2,i+1)
    plt.scatter(x[:,0],x[:,i], c=y)
    plt.xlabel('0')
    plt.ylabel(i)
    plt.colorbar(ticks=list(np.unique(y)))
plt.show()
```



**Exercice3** Dans cet exercice, nous voulons afficher l'image ainsi que l'histogramme de l'échelle de gris comme indiqué sur la figure.

```
[32]: # histogramme d'une image
from scipy import misc
face = misc.face(gray=True)
```

```
plt.figure(figsize=(12, 4))
plt.subplot(121)
plt.imshow(face, cmap='gray')
plt.subplot(122)
plt.hist(face.ravel(), bins=255)
plt.show()
```



## 2.3 Scipy

Scipy contient des modules tres puissants pour le machine learning, l'anayse de données, les time series, etc. Ce notebook vous montre quelques unes des fonctions les plus utiles

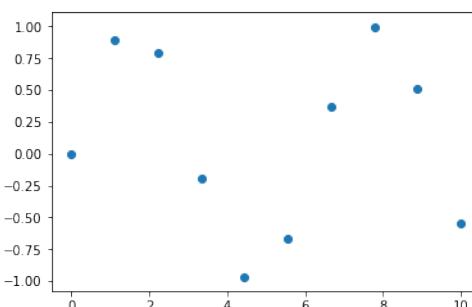
```
[2]: import numpy as np
import matplotlib.pyplot as plt
```

### 2.3.1 Interpolation

Interpoler un signal est parfois tres utile s'il vous manque des données dans un Dataset. Mais c'est une technique dangereuse, qui peut parfois transformer la réalité des choses ! Dans cette partie, nous allons interpoler les données de cet échantillon de données.

```
[2]: # Création d'un Dataset
x = np.linspace(0, 10, 10)
y = np.sin(x)
plt.scatter(x, y)
```

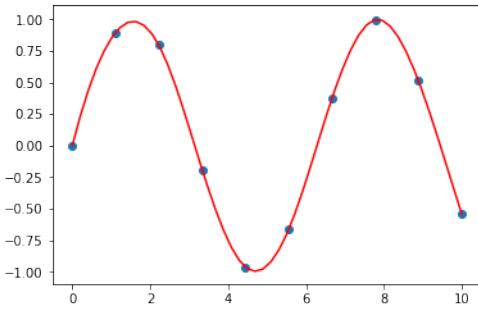
```
[2]: <matplotlib.collections.PathCollection at 0x1da399b0ac8>
```



[3]: `from scipy.interpolate import interp1d`

[4]: `# 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)`  
`result = f(new_x)`  
`# visualisation avec matplotlib`  
`plt.scatter(x, y)`  
`plt.plot(new_x, result, c='r')`

[4]: [`<matplotlib.lines.Line2D at 0x1da399de128>`]



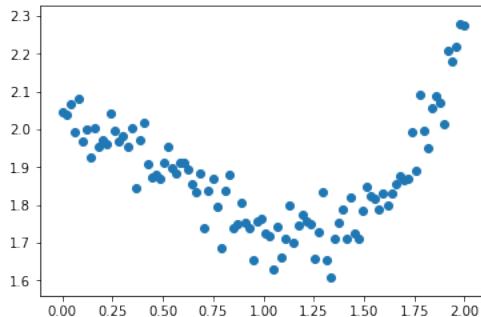
### 2.3.2 Optimisation

On trouve beaucoup de fonctions dans le module **optimize**. Certaines permettent de faire des minimisations locales, ou globales, d'autres permettent de développer des modèles statistiques avec la méthode des moindres carrés. On trouve également des fonctions pour faire de la programmation linéaire.

#### curve\_fit

[5]: `# Création d'un Dataset avec du bruit "normal"`  
`x = np.linspace(0, 2, 100)`  
`y = 1/3*x**3 - 3/5 * x**2 + 2 + np.random.randn(x.shape[0])/20`  
`plt.scatter(x, y)`

[5]: `<matplotlib.collections.PathCollection at 0x1da3a0bd940>`



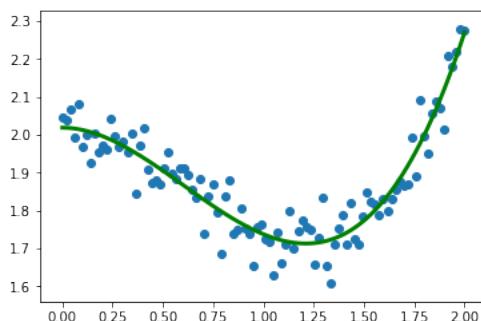
```
[6]: # Définition d'un modèle statistique sensé "coller" au dataset ci-dessus
def f (x, a, b, c, d):
    return a * x**3 + b * x**2 + c * x + d
```

```
[7]: from scipy import optimize
```

```
[8]: # curve_fit permet de trouver les paramètres du modèle f grâce à la méthode des moindres carrés
params, param_cov = optimize.curve_fit(f, x, y)
```

```
[9]: # Visualisation des résultats.
plt.scatter(x, y)
plt.plot(x, f(x, params[0], params[1], params[2], params[3]), c='g', lw=3)
```

```
[9]: [matplotlib.lines.Line2D at 0x1da3a37c160]
```



## Minimisation 1D

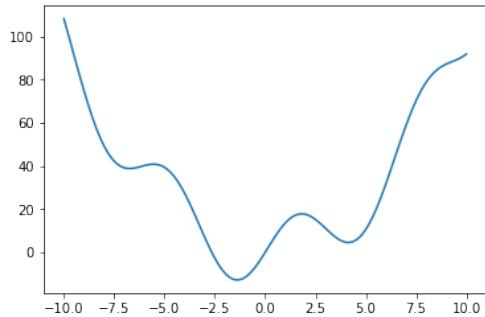
la fonction **optimize.minimize** est utile pour trouver un minimum local dans une fonction à N dimensions

```
[10]: # Définition d'une fonction à 1 Dimension
def f (x):
    return x**2 + 15*np.sin(x)
```

```
[11]: # Visualisation de la fonction
x = np.linspace(-10, 10, 100)
```

```
plt.plot(x, f(x))
```

[11]: [`<matplotlib.lines.Line2D at 0x1da3a81ec88>`]

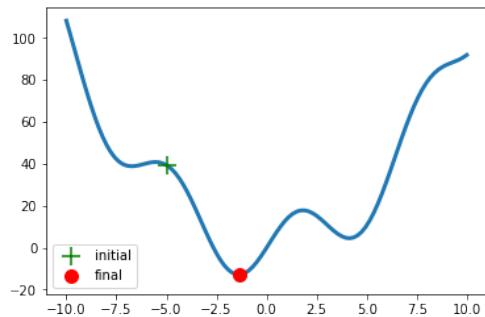


[12]: # Définition d'un point  $x_0$  pour l'algorithme de minimisation

```
x0=-5
result = optimize.minimize(f, x0=x0).x # résultat de la minimisation
```

[13]: # Visualisation du résultat

```
plt.plot(x, f(x), lw=3, zorder=-1) # Courbe de la fonction
plt.scatter(x0, f(x0), s=200, marker='+', c='g', zorder=1, label='initial') # point initial
plt.scatter(result, f(result), s=100, c='r', zorder=1, label='final') # point final
plt.legend()
plt.show()
```



## Minimisation 2D

[14]: # Définition d'une fonction 2D. X est un tableau numpy à 2-Dimension

```
def f (x):
    return np.sin(x[0]) + np.cos(x[0]+x[1])*np.cos(x[0])
```

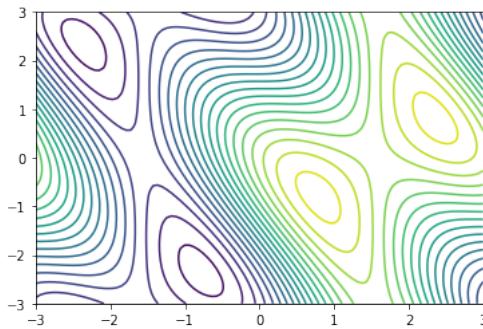
[15]: # Génération de la fonction sur un espace 2D.

```
x = np.linspace(-3, 3, 100)
y = np.linspace(-3, 3, 100)
```

```
x, y = np.meshgrid(x, y)

# Visualisation de la fonction
plt.contour(x, y, f(np.array([x, y])), 20)
```

[15]: <matplotlib.contour.QuadContourSet at 0x1da3a907630>

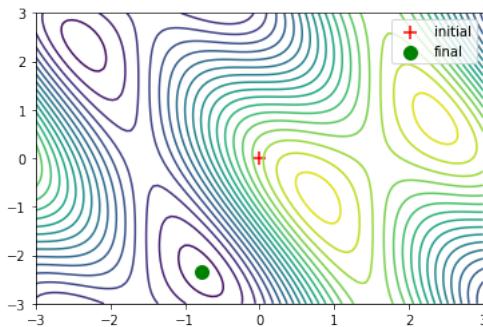


```
# Placement d'un point x0 initial aux coordonées (0, 0)
x0 = np.zeros((2, 1))

# Minimisation de la fonction
result = optimize.minimize(f, x0=x0).x
print('le minimum est aux coordonées', result) # imprimer le résultat

# Visualisation du résultat
plt.contour(x, y, f(np.array([x, y])), 20) # fonction 2D
plt.scatter(x0[0], x0[1], marker='+', c='r', s=100, label='initial') # Point de départ
plt.scatter(result[0], result[1], c='g', s=100, label='final') # Point final
plt.legend()
plt.show()
```

le minimum est aux coordonées [-0.78539917 -2.35619341]



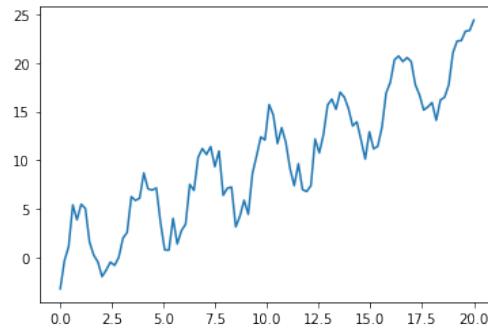
### 2.3.3 Traitement du signal

Le module **scipy.signal** contient beaucoup de fonctions de convolution et de filtres pour faire du traitement du signal. La fonction **signal.detrend** est parfaite pour éliminer une tendance linéaire dans un signal. Utile pour beaucoup d'applications !

Le module **scipy.fftpack** contient des fonctions très puissantes et simples d'utilisation pour effectuer des transformations de Fourier

```
[3]: # Création d'un Dataset avec une tendance linéaire
x = np.linspace(0, 20, 100)
y = x + 8*np.sin(x)*np.cos(x) + np.random.randn(x.shape[0])
plt.plot(x, y)
```

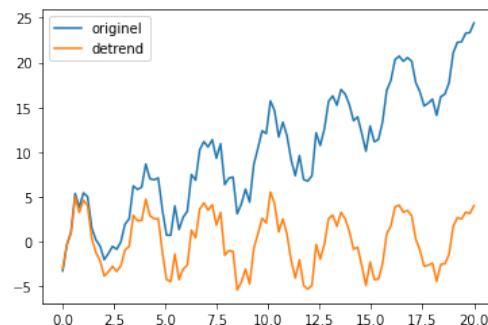
[3]: [<matplotlib.lines.Line2D at 0x7f98d93bfd90>]



```
[4]: from scipy import signal
```

```
[5]: # Élimination de la tendance linéaire
new_y = signal.detrend(y)

# Visualisation des résultats
plt.plot(x, y, label='originel')
plt.plot(x, new_y, label='detrend')
plt.legend()
plt.show()
```



### 2.3.4 Transformation de Fourier (FFT)

La transformation de Fourier est une technique mathématique puissante et normalement complexe à mettre en œuvre. Heureusement `scipy.fftpack` rend cette technique très simple à implémenter.

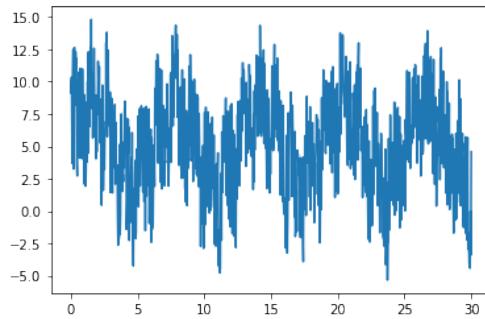
La transformation de Fourier permet d'analyser les **fréquences** qui composent un signal **périodique** (qui se répète avec le temps). Cette opération produit un graphique que l'on appelle **Spectre**.

Une fois le **Spectre** généré, il est possible de filtrer les bruits indésirables, ou bien de sélectionner seulement certaines fréquences, ou d'en atténuer d'autres... les possibilités sont infinies.

Dans l'exemple ci-dessous, nous voyons comment filtrer un signal noyé dans du bruit.

```
[6]: # Création d'un signal périodique noyé dans du bruit.
x = np.linspace(0, 30, 1000)
y = 3*np.sin(x) + 2*np.sin(5*x) + np.sin(10*x) + np.random.random(x.shape[0])*10
plt.plot(x, y)
```

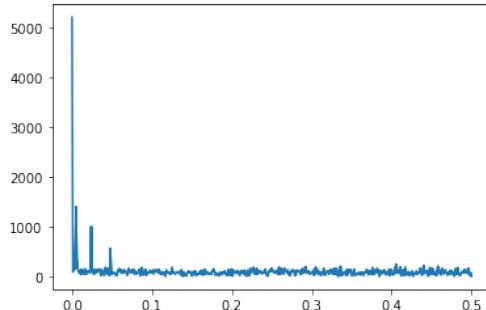
[6]: [<matplotlib.lines.Line2D at 0x7f98da3716d0>]



```
[7]: from scipy import fftpack
```

```
[8]: # création des variables Fourier et Fréquences, qui permettent de construire le spectre du signal.
fourier = fftpack.fft(y)
power = np.abs(fourier) # la variable power est créée pour éliminer les amplitudes négatives
fréquences = fftpack.fftfreq(y.size)
plt.plot(np.abs(fréquences), power)
```

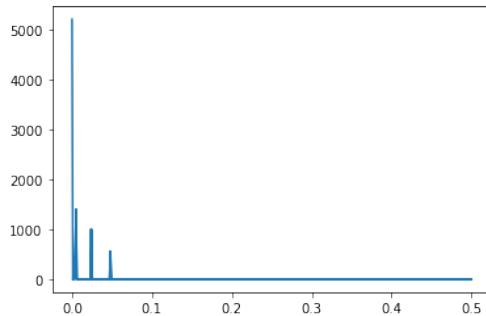
[8]: [<matplotlib.lines.Line2D at 0x7f98da5f6040>]



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

# Visualisation du spectre propre
plt.plot(np.abs(frequences), np.abs(fourier))
```

[9]: [<matplotlib.lines.Line2D at 0x7f98da704580>]

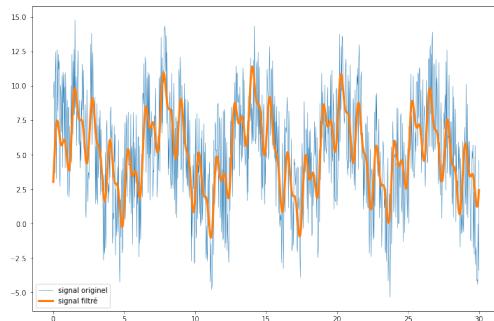


```
[10]: # Transformation de Fourier Inverse: genere un nouveau signal temporel depuis le spectre filtré
filtered_signal = fftpack.ifft(fourier)
```

```
[11]: # Visualisation des résultats

plt.figure(figsize=(12, 8))
plt.plot(x, y, lw=0.5, label='signal originel')
plt.plot(x, filtered_signal, lw=3, label='signal filtré')
plt.legend()
plt.show()
```

```
/Users/mac/opt/anaconda3/lib/python3.8/site-
packages/matplotlib/cbook/__init__.py:1289: ComplexWarning: Casting complex
values to real discards the imaginary part
    return np.asarray(x, float)
```



### 2.3.5 Traitement d'image

`scipy.ndimage` propose de nombreuses actions pour le traitement d'images: convolutions, filtres de Gauss, méthode de mesures, et morphologie.

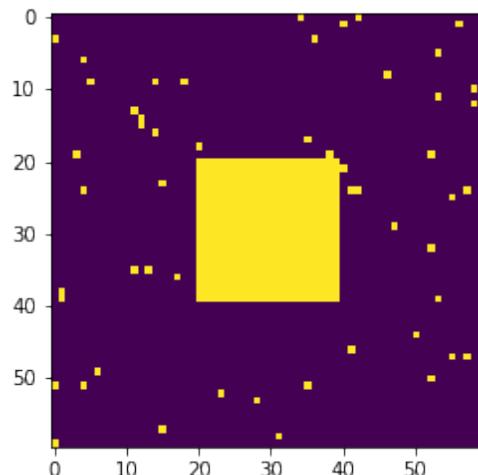
La morphologie est une technique qui permet de transformer une matrice (et donc une image) par le déplacement d'une structure sur chaque pixel de l'image. Lorsqu'un pixel "blanc" est visité, la structure peut effectuer une opération: - de dilation: imprime des pixels - d'érosion : efface des pixels

Cette technique peut-être utile pour nettoyer une image des artefacts qui peuvent la composer.

```
[12]: from scipy import ndimage
```

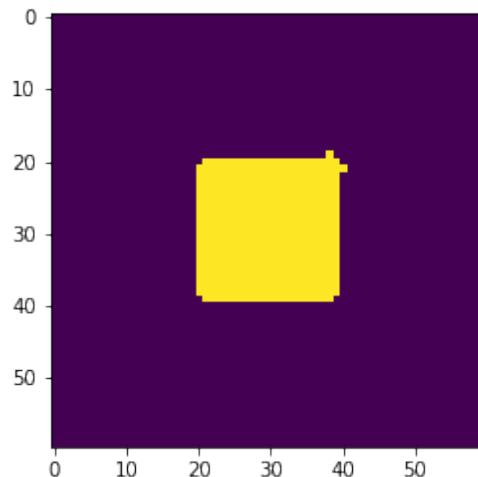
```
[18]: # Création d'une image avec quelques artefacts
np.random.seed(0)
X = np.zeros((60, 60))
X[20:-20, 20:-20] = 1
X[np.random.randint(0,60,60),np.random.randint(0,60,60)] = 1 #ajout d'artefacts
    ↪aléatoires
plt.imshow(X)
```

```
[18]: <matplotlib.image.AxesImage at 0x7f98db523850>
```



```
[19]: # opération de binary_opening = érosion puis dilation
open_X = ndimage.binary_opening(X)
plt.imshow(open_X)
```

[19]: <matplotlib.image.AxesImage at 0x7f98db8917f0>

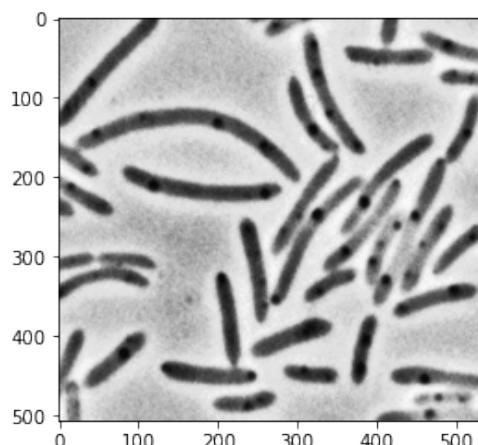


### 2.3.6 Application (cas réel)

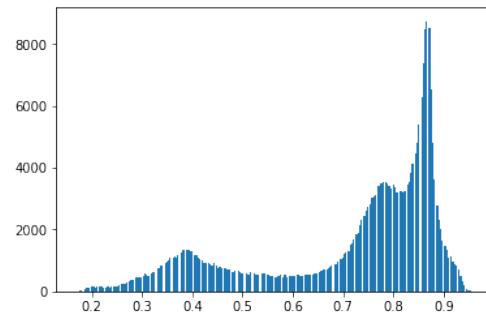
Vous trouvez cette image dans le pack de la formation:

```
[20]: # importer l'image avec pyplot
image = plt.imread('Data/bacteria.png')
image = image[:, :, 0] # réduire l'image en 2D
plt.imshow(image, cmap='gray') # afficher l'image
image.shape
```

[20]: (507, 537)

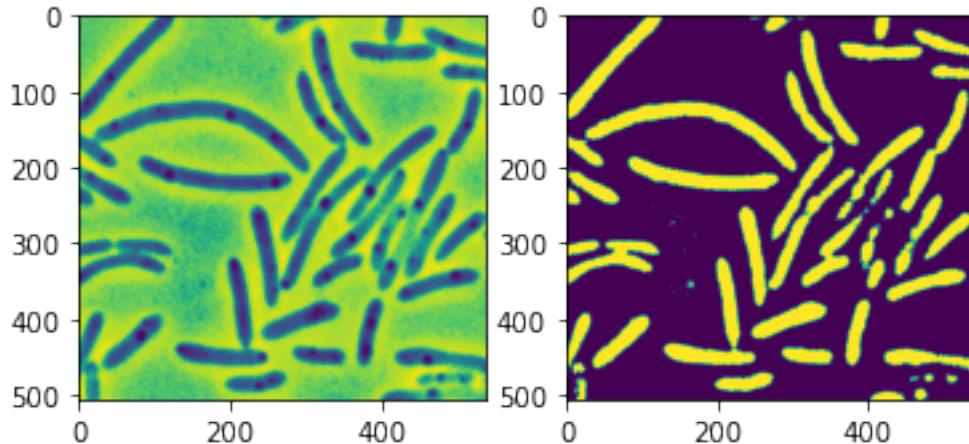


```
[30]: # copy de l'image, puis création d'un histogramme
image_2 = np.copy(image)
plt.hist(image_2.ravel(), bins=255)
plt.show()
```



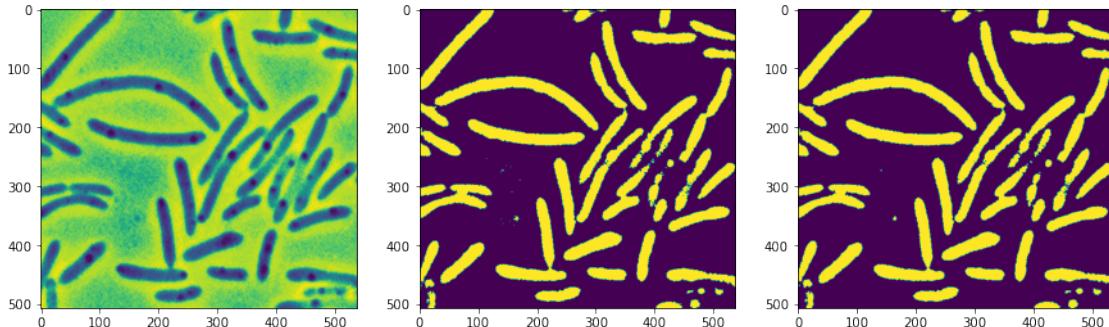
```
[23]: # boolean indexing: création d'une image binaire
image_clean= image<0.6
plt.subplot(1,2,1)
plt.imshow(image)
plt.subplot(1,2,2)
plt.imshow(image_clean)
```

[23]: <matplotlib.image.AxesImage at 0x7f98dc024400>



```
[27]: # morphologie utilisée pour enlever les artefacts
image_clean2 = ndimage.binary_opening(image_clean)
plt.figure(figsize=(15, 5))
plt.subplot(1,3,1)
plt.imshow(image)
plt.subplot(1,3,2)
plt.imshow(image_clean)
plt.subplot(1,3,3)
plt.imshow(image_clean2)
```

[27]: <matplotlib.image.AxesImage at 0x7f98db184760>

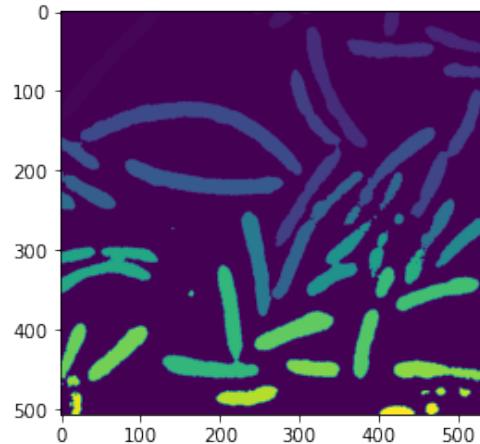


```
[29]: # Segmentation de l'image: label_image contient les différents labels et ↴
      n_labels est le nombre de labels
label_image, n_labels = ndimage.label(image_clean2)
print(f'il y a {n_labels} groupes')
```

il y a 53 groupes

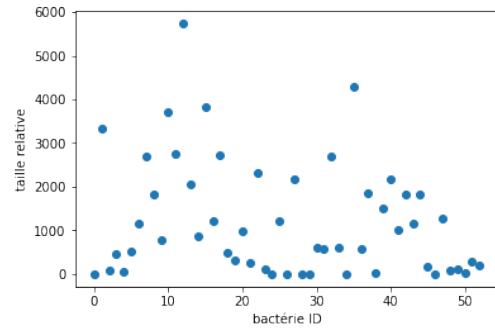
```
[30]: # Visualisation de l'image étiquetée
plt.imshow(label_image)
```

[30]: <matplotlib.image.AxesImage at 0x7f98dbb0f130>



```
[32]: # Mesure de la taille de chaque groupes de label_images (fait la somme des ↴
      pixels)
sizes = ndimage.sum(image_clean2, label_image, range(n_labels))
```

```
[33]: # Visualisation des résultats
plt.scatter(range(n_labels), sizes)
plt.xlabel('bactérie ID')
plt.ylabel('taille relative')
plt.show()
```





# Chapter 3

## Analyse de données et visualisation

### 3.1 Pandas

pandas est une bibliothèque logicielle écrite pour le langage de programmation Python pour la manipulation et l'analyse de données. En particulier, il propose des structures de données et des opérations de manipulation de tableaux numériques et de séries chronologiques.

```
[35]: import numpy as np  
import matplotlib.pyplot as plt  
import pandas as pd
```

#### 3.1.1 DataFrame Pandas

Données tabulaires bidimensionnelles, variables en taille et potentiellement hétérogènes. La structure de données contient également des axes étiquetés (lignes et colonnes). Les opérations arithmétiques s'alignent sur les étiquettes de ligne et de colonne. Peut être considéré comme un conteneur de type dict pour les objets Series. La structure de données principale des pandas

##### Charger vos données dans un

Les options les plus courantes : - read\_csv - read\_excel

```
[36]: data = pd.read_excel('Datasets/titanic3.xls')
```

```
[37]: data.shape  
data.head()
```

```
[37]:   pclass  survived          name      sex  \n 0      1        1  Allen, Miss. Elisabeth Walton  female\n 1      1        1  Allison, Master. Hudson Trevor    male\n 2      1        0  Allison, Miss. Helen Loraine  female\n 3      1        0  Allison, Mr. Hudson Joshua Creighton    male\n 4      1        0  Allison, Mrs. Hudson J C (Bessie Waldo Daniels)  female
```

```

      age  sibsp  parch  ticket      fare    cabin embarked boat   body \
0  29.0000     0      0   24160  211.3375      B5        S     2    NaN
1  0.9167     1      2  113781  151.5500     C22  C26        S    11    NaN
2  2.0000     1      2  113781  151.5500     C22  C26        S    NaN    NaN
3 30.0000     1      2  113781  151.5500     C22  C26        S    NaN  135.0
4 25.0000     1      2  113781  151.5500     C22  C26        S    NaN    NaN

                    home.dest
0                  St Louis, MO
1  Montreal, PQ / Chesterville, ON
2  Montreal, PQ / Chesterville, ON
3  Montreal, PQ / Chesterville, ON
4  Montreal, PQ / Chesterville, ON

```

[38] : `data.describe() # Avoir une analyse statistique des donnees`

```

[38]:      pclass      survived       age      sibsp      parch \
count  1309.000000  1309.000000  1046.000000  1309.000000  1309.000000
mean    2.294882    0.381971    29.881135    0.498854    0.385027
std     0.837836    0.486055    14.413500    1.041658    0.865560
min     1.000000    0.000000    0.166700    0.000000    0.000000
25%    2.000000    0.000000    21.000000    0.000000    0.000000
50%    3.000000    0.000000    28.000000    0.000000    0.000000
75%    3.000000    1.000000    39.000000    1.000000    0.000000
max    3.000000    1.000000    80.000000    8.000000    9.000000

          fare      body
count  1308.000000  121.000000
mean    33.295479  160.809917
std     51.758668  97.696922
min     0.000000  1.000000
25%    7.895800  72.000000
50%   14.454200 155.000000
75%   31.275000 256.000000
max   512.329200 328.000000

```

## Nettoyer des Datasets

La fonction “drop” supprime des lignes ou des colonnes en spécifiant les noms d’étiquette et l’axe correspondant, ou en spécifiant directement les noms d’index ou de colonne.

[39] : `data = data.drop(['name', 'sibsp', 'parch', 'ticket', 'fare', 'cabin',  
→'embarked', 'boat', 'body', 'home.dest'], axis=1)`

Remplir les valeurs vides (NA/Nan) à l’aide de la méthode spécifiée. Dans ce cas, les données vides sont remplies par des moyens.

```
[9]: data2= data.fillna(data['age'].mean())
print(data2.describe())
data2.shape
```

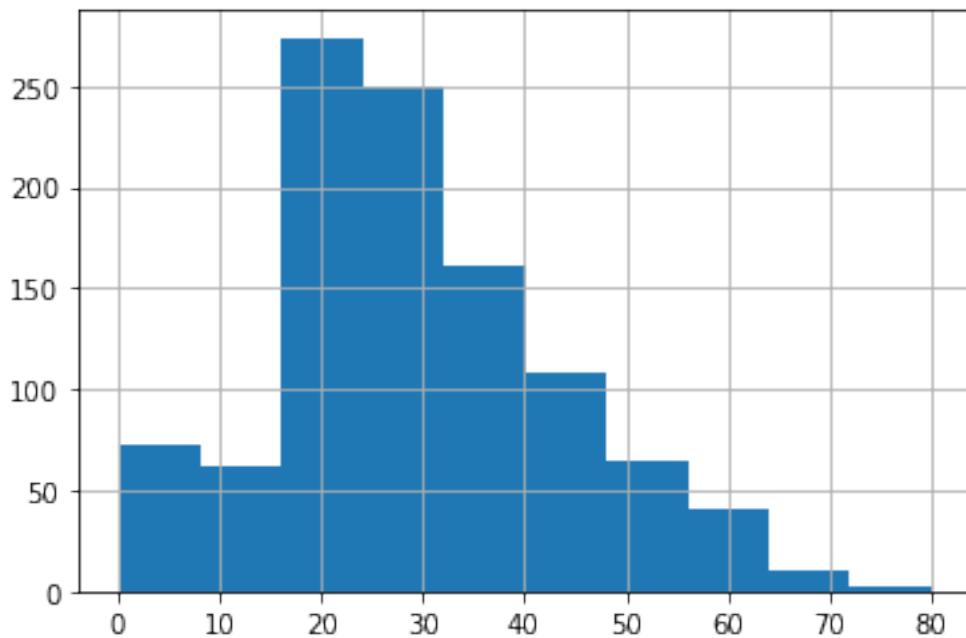
```
      pclass    survived       age
count  1309.000000  1309.000000  1309.000000
mean    2.294882    0.381971   29.881135
std     0.837836    0.486055   12.883199
min     1.000000    0.000000   0.166700
25%    2.000000    0.000000   22.000000
50%    3.000000    0.000000   29.881135
75%    3.000000    1.000000   35.000000
max    3.000000    1.000000   80.000000
```

Où nous les éliminerons tous.

```
[40]: data = data.dropna(axis=0)
print(data.describe())
data.shape
```

```
      pclass    survived       age
count  1046.000000  1046.000000  1046.000000
mean    2.207457    0.408222   29.881135
std     0.841497    0.491740   14.413500
min     1.000000    0.000000   0.166700
25%    1.000000    0.000000   21.000000
50%    2.000000    0.000000   28.000000
75%    3.000000    1.000000   39.000000
max    3.000000    1.000000   80.000000
```

```
[41]: data['age'].hist()
```



### 3.1.2 Statistics avec Groupby() et value\_counts()

Une opération groupby implique une combinaison de fractionnement de l'objet, d'application d'une fonction et de combinaison des résultats. Cela peut être utilisé pour regrouper de grandes quantités de données et d'opérations de calcul sur ces groupes.

```
[42]: data.groupby(['sex']).mean()
```

```
[42]:      pclass  survived      age
sex
female  2.048969  0.752577  28.687071
male    2.300912  0.205167  30.585233
```

On observe que plus de femmes ont survécu au naufrage du Titanic.

```
[28]: data.groupby(['sex', 'pclass']).mean()
```

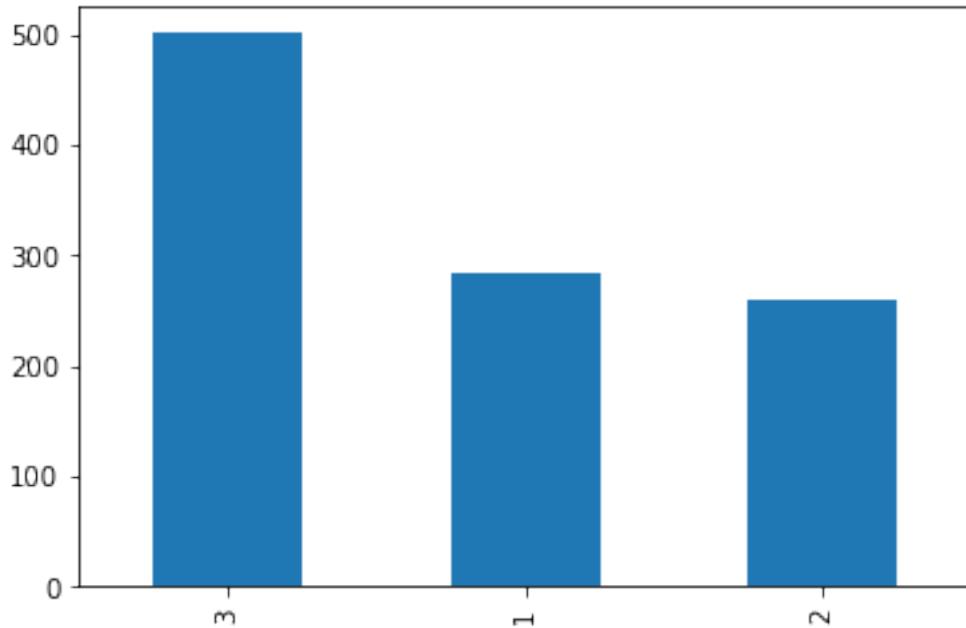
```
[28]:      survived      age
sex   pclass
female 1      0.962406  37.037594
      2      0.893204  27.499191
      3      0.473684  22.185307
male   1      0.350993  41.029250
      2      0.145570  30.815401
      3      0.169054  25.962273
```

```
[21]: data['pclass'].value_counts()
```

```
[21]: 3    501
      1    284
      2    261
Name: pclass, dtype: int64
```

```
[12]: data['pclass'].value_counts().plot.bar()
```

```
[12]: <AxesSubplot:>
```



Nous voulons obtenir le nombre de passagers par classe qui avaient moins de 18 ans.

```
[43]: data[data['age'] < 18]['pclass'].value_counts()
```

```
[43]: 3    106
      2    33
      1    15
Name: pclass, dtype: int64
```

```
[48]: data[data['age']>18].groupby(['sex', 'pclass']).mean()
```

```
[48]:          survived      age
sex   pclass
female 1      0.966667  39.358333
      2      0.878049  32.067073
      3      0.436170  29.457447
male   1      0.328671  42.716783
      2      0.087591  34.069343
      3      0.158845  29.799639
```

### 3.1.3 Opération sur les série

Ndarray unidimensionnel avec étiquettes d'axe (y compris les séries chronologiques).

Les étiquettes n'ont pas besoin d'être uniques, mais doivent être de type hachable. L'objet prend en charge l'indexation basée à la fois sur les entiers et sur les étiquettes et fournit une multitude de méthodes pour effectuer des opérations impliquant l'index. Les méthodes statistiques de ndarray ont été remplacées pour exclure automatiquement les données manquantes (actuellement représentées par NaN).

```
[20]: #data['age'] est une serie
print(data['age'][0:10]) # slicing a serie
data2 = data['age']> 18 # cree un mask
print(data2)
print(data[data['age']< 18]) # boolean indexing
```

```
0    29.0000
1    0.9167
2    2.0000
3    30.0000
4    25.0000
5    48.0000
6    63.0000
7    39.0000
8    53.0000
9    71.0000
Name: age, dtype: float64
0      True
1     False
2     False
3      True
4      True
...
1301   True
1304   False
1306   True
1307   True
1308   True
Name: age, Length: 1046, dtype: bool
   pclass  survived     sex      age
1       1        1   male  0.9167
2       1        0 female  2.0000
53      1        0   male  17.0000
54      1        1   male  11.0000
55      1        1 female  14.0000
...     ...
1265     3        0 female  10.0000
1275     3        0   male  16.0000
1279     3        0 female  14.0000
```

```
1300      3      1  female  15.0000
1304      3      0  female  14.5000
```

[154 rows x 4 columns]

[52]: `data.head()`

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

### 3.1.4 Méthodes loc et iloc

Les méthodes loc et iloc sont des méthodes Pandas essentielles utilisées pour filtrer, sélectionner et manipuler des données. Ils nous permettent d'accéder à la combinaison souhaitée de lignes et de colonnes.

La principale différence entre eux est la façon dont ils accèdent aux lignes et aux colonnes :

- loc utilise des étiquettes de ligne et de colonne.
- iloc utilise des index de ligne et de colonne.

[60]: `print(data.iloc[3,3]) # iloc pour la localisation d'index récupère la valeur de la 4ème ligne et de la 4ème colonne`  
`print(data.iloc[0,3]) # récupère la valeur de la première ligne et de la 4ème colonne`  
`print(data.loc[0:2, 'age'])`

```
30.0
29.0
0    29.0000
1    0.9167
2    2.0000
Name: age, dtype: float64
```

### 3.1.5 Codification des données

Dans cette partie nous allons créer des catégories et codifier les données string avec les fonctions map(), replace() et cat.codes.

[12]: `import numpy as np`  
`import matplotlib.pyplot as plt`  
`import pandas as pd`

```
[46]: data = pd.read_excel('Datasets/titanic3.xls')
data = data.drop(['name', 'sibsp', 'parch', 'ticket', 'fare', 'cabin', 'embarked', 'boat', 'body', 'home.dest'], axis=1)
data = data.dropna(axis=0)
```

```
[41]: data.head()
```

```
[41]:   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
```

Pour créer des catégories d'âge comme indiqué dans l'exemple, il existe deux méthodes, nous utilisons soit l'indexation boléenne, soit la fonction de carte qui est illustrée ci-dessous

```
[ ]: data.loc[data['age'] < 20, 'age']=0
data.loc[(data['age'] >= 20) & (data['age'] < 30), 'age']=1
data.loc[(data['age'] >= 30) & (data['age'] < 40), 'age']=2
data.loc[data['age'] >= 40, 'age']=3
data.head()
```

```
[17]: data['age'].value_counts()
```

```
[17]: 1.0    344
3.0    245
2.0    232
0.0    225
Name: age, dtype: int64
```

```
[18]: data.groupby(['age']).mean()
```

```
[18]:      pclass  survived
age
0.0  2.542222  0.471111
1.0  2.436047  0.369186
2.0  2.103448  0.422414
3.0  1.677551  0.391837
```

La fonction map() applique une fonction sur tous les éléments d'une colonne.

```
[34]: data['age'].map(lambda x:x+2)
```

```
[34]: 0      31.0000
1      2.9167
2      4.0000
3      32.0000
```

```

4      27.0000
...
1301   47.5000
1304   16.5000
1306   28.5000
1307   29.0000
1308   31.0000
Name: age, Length: 1046, dtype: float64

```

```
[31]: def category_ages(age):
    if age <= 20:
        return '<20 ans'
    elif (age > 20) & (age <= 30):
        return '20-30 ans'
    elif (age > 30) & (age <= 40):
        return '30-40 ans'
    else:
        return '+40 ans'
```

```
[35]: data['age'] = data['age'].map(category_ages)
```

```
[36]: data.head()
```

```
[36]:   pclass  survived     sex       age
 0      1         1  female  20-30 ans
 1      1         1  male    <20 ans
 2      1         0  female  <20 ans
 3      1         0  male   20-30 ans
 4      1         0  female  20-30 ans
```

De plus, ici, la colonne sex contient des informations au format String, ce qui est inhabituel et inadapté à une application d'apprentissage automatique. Ainsi, nous utiliserons une codification pour convertir cette colonne au format numérique. Pour cette tâche, il existe trois méthodes adaptées à cette tâche : la fonction map(), replace() et cat.codes

```
[38]: def Conv_sex_to_number(sex):
    if sex == 'female':
        return 0
    else :
        return 1
data['sex'] = data['sex'].map(Conv_sex_to_number)
data.head()
```

```
[38]:   pclass  survived  sex       age
 0      1         1  0  20-30 ans
 1      1         1  1  <20 ans
 2      1         0  0  <20 ans
```

```
3      1      0      1  20-30 ans
4      1      0      0  20-30 ans
```

```
[43]: data['sex'] = data['sex'].map({'male':0,'female':1})
data.head()
```

```
[43]:   pclass  survived  sex  age
0      1          1    1  1.0
1      1          1    0  0.0
2      1          0    1  0.0
3      1          0    0  2.0
4      1          0    1  1.0
```

```
[45]: data['sex'] = data['sex'].replace(['male','female'],[0,1])
data.head()
```

```
[45]:   pclass  survived  sex      age
0      1          1    1  29.0000
1      1          1    0  0.9167
2      1          0    1  2.0000
3      1          0    0  30.0000
4      1          0    1  25.0000
```

```
[51]: data['sex']= data['sex'].astype('category').cat.codes
data.head()
```

```
[51]:   pclass  survived  sex      age
0      1          1    0  29.0000
1      1          1    1  0.9167
2      1          0    0  2.0000
3      1          0    1  30.0000
4      1          0    0  25.0000
```

```
[ ]:
```

### 3.1.6 Séries temporelles

Développées à l'origine pour les séries chronologiques financières telles que les cours boursiers quotidiens, les structures de données robustes et flexibles des pandas peuvent être appliquées aux données de séries chronologiques dans n'importe quel domaine, y compris la science, l'ingénierie, la santé publique et bien d'autres. Avec ces outils, vous pouvez facilement organiser, transformer, analyser et visualiser vos données à n'importe quel niveau de granularité - en examinant les détails pendant des périodes d'intérêt spécifiques et en effectuant un zoom arrière pour explorer les variations sur différentes échelles de temps, telles que les agrégations mensuelles ou annuelles, récurrentes modèles et tendances à long terme.

Dans la définition la plus large, une série chronologique est un ensemble de données où les valeurs sont mesurées à différents moments dans le temps. De nombreuses séries chronologiques

sont espacées de manière uniforme à une fréquence spécifique, par exemple, des mesures météorologiques horaires, des comptages quotidiens de visites de sites Web ou des totaux de ventes mensuels. Les séries chronologiques peuvent également être irrégulièrement espacées et sporadiques, par exemple, des données horodatées dans le journal des événements d'un système informatique ou un historique des appels d'urgence au 911. Les outils de séries chronologiques de Pandas s'appliquent aussi bien aux deux types de séries chronologiques.

```
[ ]: import numpy as np
      import matplotlib.pyplot as plt
      import pandas as pd
```

### Chargement d'une Séries temporelles

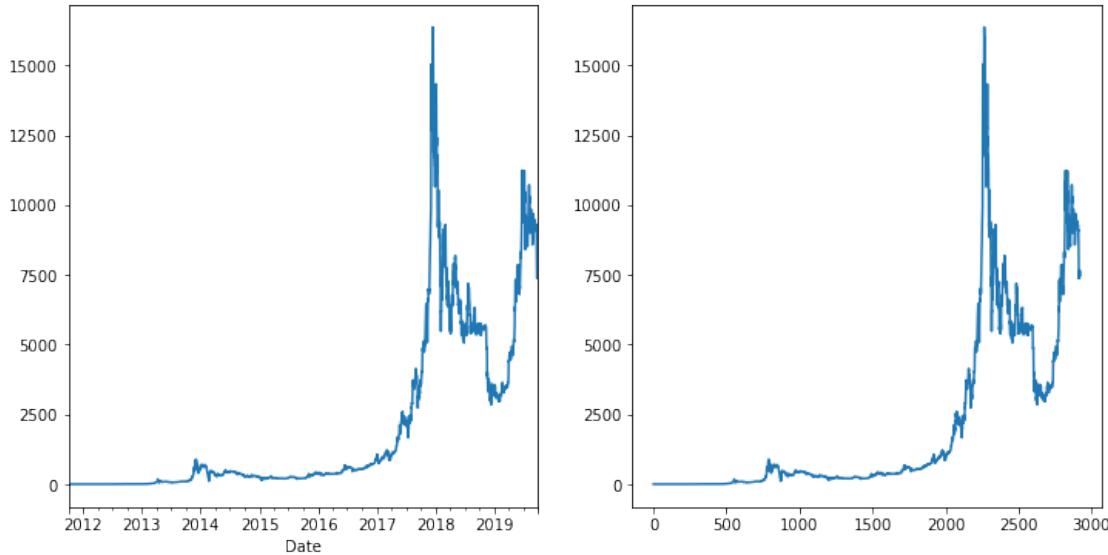
```
[4]: bitcoin2 = pd.read_csv('Datasets/BTC-EUR.csv')
      bitcoin2.head()

      bitcoin = pd.read_csv('Datasets/BTC-EUR.csv', index_col='Date', parse_dates=True)
      bitcoin.head()
```

```
[4]:
```

	Open	High	Low	Close	Adj Close	Volume
Date						
2011-10-04	3.700	3.821	3.746	3.750	3.750	1357
2011-10-05	3.750	3.820	3.650	3.676	3.676	3349
2011-10-06	3.676	3.743	3.450	3.550	3.550	6642
2011-10-07	3.550	3.590	2.900	3.293	3.293	7135
2011-10-08	3.293	3.283	2.872	2.890	2.890	2007

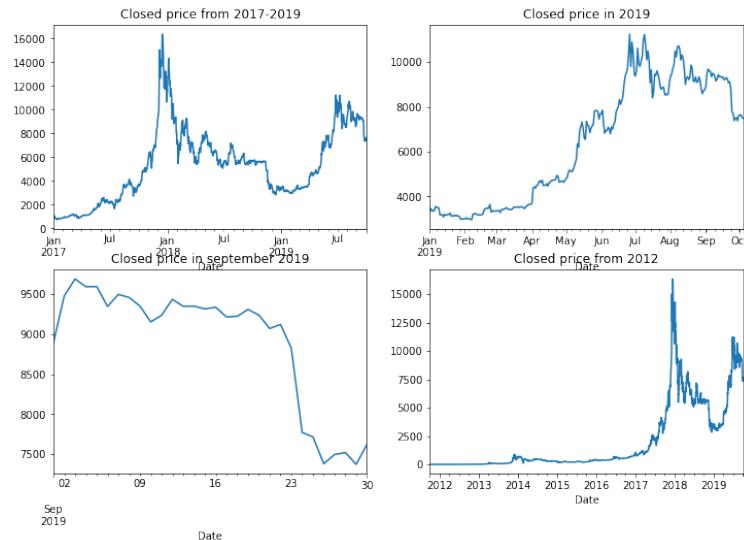
```
[7]: plt.figure(figsize=(12, 6))
      plt.subplot(1,2,1)
      bitcoin['Close'].plot()
      plt.subplot(1,2,2)
      bitcoin2['Close'].plot()
      plt.show()
```



```
[5]: bitcoin.index
```

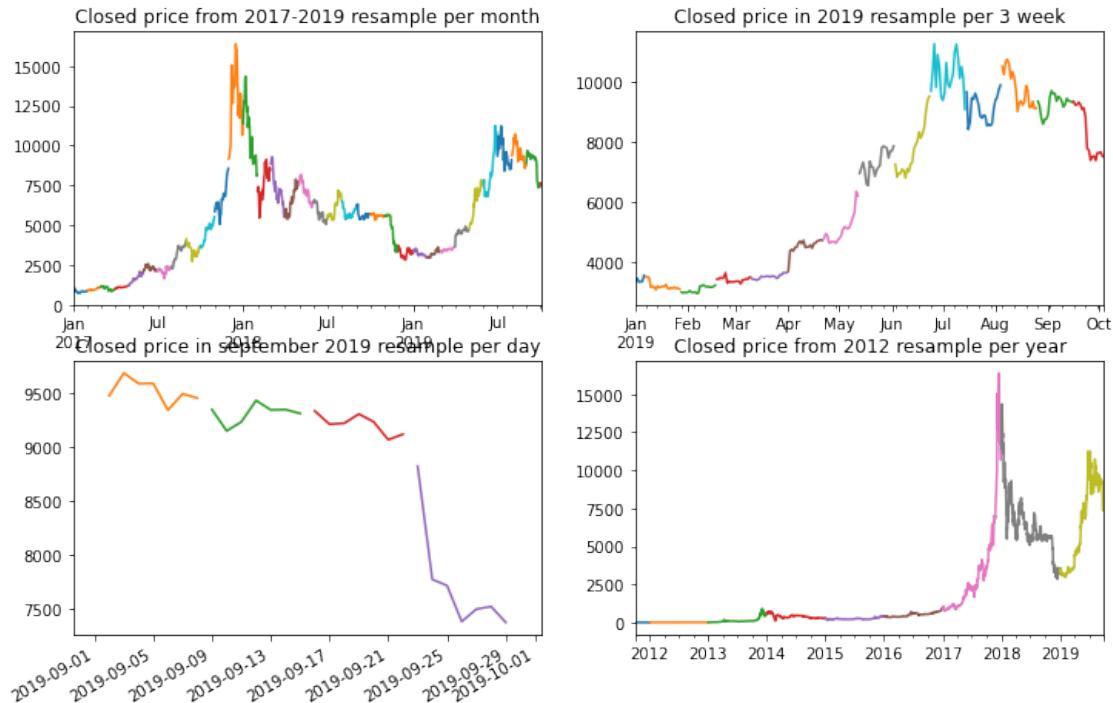
```
[5]: DatetimeIndex(['2011-10-04', '2011-10-05', '2011-10-06', '2011-10-07',
                   '2011-10-08', '2011-10-09', '2011-10-10', '2011-10-11',
                   '2011-10-12', '2011-10-13',
                   ...
                   '2019-09-25', '2019-09-26', '2019-09-27', '2019-09-28',
                   '2019-09-29', '2019-09-30', '2019-10-01', '2019-10-02',
                   '2019-10-03', '2019-10-04'],
                  dtype='datetime64[ns]', name='Date', length=2923, freq=None)
```

```
[13]: plt.figure(figsize=(12, 8))
plt.subplot(2,2,1)
bitcoin['2017':'2019']['Close'].plot()
plt.title("Closed price from 2017-2019")
plt.subplot(2,2,2)
bitcoin['2019']['Close'].plot()
plt.title("Closed price in 2019")
plt.subplot(2,2,3)
bitcoin['2019-09']['Close'].plot()
plt.title("Closed price in september 2019")
plt.subplot(2,2,4)
bitcoin['Close'].plot()
plt.title("Closed price from 2012")
plt.show()
```

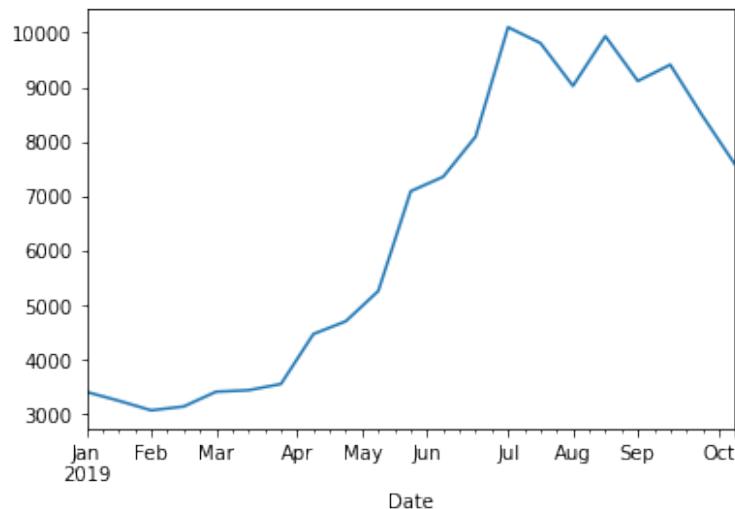


## Resample

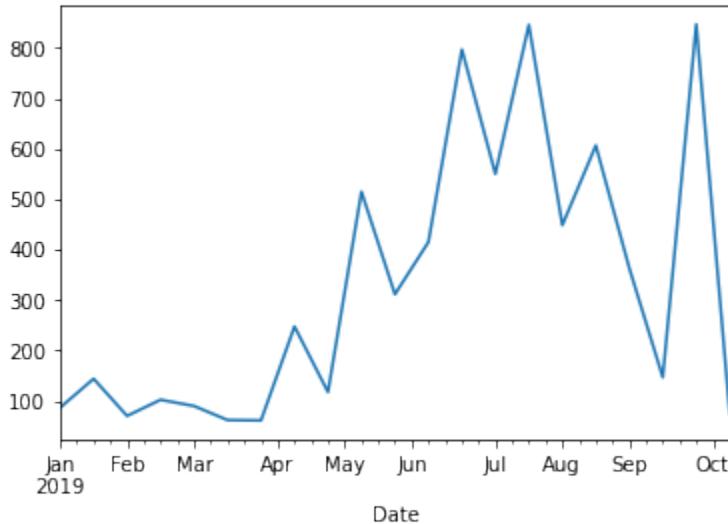
```
[16]: plt.figure(figsize=(12, 8))
plt.subplot(2,2,1)
bitcoin['2017':'2019']['Close'].resample('M').plot()
plt.title("Closed price from 2017-2019 resample per month")
plt.subplot(2,2,2)
bitcoin['2019']['Close'].resample('3W').plot()
plt.title("Closed price in 2019 resample per 3 week")
plt.subplot(2,2,3)
bitcoin['2019-09']['Close'].resample('W').plot()
plt.title("Closed price in september 2019 resample per day")
plt.subplot(2,2,4)
bitcoin['Close'].resample('Y').plot()
plt.title("Closed price from 2012 resample per year")
plt.show()
```



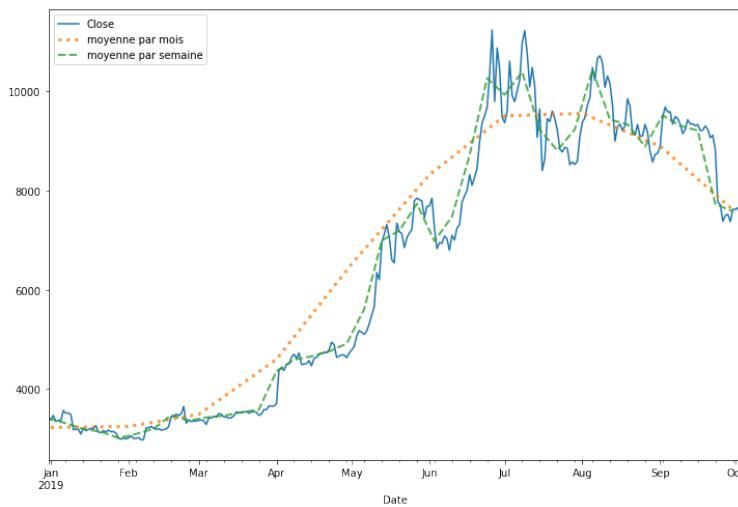
```
[8]: bitcoin.loc['2019', 'Close'].resample('2W').mean().plot()
plt.show()
```



```
[9]: bitcoin.loc['2019', 'Close'].resample('2W').std().plot()
plt.show()
```



```
[10]: plt.figure(figsize=(12, 8))
bitcoin.loc['2019', 'Close'].plot()
bitcoin.loc['2019', 'Close'].resample('M').mean().plot(label='moyenne par mois', lw=3, ls=':', alpha=0.8)
bitcoin.loc['2019', 'Close'].resample('W').mean().plot(label='moyenne par semaine', lw=2, ls='--', alpha=0.8)
plt.legend()
plt.show()
```



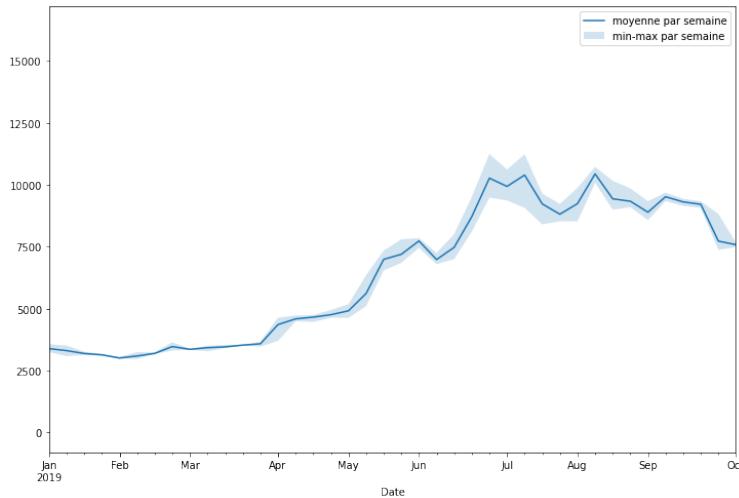
## Aggregate

```
[11]: m = bitcoin['Close'].resample('W').agg(['mean', 'std', 'min', 'max'])

plt.figure(figsize=(12, 8))
m['mean']['2019'].plot(label='moyenne par semaine')
```

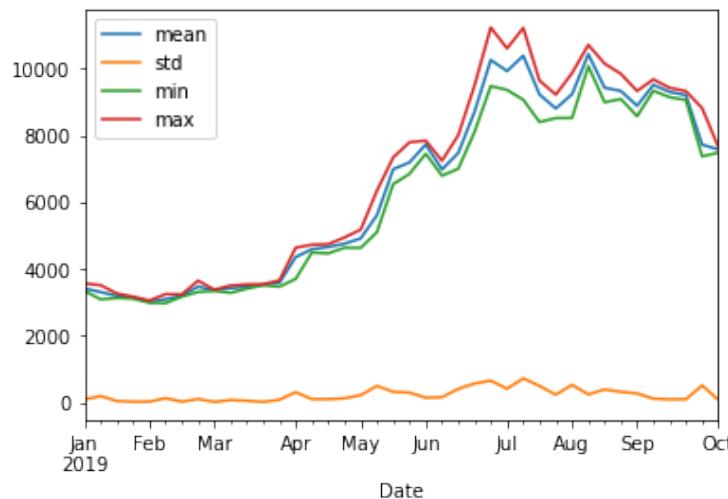
```
plt.fill_between(m.index, m['max'], m['min'], alpha=0.2, label='min-max par semaine')

plt.legend()
plt.show()
```



```
[12]: bitcoin.loc['2019', 'Close'].resample('W').agg(['mean', 'std', 'min', 'max']).plot()
```

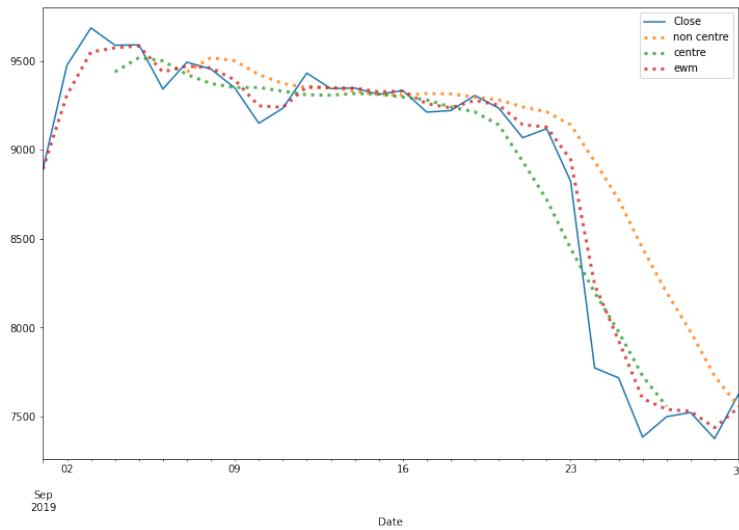
```
[12]: <matplotlib.axes._subplots.AxesSubplot at 0x7f6919691470>
```



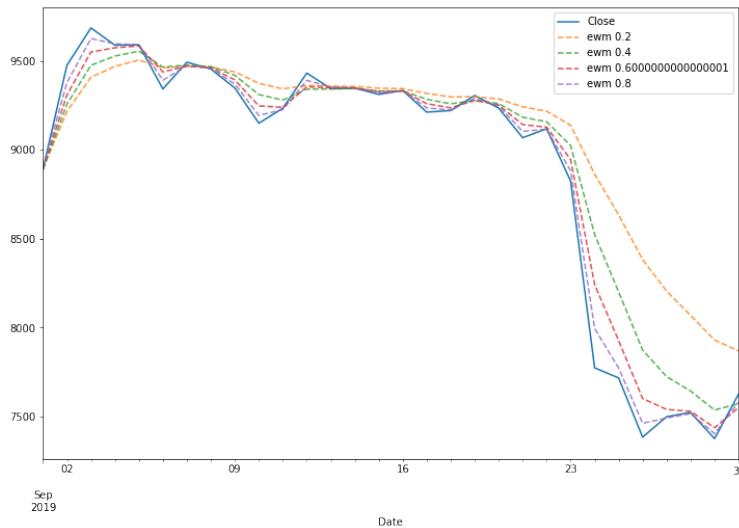
### 3.1.7 Moving Average et EWM

```
[21]: plt.figure(figsize=(12, 8))
bitcoin.loc['2019-09', 'Close'].plot()
bitcoin.loc['2019-09', 'Close'].rolling(window=7).mean().plot(label='non centre', lw=3, ls='--', alpha=0.8)
```

```
bitcoin.loc['2019-09', 'Close'].rolling(window=7, center=True).mean() .
    plot(label='centre', lw=3, ls=':', alpha=0.8)
bitcoin.loc['2019-09', 'Close'].ewm(alpha=0.6).mean().plot(label='ewm', lw=3, ls=':', alpha=0.8)
plt.legend()
plt.show()
```



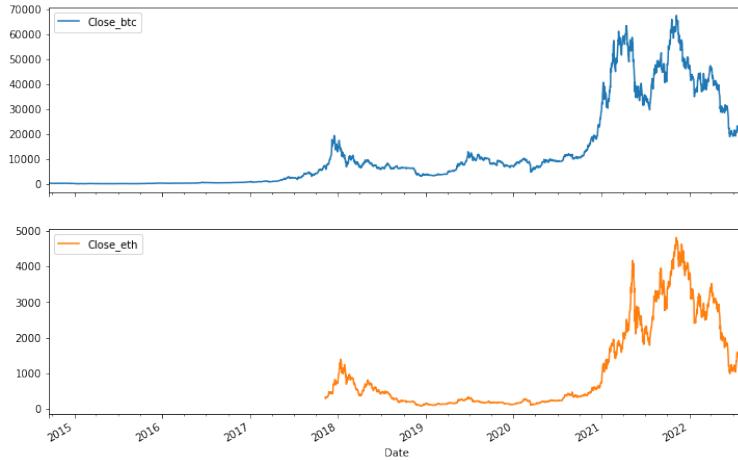
```
[22]: plt.figure(figsize=(12, 8))
bitcoin.loc['2019-09', 'Close'].plot()
for i in np.arange(0.2, 1, 0.2):
    bitcoin.loc['2019-09', 'Close'].ewm(alpha=i).mean().plot(label=f'ewm {i}', ls='--', alpha=0.8)
plt.legend()
plt.show()
```



### Comparaison de 2 série temporelles

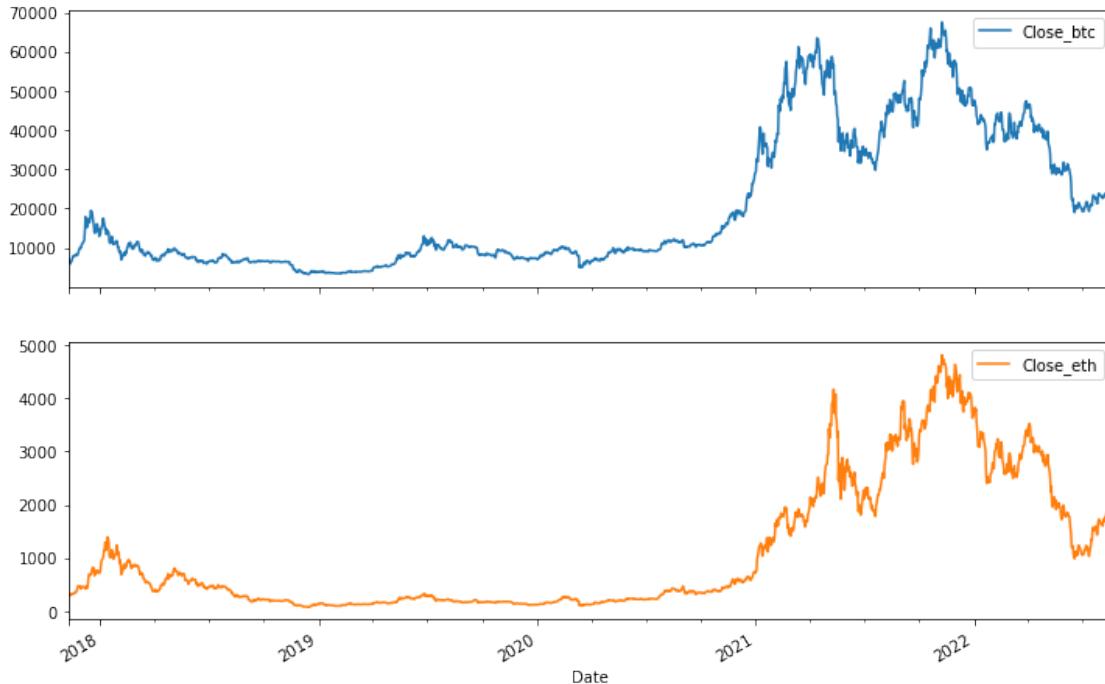
```
[27]: ethereum = pd.read_csv('Datasets/ETH-USD22.csv', index_col='Date',  
                           parse_dates=True)  
bitcoin2 = pd.read_csv('Datasets/BTC-USD22.csv', index_col='Date',  
                      parse_dates=True)
```

```
[33]: btc_eth = pd.merge(bitcoin2, ethereum, on='Date', how='outer', suffixes=('_btc',  
                           '_eth'))  
btc_eth[['Close_btc', 'Close_eth']].plot(subplots=True, figsize=(12, 8))
```



```
[34]: btc_eth = pd.merge(bitcoin2, ethereum, on='Date', how='inner', suffixes=('_btc',  
                           '_eth'))  
btc_eth[['Close_btc', 'Close_eth']].plot(subplots=True, figsize=(12, 8))
```

```
[34]: array([<AxesSubplot:xlabel='Date'>, <AxesSubplot:xlabel='Date'>],  
        dtype=object)
```



```
[39]: btc_eth[['Close_btc', 'Close_eth']].corr()
```

```
[39]:
```

	Close_btc	Close_eth
Close_btc	1.000000	0.926472
Close_eth	0.926472	1.000000

### 3.1.8 Exercice et Solution

Dans cet exercice, il s'agira de mettre en place la stratégie de la tortue qui est une technique de trading très ancienne afin de décider quand acheter ou vendre du bitcoin en fonction de la valeur de Close par rapport au minimum ou maximum des 28 derniers jours. Pour écrire ce code, vous devrez utiliser la fonction rowling :

- max sur les 28 derniers jours
- min au cours des 28 derniers jours

Lorsque la valeur de Close est supérieure au maximum des 28 derniers jours c'est signe qu'il faut acheter et à l'inverse lorsqu'elle est inférieure au minimum des 28 derniers jours c'est signe qu'il faut absolument vendre .

- Si 'Close' > max 28 derniers jours alors Buy =1
- Si 'Close' < min 28 derniers jours alors vendre = -1

Vous pouvez également utiliser la méthode de décalage shift( 1 ) qui vous permet de décaler vos fenêtres d'autant de jours que vous le souhaitez afin de prendre vos décisions une à l'avance.

```
[ ]:
```

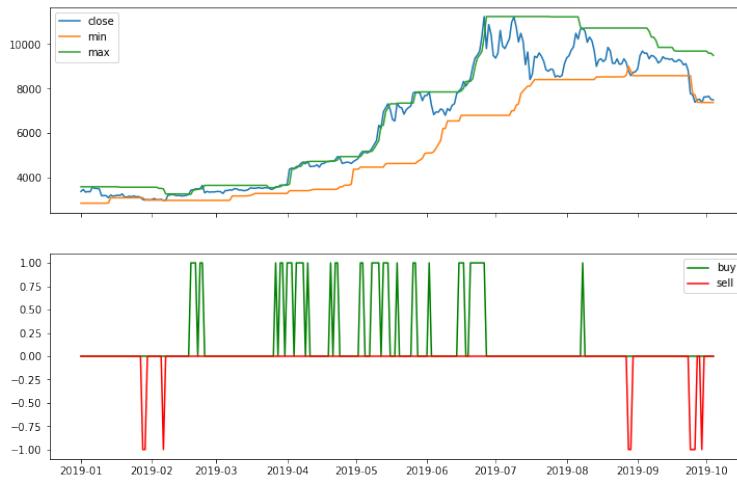
```
data = bitcoin.copy()
data['Buy'] = np.zeros(len(data))
```

```
data['Sell'] = np.zeros(len(data))
```

[0]: # votre code ici

[20]: # votre code ici

[20]: <matplotlib.legend.Legend at 0x7f6917b80710>



```
[ ]: data['RollingMax'] = data['Close'].shift(1).rolling(window=28).max()
data['RollingMin'] = data['Close'].shift(1).rolling(window=28).min()
data.loc[data['RollingMax'] < data['Close'], 'Buy'] = 1
data.loc[data['RollingMin'] > data['Close'], 'Sell'] = -1
```

```
[0]: start = '2019'
end='2019'
fig, ax = plt.subplots(2, figsize=(12, 8), sharex=True)
# plt.figure(figsize=(12, 8))
# plt.subplot(211)
ax[0].plot(data['Close'][start:end])
ax[0].plot(data['RollingMin'][start:end])
ax[0].plot(data['RollingMax'][start:end])
ax[0].legend(['close', 'min', 'max'])
ax[1].plot(data['Buy'][start:end], c='g')
ax[1].plot(data['Sell'][start:end], c='r')
ax[1].legend(['buy', 'sell'])
```

## 3.2 Seaborn

Seaborn est une bibliothèque Python de visualisation de données basée sur matplotlib. Elle fournit une interface de haut niveau pour dessiner des graphiques statistiques attrayants et informatifs.

Vous pouvez en savoir plus sur Seaborn : <https://seaborn.pydata.org/api.html>

```
[2]: import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

```
[4]: iris = sns.load_dataset('iris')
iris.head()
```

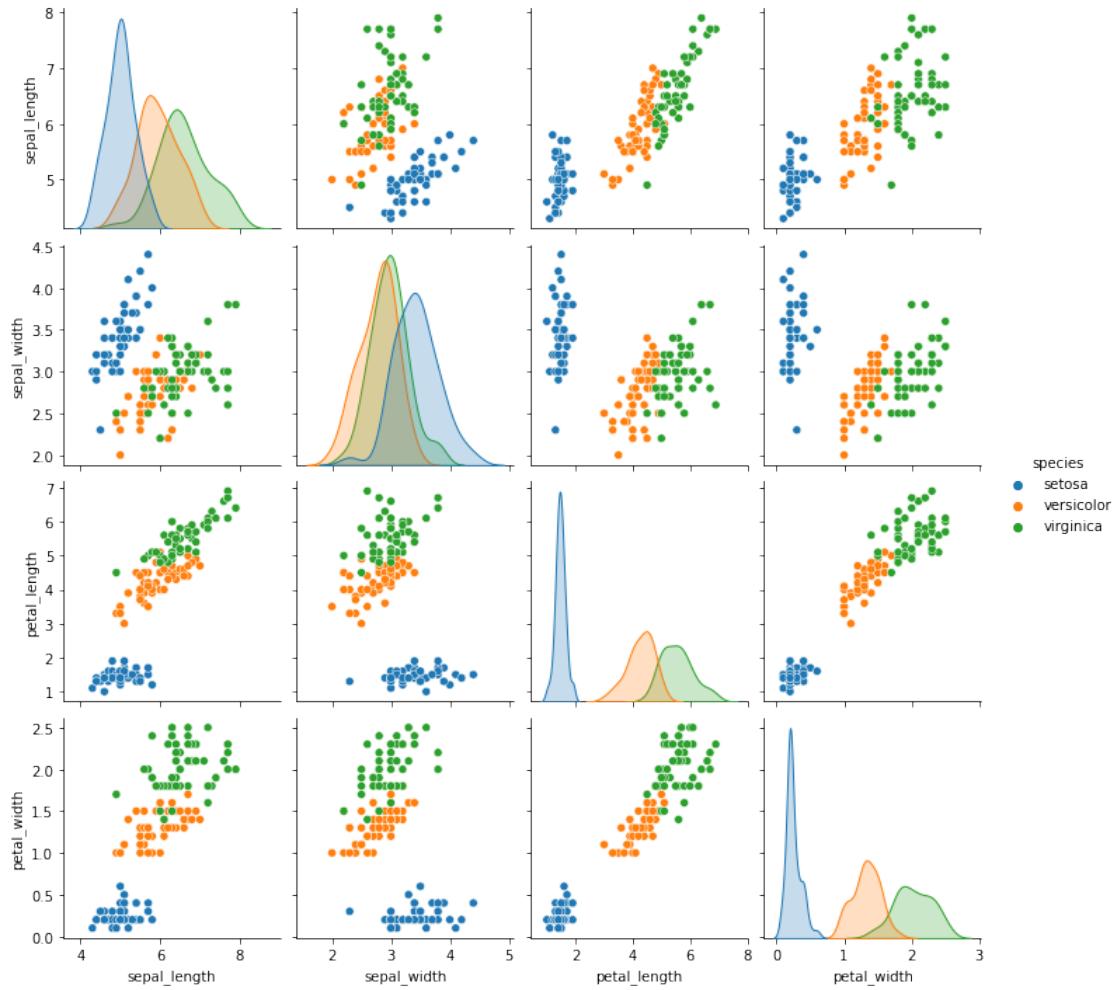
```
[4]:   sepal_length  sepal_width  petal_length  petal_width species
0            5.1         3.5          1.4         0.2  setosa
1            4.9         3.0          1.4         0.2  setosa
2            4.7         3.2          1.3         0.2  setosa
3            4.6         3.1          1.5         0.2  setosa
4            5.0         3.6          1.4         0.2  setosa
```

### 3.2.1 La vue d'ensemble Pairplot()

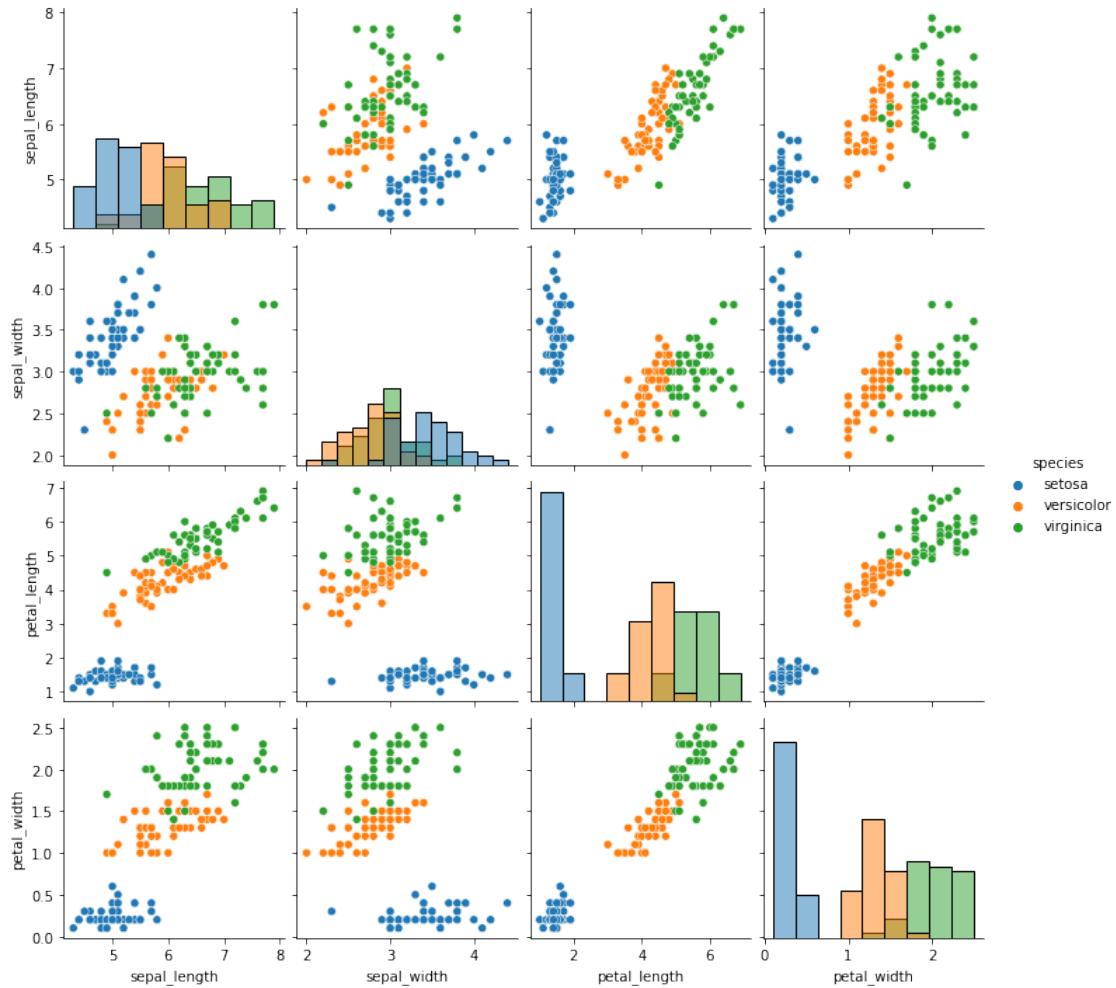
Tracer des relations par paires dans un jeu de données. Par défaut, cette fonction créera une grille d'axes telle que chaque variable numérique dans les données sera partagée sur les axes y sur une seule ligne et les axes x sur une seule colonne. Les diagrammes diagonaux sont traités différemment : un diagramme de distribution univariée est tracé pour montrer la distribution marginale des données dans chaque colonne.

Il est également possible d'afficher un sous-ensemble de variables ou de tracer différentes variables sur les lignes et les colonnes.

```
[5]: sns.pairplot(iris, hue='species')
```



```
[16]: sns.pairplot(iris, hue='species', diag_kind="hist")
```



### 3.2.2 Visualiser de catégories

Interface au niveau de la figure pour dessiner des tracés catégoriels sur un FacetGrid.

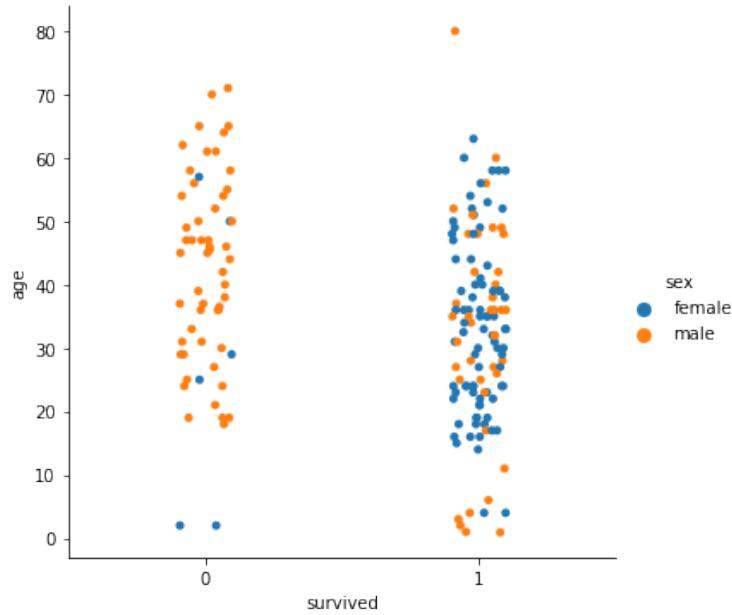
Cette fonction donne accès à plusieurs fonctions au niveau des axes qui montrent la relation entre une variable numérique et une ou plusieurs variables catégorielles à l'aide d'une des nombreuses représentations visuelles.

```
[6]: titanic = sns.load_dataset('titanic')
titanic.drop(['alone', 'alive', 'who', 'adult_male', 'embark_town', 'class'], axis=1, inplace=True)
titanic.dropna(axis=0, inplace=True)
titanic.head()
```

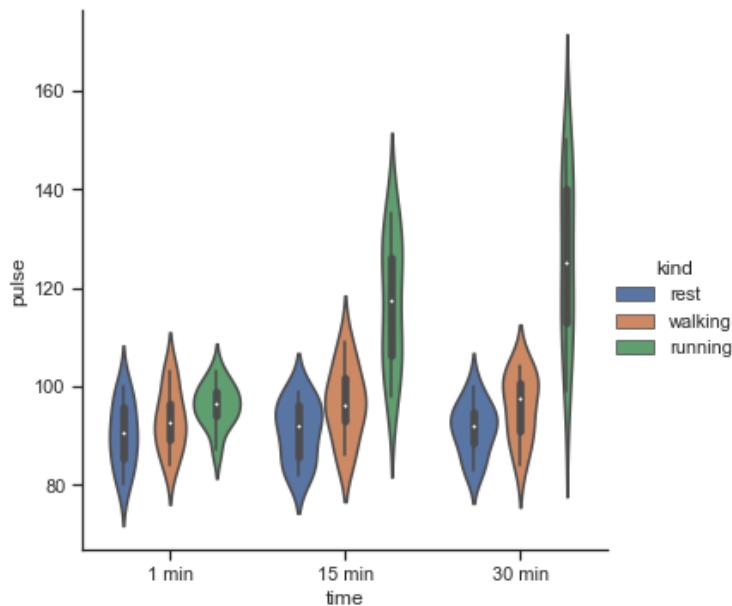
	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
1	1	1	female	38.0	1	0	71.2833	C	C
3	1	1	female	35.0	1	0	53.1000	S	C
6	0	1	male	54.0	0	0	51.8625	S	E
10	1	3	female	4.0	1	1	16.7000	S	G

```
11          1          1  female  58.0          0          0  26.5500      S      C
```

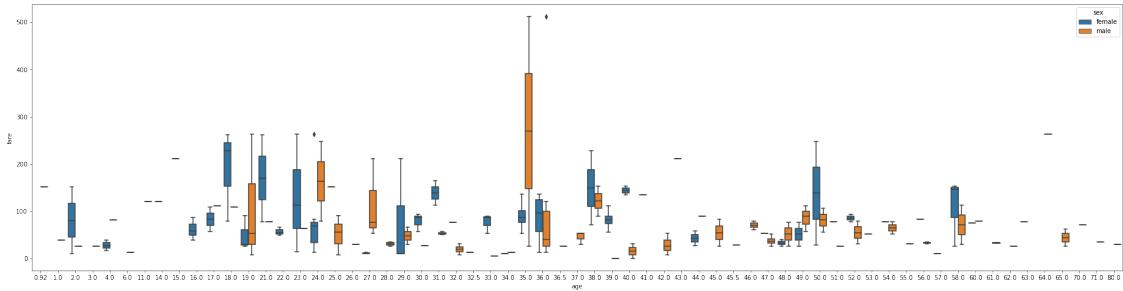
```
[7]: sns.catplot(x='survived', y='age', data=titanic, hue='sex')
```



```
[22]: sns.set_theme(style="ticks")
exercise = sns.load_dataset("exercise")
g = sns.catplot(x="time", y="pulse", hue="kind", data=exercise, kind="violin")
```

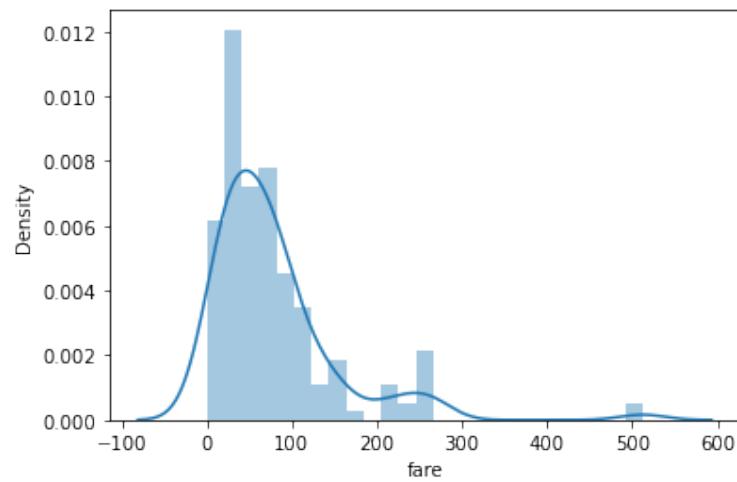


```
[8]: plt.figure(figsize=(32, 8))
sns.boxplot(x='age', y='fare', data=titanic, hue='sex')
```

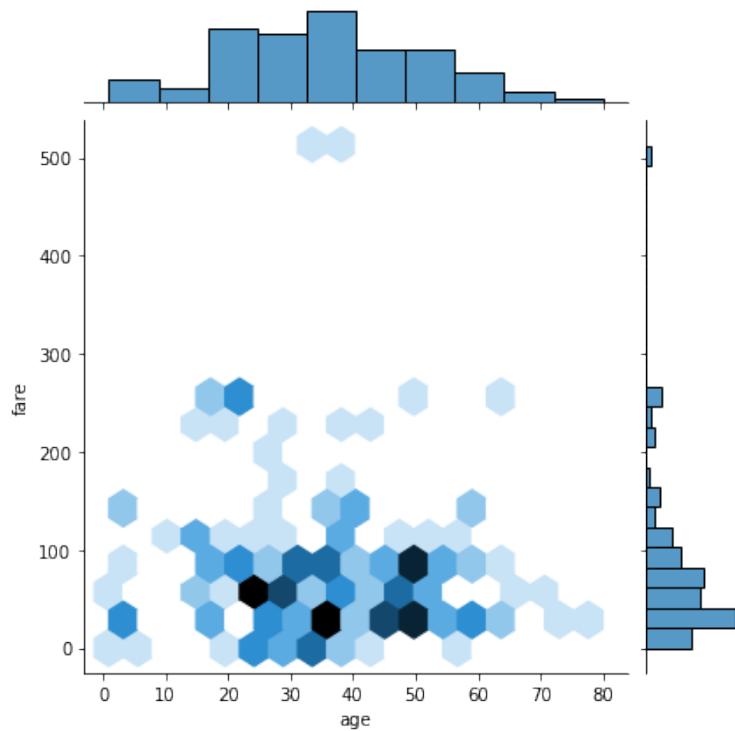


### 3.2.3 Visualisation de Distributions

```
[13]: sns.distplot(titanic['fare'])
```



```
[14]: sns.jointplot('age', 'fare', data=titanic, kind='hex')
```



```
[15]: sns.heatmap(titanic.corr())
```



# Chapter 4

## Machine learning avec Sklearn

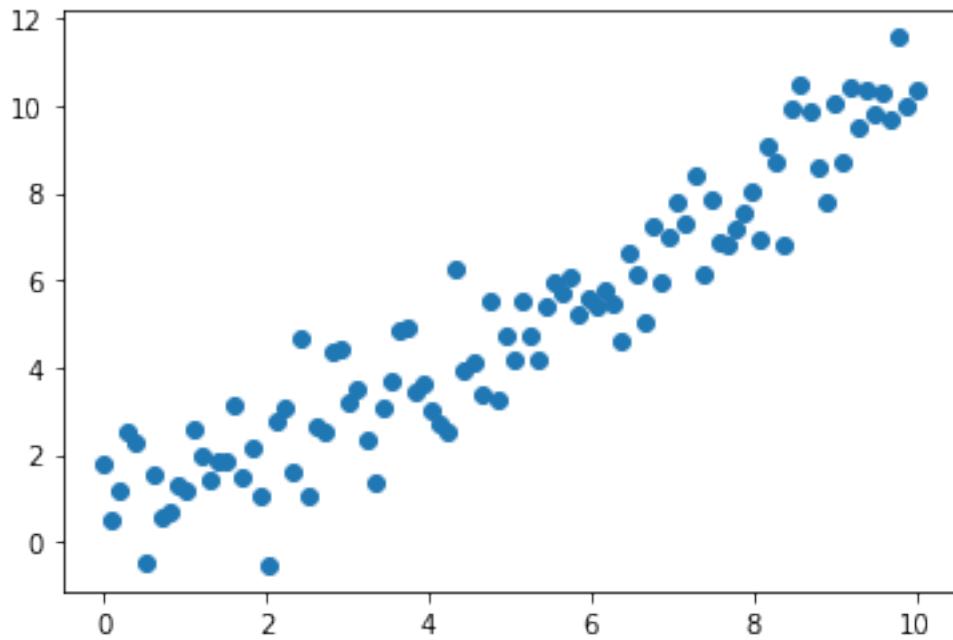
### 4.1 Introduction sur Sklearn

```
[1]: import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns
```

#### 4.1.1 Régression

```
[2]: np.random.seed(0)  
m = 100 # creation de 100 échantillons  
X = np.linspace(0, 10, m).reshape(m,1)  
y = X + np.random.randn(m, 1)  
  
plt.scatter(X, y)
```

```
[2]: <matplotlib.collections.PathCollection at 0x7fb847567cd0>
```



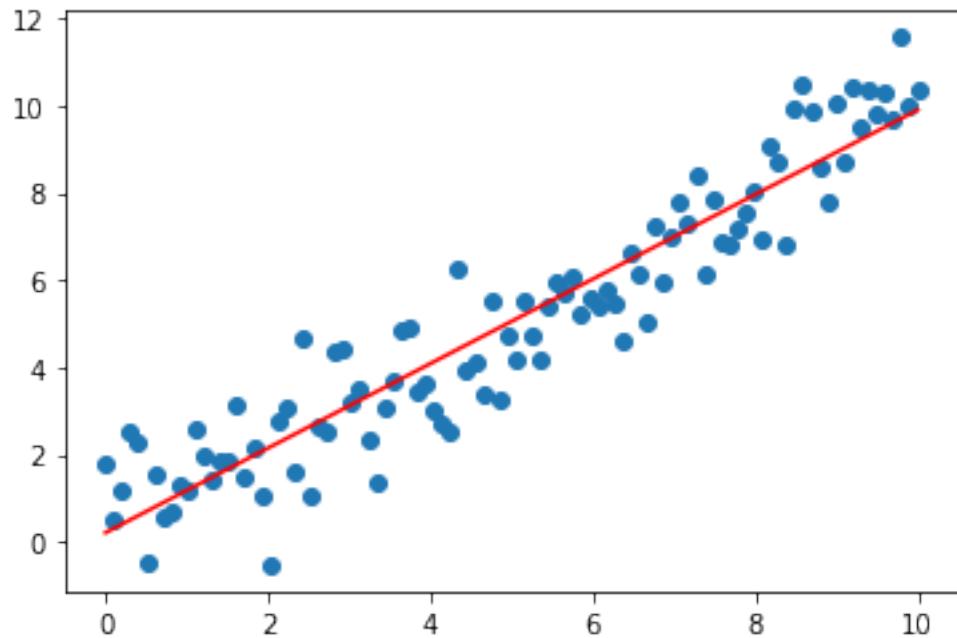
```
[3]: from sklearn.linear_model import LinearRegression
```

```
[4]: model = LinearRegression()
model.fit(X, y) # entrainement du modèle
model.score(X, y) # évaluation avec le coefficient de corrélation
```

```
[4]: 0.8881140743377214
```

```
[5]: plt.scatter(X, y)
plt.plot(X, model.predict(X), c='red')
```

```
[5]: [<matplotlib.lines.Line2D at 0x7fb84ad51700>]
```

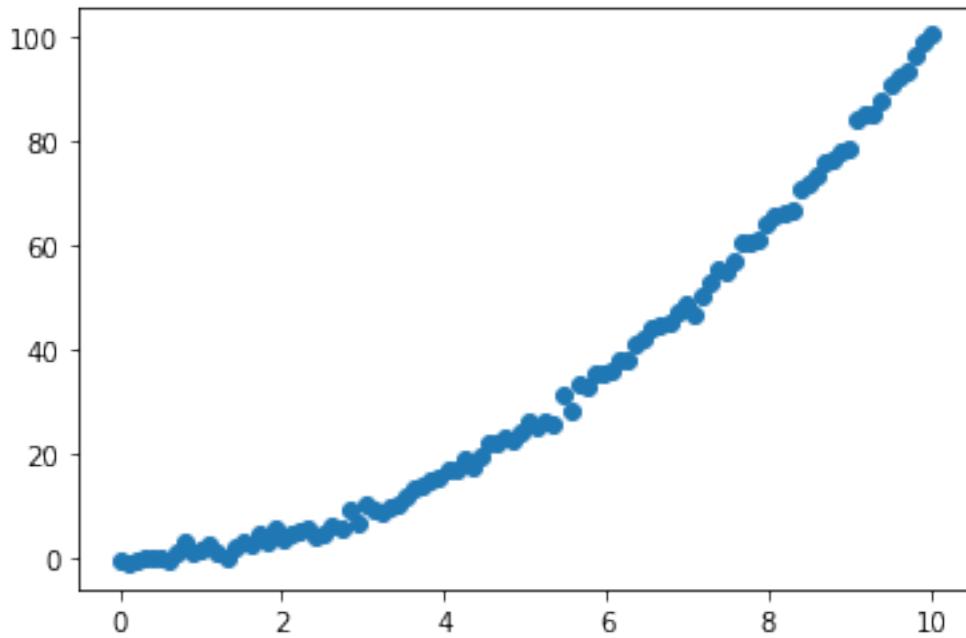


```
[9]: X = np.linspace(0, 10, m).reshape(m,1)
y = X**2 + np.random.randn(m, 1)

plt.scatter(X, y)
```

(100, 1)  
(100, 1)

[9]: <matplotlib.collections.PathCollection at 0x7fb84c2853d0>

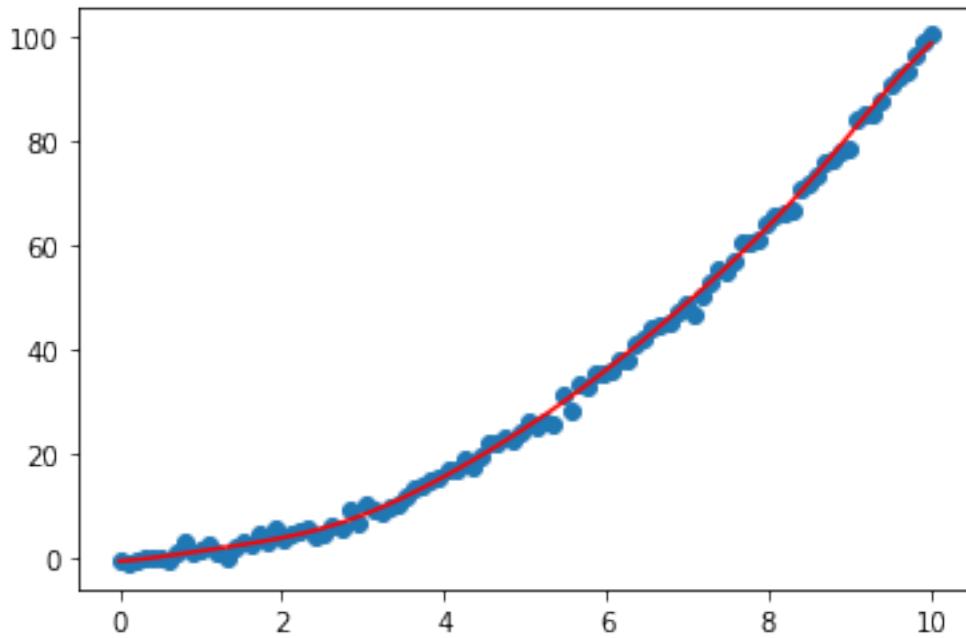


```
[11]: from sklearn.svm import SVR  
y = y.ravel()  
model = SVR(C =100)  
model.fit(X, y) # entrainement du modèle  
model.score(X, y) # évaluation avec le coefficient de corrélation
```

[11]: 0.9986816549763045

```
[12]: plt.scatter(X, y)  
plt.plot(X, model.predict(X), c='red')
```

[12]: [<matplotlib.lines.Line2D at 0x7fb84adb25b0>]



#### 4.1.2 Classification

```
[8]: titanic = sns.load_dataset('titanic')
titanic = titanic[['survived', 'pclass', 'sex', 'age']]
titanic.dropna(axis=0, inplace=True)
titanic['sex'].replace(['male', 'female'], [0, 1], inplace=True)
titanic.head()
```

```
[8]:    survived  pclass  sex   age
0          0       3   0  22.0
1          1       1   1  38.0
2          1       3   1  26.0
3          1       1   1  35.0
4          0       3   0  35.0
```

```
[ ]: from sklearn.neighbors import KNeighborsClassifier
```

```
[ ]: model = KNeighborsClassifier()
```

```
[ ]: y = titanic['survived']
X = titanic.drop('survived', axis=1)
```

```
[14]: model.fit(X, y) # entrainement du modèle
model.score(X, y) # évaluation
```

[14]: 0.8319327731092437

Prediction de survie

```
[ ]: def survie(model, pclass=3, sex=0, age=26):
    x = np.array([pclass, sex, age]).reshape(1, 3)
    print(model.predict(x))
    print(model.predict_proba(x))
```

[18]: survie(model)

```
[0]
[[0.6 0.4]]
```

### 4.1.3 Exercice et Solution

Écrire un code qui permet de trouver la meilleure valeur de voisin n\_neighbors pour le modèle de KNeighborsClassifier.

Dans sklearn, il est possible de faire cela avec la classe GridSearchCV. Mais il peut également être utile de savoir écrire soi-même ce genre de code de recherche.

```
[19]: # SOLUTION
score = []
best_k = 1
best_score = 0

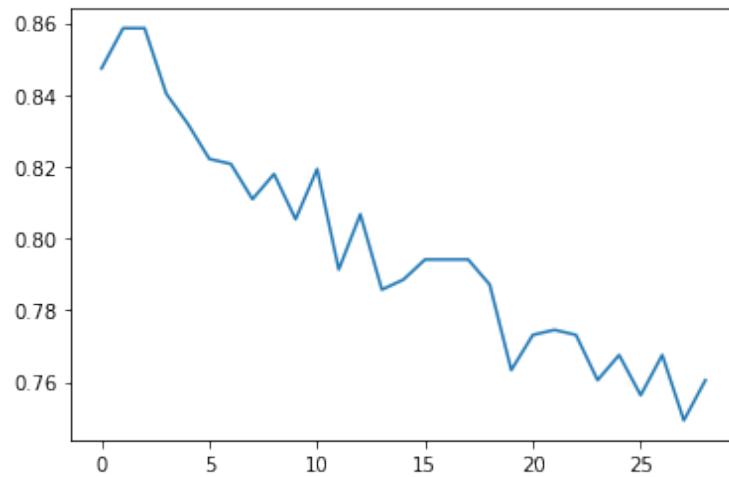
for k in range(best_k, 30):
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(X, y)
    score.append(model.score(X, y))

    if best_score < model.score(X, y):
        best_k = k
        best_score = model.score(X, y)

print(best_k)
plt.plot(score)
```

2

[19]: [<matplotlib.lines.Line2D at 0x7fd1eef752b0>]



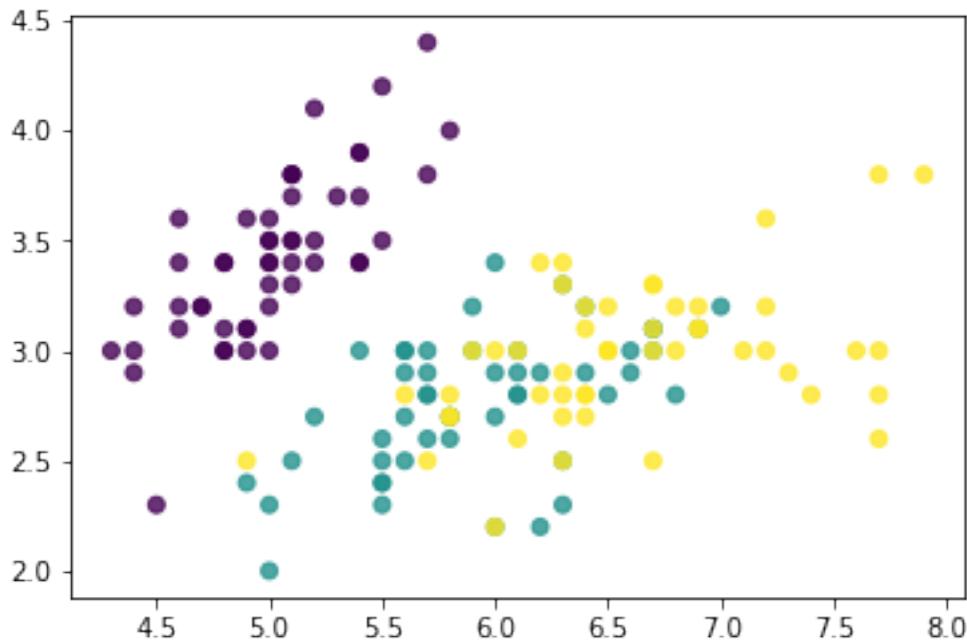
## 4.2 Sélection de Modele

```
[14]: import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
import seaborn as sns
```

```
[15]: iris = load_iris()
X = iris.data
y = iris.target

plt.scatter(X[:, 0], X[:, 1], c=y, alpha=0.8)
```

```
[15]: <matplotlib.collections.PathCollection at 0x7fbc1672beb0>
```



### 4.2.1 Train Test Split

```
[16]: from sklearn.model_selection import train_test_split
```

```
[17]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=5)

print('Train set:', X_train.shape)
print('Test set:', X_test.shape)
```

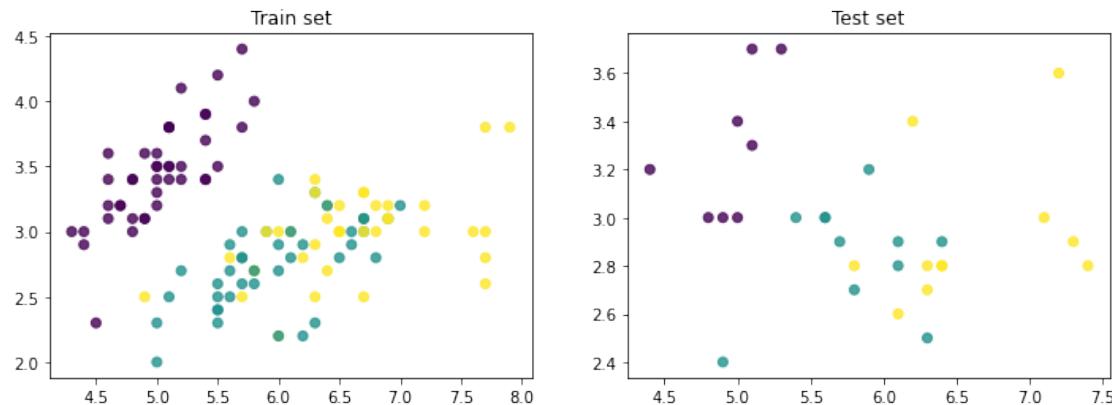
Train set: (120, 4)

Test set: (30, 4)

[ ]:

```
[5]: plt.figure(figsize=(12, 4))
plt.subplot(121)
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, alpha=0.8)
plt.title('Train set')
plt.subplot(122)
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, alpha=0.8)
plt.title('Test set')
```

[5]: Text(0.5, 1.0, 'Test set')



```
[12]: from sklearn.neighbors import KNeighborsClassifier
```

```
[8]: model = KNeighborsClassifier(n_neighbors=6)

model.fit(X_train, y_train)

print('train score:', model.score(X_train, y_train))
print('test score:', model.score(X_test, y_test))
```

train score: 0.9833333333333333  
 test score: 0.9666666666666667

## 4.2.2 Validation Set et Cross Validation

```
[21]: from sklearn.model_selection import cross_val_score
```

```
[7]: model = KNeighborsClassifier()
cross_val_score(model, X_train, y_train, cv=5, scoring='accuracy')
```

```
[7]: array([1.          , 1.          , 1.          , 0.95833333, 0.95833333])
```

## Kfold

```
[11]: from sklearn.model_selection import KFold
model = KNeighborsClassifier()
cv= KFold()
cross_val_score(model, X_train, y_train, cv=cv, scoring='accuracy')
```

```
[11]: array([1.          , 0.95833333, 1.          , 0.95833333, 0.95833333])
```

## ShuffleSplit

```
[13]: from sklearn.model_selection import ShuffleSplit  
model = KNeighborsClassifier()  
cv= ShuffleSplit(n_splits=5, test_size=.25, random_state=0)  
cross_val_score(model, X_train, y_train, cv=cv, scoring='accuracy')
```

```
[13]: array([1.          , 0.93333333, 0.86666667, 1.          , 1.          ])
```

## LeaveOneOut

```
[14]: from sklearn.model_selection import LeaveOneOut
      model = KNeighborsClassifier()
      cv= LeaveOneOut()
      cross_val_score(model, X_train, y_train, cv=cv, scoring='accuracy')
```

## StratifiedKFold

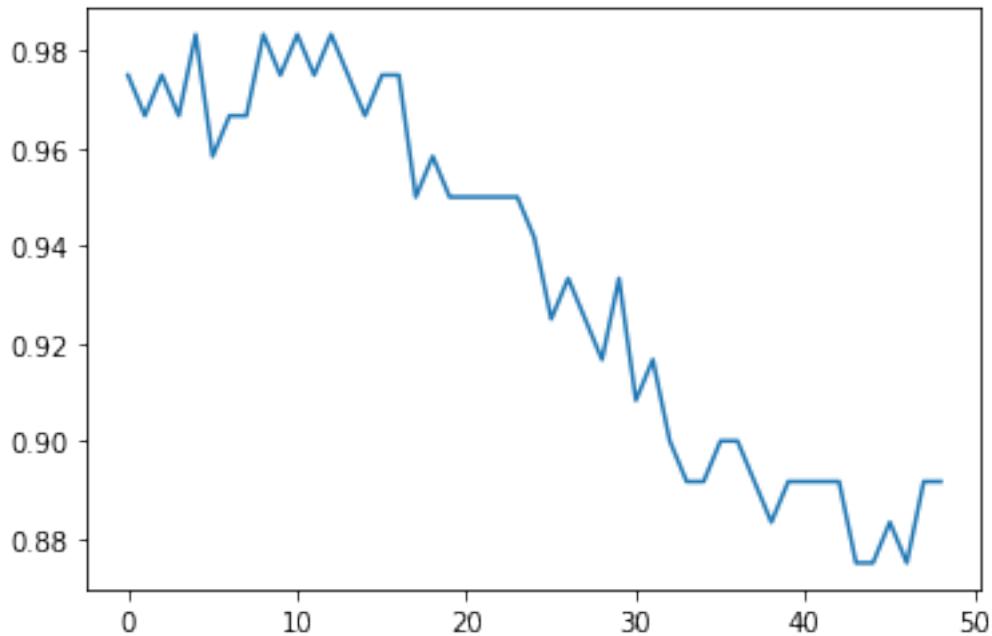
```
[17]: from sklearn.model_selection import StratifiedKFold  
model = KNeighborsClassifier()  
cv= StratifiedKFold(n_splits=5, shuffle=True)  
cross_val_score(model, X_train, y_train, cv=cv, scoring='accuracy')
```

```
[17]: array([0.95833333, 0.95833333, 0.95833333, 1. , 0.95833333])
```

```
[20]: 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)

plt.plot(val_score)
```

[20]: [`<matplotlib.lines.Line2D at 0x7f7c5cfcccd68>`]



### 4.2.3 Validation Curve

```
[11]: from sklearn.model_selection import validation_curve

[12]: model = KNeighborsClassifier()
k = np.arange(1, 50)

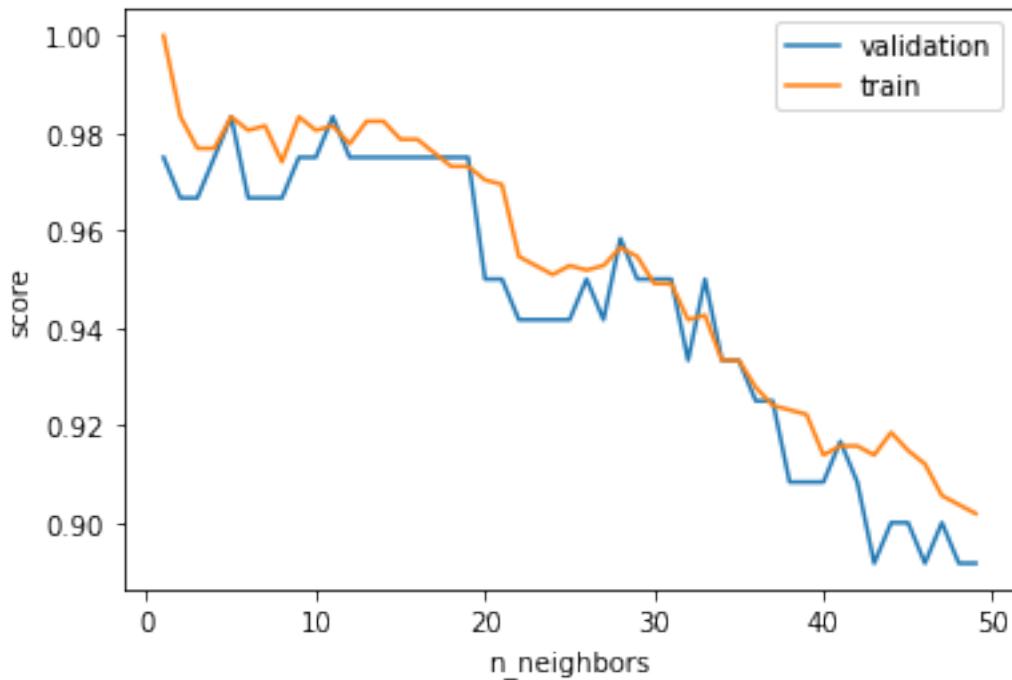
train_score, val_score = validation_curve(model, X_train, y_train,
                                         'n_neighbors', k, cv=10)

plt.plot(k, val_score.mean(axis=1), label='validation')
plt.plot(k, train_score.mean(axis=1), label='train')

plt.ylabel('score')
plt.xlabel('n_neighbors')
plt.legend()

/Users/mac/opt/anaconda3/lib/python3.8/site-
packages/sklearn/utils/validation.py:67: FutureWarning: Pass
param_name=n_neighbors, param_range=[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14
15 16 17 18 19 20 21 22 23 24
25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
49] as keyword args. From version 0.25 passing these as positional arguments
will result in an error
    warnings.warn("Pass {} as keyword args. From version 0.25 "
```

[12]: <matplotlib.legend.Legend at 0x7fc020dfc550>



#### 4.2.4 GridSearchCV

[10]: `from sklearn.model_selection import GridSearchCV`

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

grid = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5)

grid.fit(X_train, y_train)
```

```
[18]: GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
                  param_grid={'metric': ['euclidean', 'manhattan'],
                  'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8,
9, 10, 11, 12, 13, 14, 15, 16, 17,
18, 19])})
```

```
[19]: print(grid.best_score_)
print(grid.best_params_)
```

```
0.9833333333333334
{'metric': 'euclidean', 'n_neighbors': 5}
```

```
[16]: model = grid.best_estimator_
model.score(X_test, y_test)
```

```
[16]: 0.9333333333333333
```

#### 4.2.5 Learning Curve

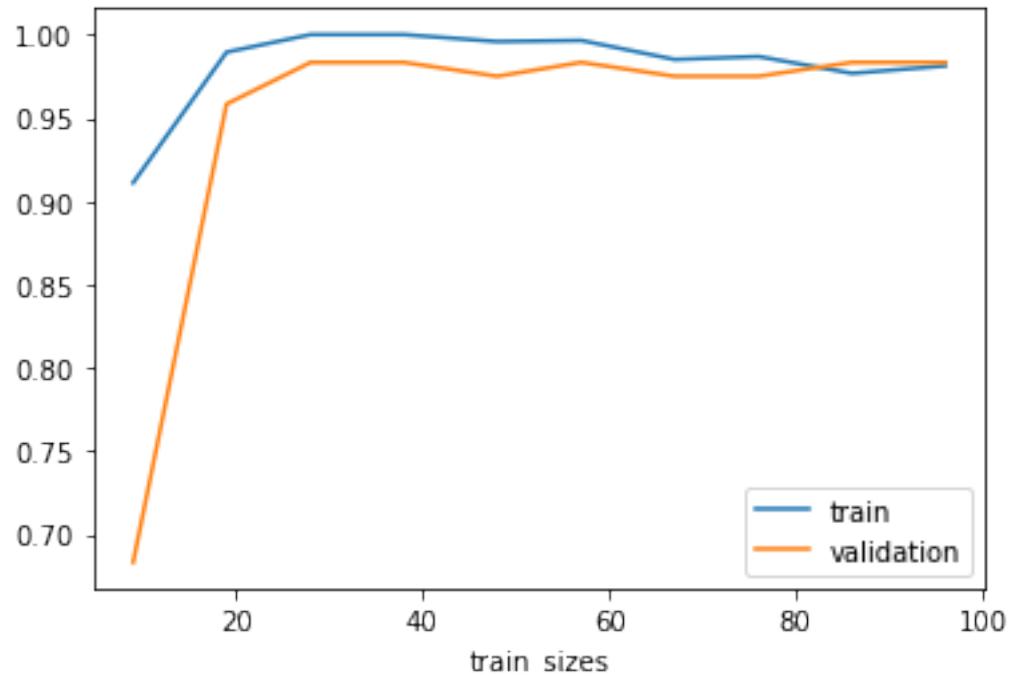
```
[19]: from sklearn.model_selection import learning_curve
```

```
[20]: N, train_score, val_score = learning_curve(model, X_train, y_train,
                                                train_sizes=np.linspace(0.1, 1, 10), cv=5)

print(N)
plt.plot(N, train_score.mean(axis=1), label='train')
plt.plot(N, val_score.mean(axis=1), label='validation')
plt.xlabel('train_sizes')
plt.legend()
```

```
[ 9 19 28 38 48 57 67 76 86 96]
```

```
[20]: <matplotlib.legend.Legend at 0x7fc020e71550>
```



#### 4.2.6 Metrics d'évaluation

##### Erreur moyen MAE et MSE

```
[8]: from sklearn.metrics import *
from sklearn.linear_model import LinearRegression
np.random.seed(0)
m = 100 # creation de 100 échantillons
X = np.linspace(0, 10, m).reshape(m,1)
y = X + np.random.randn(m, 1)
model = LinearRegression()
model.fit(X, y) # entrainement du modèle
model.score(X, y) # évaluation avec le coefficient de corrélation
```

[8]: 0.8881140743377214

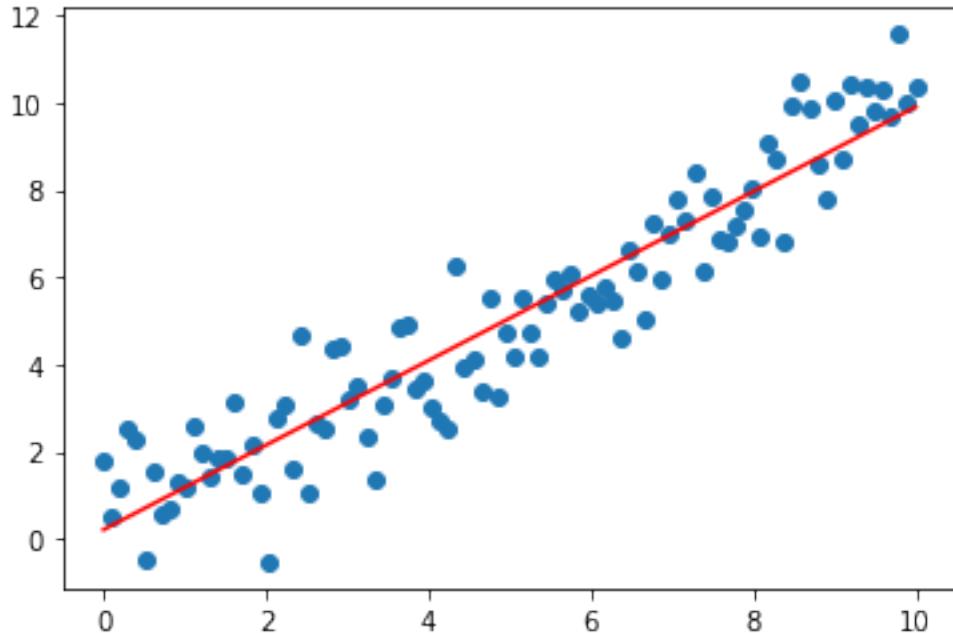
```
[6]: plt.scatter(X, y)
y_pred = model.predict(X)
plt.plot(X, y_pred, c='red')

print("MAE: ", mean_absolute_error(y,y_pred))
print("MSE: ", np.sqrt(mean_squared_error(y,y_pred)))
print("MdAE", median_absolute_error(y,y_pred))
```

MAE: 0.8051083770818453

MSE: 1.0041468286404196

MdAE 0.6846679204665103



### Confusion Matrix

```
[ ]: from sklearn.metrics import confusion_matrix  
confusion_matrix(y_test, model.predict(X_test))
```

### Metrics d'évaluation des modèles

```
[9]: import sklearn.metrics  
sorted(sklearn.metrics.SCORERS.keys())
```

```
[9]: ['accuracy',  
'adjusted_mutual_info_score',  
'adjusted_rand_score',  
'average_precision',  
'balanced_accuracy',  
'completeness_score',  
'explained_variance',  
'f1',  
'f1_macro',  
'f1_micro',  
'f1_samples',  
'f1_weighted',  
'fowlkes_mallows_score',  
'homogeneity_score',  
'jaccard',  
'jaccard_macro',  
'jaccard_micro',  
'jaccard_samples',  
'jaccard_weighted',  
'max_error',  
'mutual_info_score',  
'neg_brier_score',  
'neg_log_loss',  
'neg_mean_absolute_error',  
'neg_mean_gamma_deviance',  
'neg_mean_poisson_deviance',  
'neg_mean_squared_error',  
'neg_mean_squared_log_error',  
'neg_median_absolute_error',  
'neg_root_mean_squared_error',  
'normalized_mutual_info_score',  
'precision',  
'precision_macro',  
'precision_micro',  
'precision_samples',
```

```
'precision_weighted',
'r2',
'recall',
'recall_macro',
'recall_micro',
'recall_samples',
'recall_weighted',
'roc_auc',
'roc_auc_ovr',
'roc_auc_ovr_weighted',
'roc_auc_ovr_weighted',
've_measure_score']
```

```
[42]: model = KNeighborsClassifier()
print(cross_val_score(model, X_train, y_train, cv=5, scoring='accuracy'))
print(cross_val_score(model, X_train, y_train, cv=5, scoring='f1_micro'))
print(cross_val_score(model, X_train, y_train, cv=5, scoring='precision_micro'))
```

```
[1.      1.      1.      0.95833333 0.95833333]
[1.      1.      1.      0.95833333 0.95833333]
[1.      1.      1.      0.95833333 0.95833333]
```

#### 4.2.7 Exercice et solution

Pour cet exercice, appliquez ce que nous avons appris sur la sélection de modèles sur le jeu de données Titanic : - Créer un Train-Set et un Test-set. s'entraîner puis évaluer - Avec GridSearch, trouvez les meilleurs hyper-paramètres, métriques et pondérations n\_neighbors - La collecte de plus de données sera-t-elle utile ?

```
[22]: titanic = sns.load_dataset('titanic')
titanic = titanic[['survived', 'pclass', 'sex', 'age']]
titanic.dropna(axis=0, inplace=True)
titanic['sex'].replace(['male', 'female'], [0, 1], inplace=True)
titanic.head()
y= titanic[ 'survived']
X= titanic.drop('survived', axis =1)

print(y.shape)
print(X.shape)
```

```
(714,)
(714, 3)
```

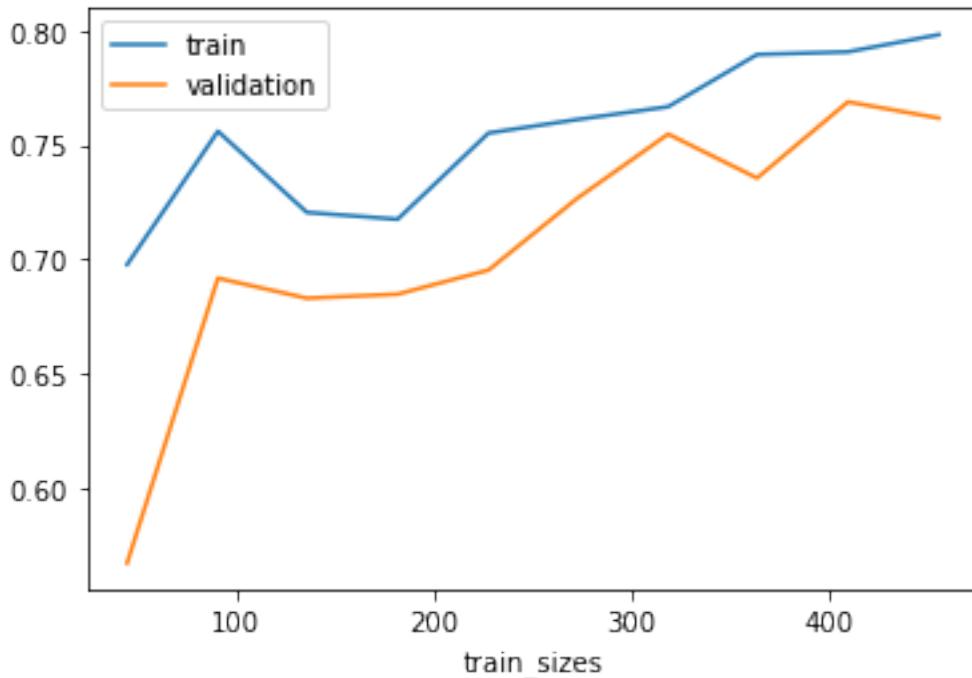
```
[23]: # data split
X_train,X_test,y_train,y_test = train_test_split(X ,y,test_size=0.2)
print(X_train.shape)
print(y_test.shape)
```

```
# train model
param_grid = {'n_neighbors': np.arange(1, 20),
              'metric': ['euclidean', 'manhattan']}
grid = GridSearchCV(KNeighborsClassifier(), param_grid, cv=10)
grid.fit(X_train, y_train)
# get the best estimator
print(grid.best_score_, grid.best_params_)
model = grid.best_estimator_
print(model.score(X_test, y_test))
# evalutate the mode
confusion_matrix(y_test, model.predict(X_test))

# get the learning curve
N, train_score, val_score = learning_curve(model, X_train, y_train,
                                             train_sizes=np.linspace(0.1, 1, 10), cv=5)
print(N)
plt.plot(N, train_score.mean(axis=1), label='train')
plt.plot(N, val_score.mean(axis=1), label='validation')
plt.xlabel('train_sizes')
plt.legend()
```

```
(571, 3)
(143,)
0.7705081669691469 {'metric': 'manhattan', 'n_neighbors': 13}
0.8251748251748252
[ 45  91 136 182 228 273 319 364 410 456]
```

[23]: <matplotlib.legend.Legend at 0x7fc02334b3a0>



## 4.3 Pre-processing

### 4.3.1 Encodage

Encodage LabelEncoder, MultiLabelBinarizer et LabelBinarizer

```
[19]: import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder,
    LabelBinarizer, MultiLabelBinarizer, OrdinalEncoder, OneHotEncoder
```

```
[83]: y = np.array(['chat', 'chien', 'chat', 'oiseau', 'lion', 'chien', 'chat'])

encoder = LabelEncoder()
print(encoder.fit_transform(y))
print(encoder.inverse_transform(np.array([0, 0, 2, 3, 3, 3, 0, 1, 2])))
```

[0 1 0 3 2 1 0]  
['chat' 'chat' 'lion' 'oiseau' 'oiseau' 'oiseau' 'chat' 'chien' 'lion']

```
[16]: encoder = LabelBinarizer()
encoder.fit_transform(y)
```

```
[16]: array([[1, 0, 0, 0],
           [0, 1, 0, 0],
           [1, 0, 0, 0],
           [0, 0, 0, 1],
           [0, 0, 1, 0],
           [0, 1, 0, 0],
           [1, 0, 0, 0]])
```

```
[20]: y = np.array([['chat', 'poils'],
                  ['lion', 'poils'],
                  ['chien', 'poils'],
                  ['vache', 'cuire'],
                  ['chat', 'poils'],
                  ['oiseau', 'plumes']])
encoder = MultiLabelBinarizer()
encoder.fit_transform(y)
```

```
[20]: array([[1, 0, 0, 0, 0, 0, 1, 0],
           [0, 0, 0, 1, 0, 0, 1, 0],
           [0, 1, 0, 0, 0, 0, 1, 0],
           [0, 0, 1, 0, 0, 0, 0, 1],
           [1, 0, 0, 0, 0, 0, 1, 0],
           [0, 0, 0, 0, 1, 1, 0, 0]])
```

## Encodage Ordinal et Encodage OneHot

```
[6]: X = np.array([['chat', 'poils'],
                 ['lion', 'poils'],
                 ['chien', 'poils'],
                 ['vache', 'cuire'],
                 ['chat', 'poils'],
                 ['oiseau', 'plumes']])
encoder = OrdinalEncoder()
encoder.fit_transform(X)
```

```
[6]: array([[0., 2.],
           [2., 2.],
           [1., 2.],
           [4., 0.],
           [0., 2.],
           [3., 1.]])
```

```
[8]: encoder = OneHotEncoder(sparse=True)
encoder.fit_transform(X)
```

```
[8]: <6x8 sparse matrix of type '<class 'numpy.float64'>'  
      with 12 stored elements in Compressed Sparse Row format>
```

### 4.3.2 Normalisation

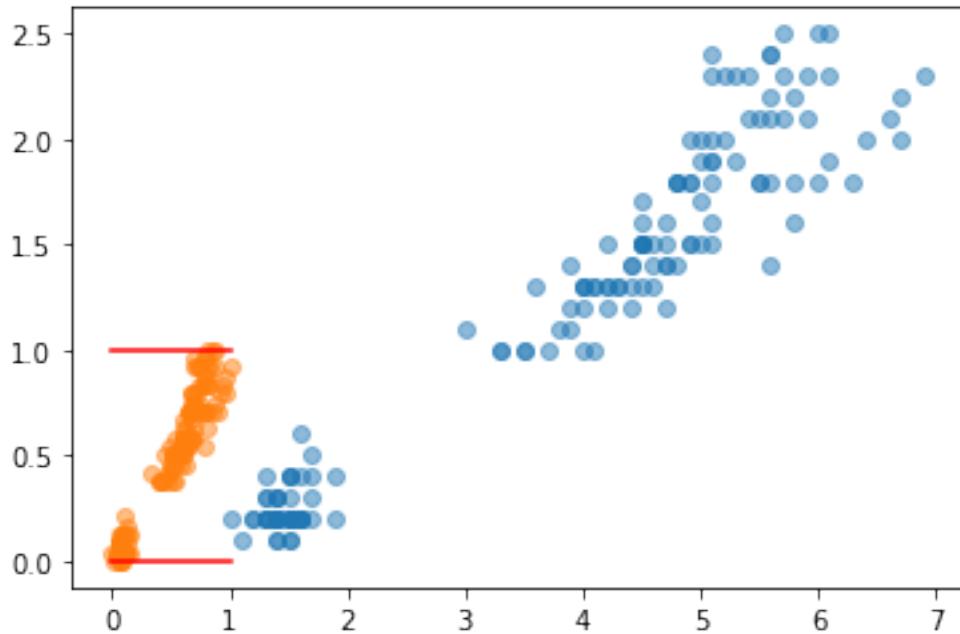
```
[22]: from sklearn.preprocessing import MinMaxScaler, StandardScaler, RobustScaler
```

```
[35]: from sklearn.datasets import load_iris  
iris = load_iris()  
X = iris.data
```

#### MinMaxScaler

```
[38]: X_minmax = MinMaxScaler().fit_transform(X)  
  
plt.scatter(X[:, 2], X[:, 3], alpha=0.5)  
plt.scatter(X_minmax[:, 2], X_minmax[:, 3], alpha=0.5)  
plt.plot(np.array([0,1]),np.array([1,1]),c='r')  
plt.plot(np.array([0,1]),np.array([0,0]),c='r')
```

```
[38]: [<matplotlib.lines.Line2D at 0x7f7ba491d040>]
```

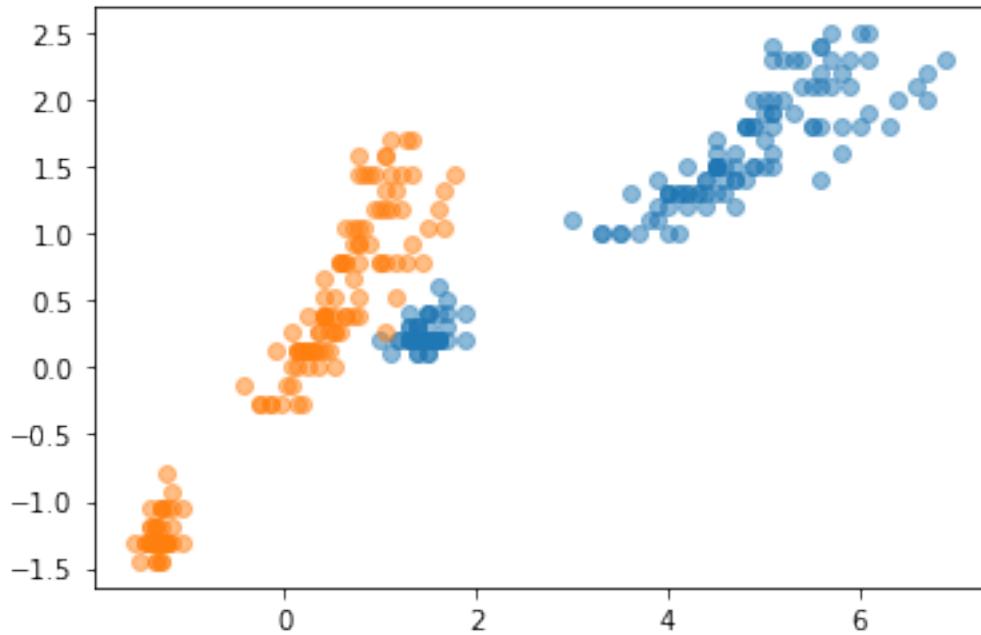


**StandardScaler**

```
[39]: X_stdscl = StandardScaler().fit_transform(X)

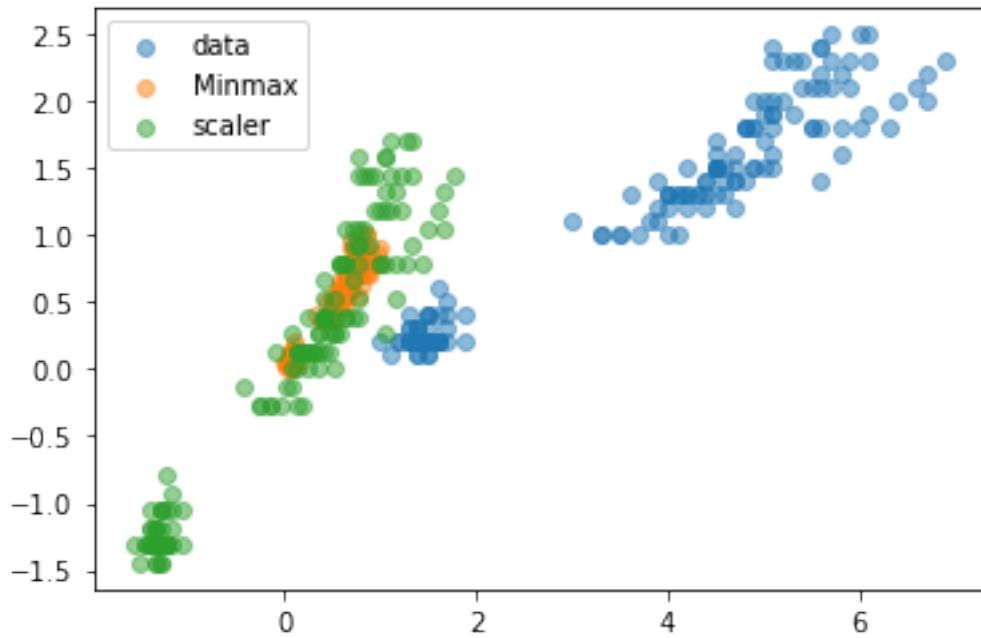
plt.scatter(X[:, 2], X[:, 3], alpha=0.5)
plt.scatter(X_stdscl[:, 2], X_stdscl[:, 3], alpha=0.5)
```

[39]: <matplotlib.collections.PathCollection at 0x7f7ba4a06190>



```
[31]: plt.scatter(X[:, 2], X[:, 3], alpha=0.5, label='data')
plt.scatter(X_minmax[:, 2], X_minmax[:, 3], alpha=0.5, label='Minmax')
plt.scatter(X_stdscl[:, 2], X_stdscl[:, 3], alpha=0.5, label='scaler')
plt.legend()
```

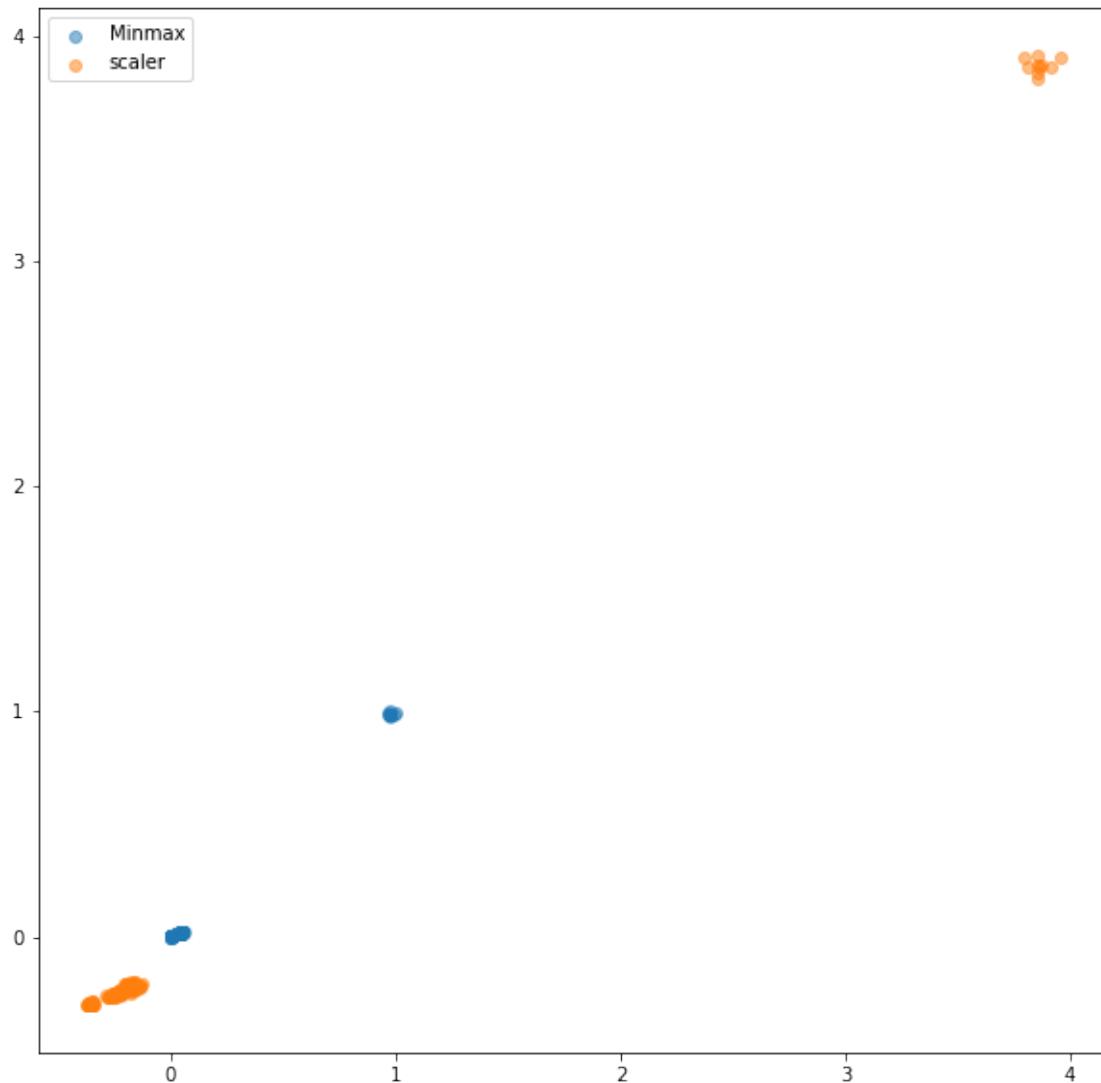
[31]: <matplotlib.legend.Legend at 0x7f7ba3d594f0>



```
[45]: outliers = np.full((10,4),100)+ np.random.randn(10,4)
x = np.vstack((X,outliers))
X_minmax = MinMaxScaler().fit_transform(x)
X_stdscl = StandardScaler().fit_transform(x)
plt.figure(figsize=(10,10))

plt.scatter(X_minmax[:, 2], X_minmax[:, 3],alpha=0.5,label='Minmax')
plt.scatter(X_stdscl[:, 2], X_stdscl[:, 3],alpha=0.5,label='scaler')
plt.legend()
```

```
[45]: <matplotlib.legend.Legend at 0x7f7ba3e955b0>
```

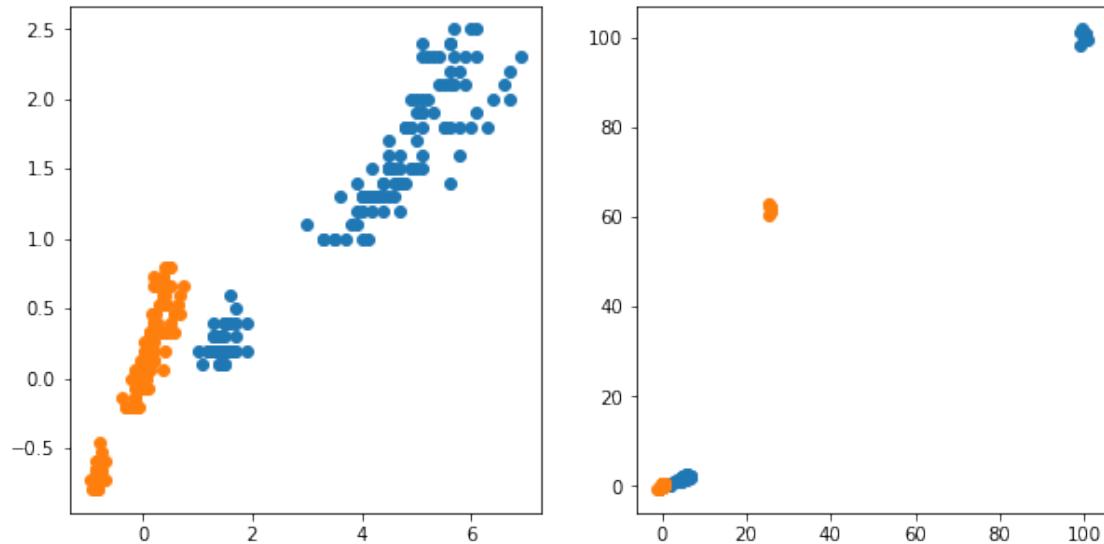


### RobustScaler

```
[44]: X_robust = RobustScaler().fit_transform(X)
plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
plt.scatter(X[:, 2], X[:, 3])
plt.scatter(X_robust[:, 2], X_robust[:, 3])

X_robust = RobustScaler().fit_transform(x)
plt.subplot(1,2,2)
plt.scatter(x[:, 2], x[:, 3])
plt.scatter(X_robust[:, 2], X_robust[:, 3])
```

[44]: <matplotlib.collections.PathCollection at 0x7f7ba51134f0>

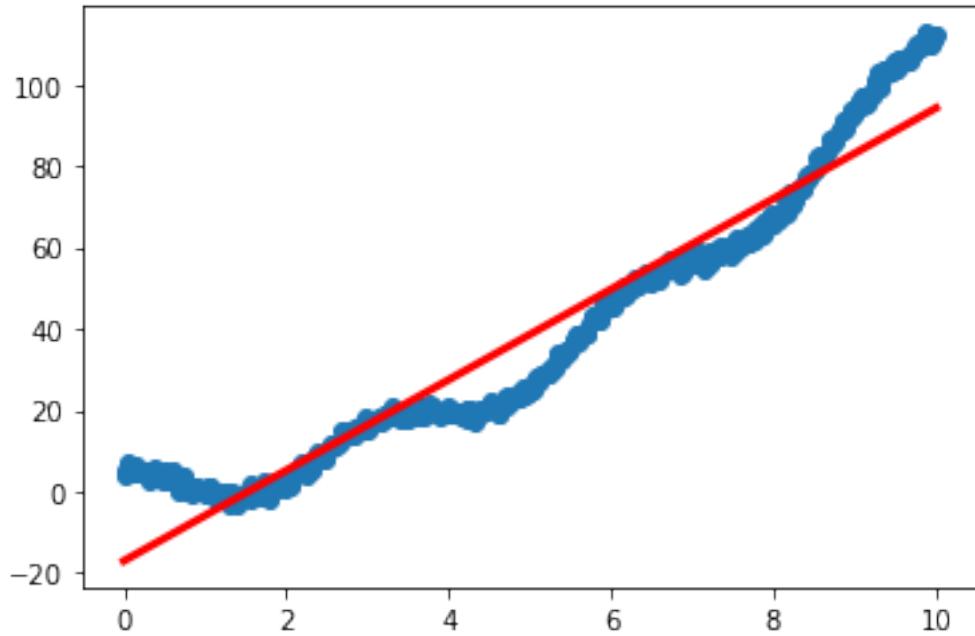


### 4.3.3 Polynomial Features

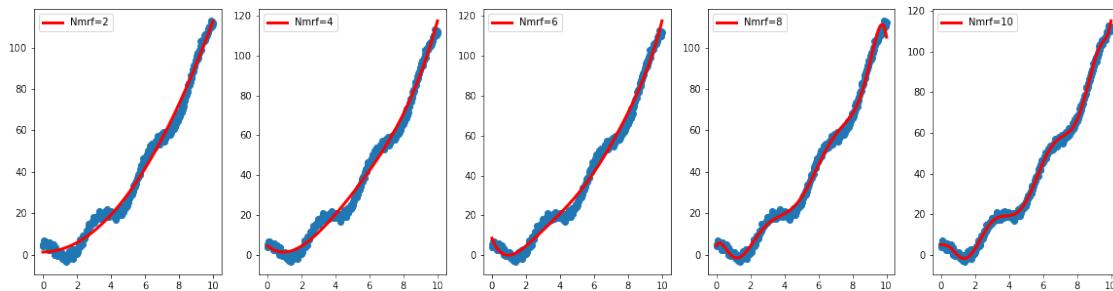
```
[46]: from sklearn.preprocessing import PolynomialFeatures  
from sklearn.linear_model import LinearRegression
```

```
[56]: m = 500  
X = np.linspace(0, 10, m).reshape((m, 1))  
y = X**2 + X + 5*np.cos(2*X) + np.random.randn(m, 1)  
  
model = LinearRegression().fit(X, y)  
y_pred = model.predict(X)  
  
plt.scatter(X, y)  
plt.plot(X, y_pred, c='r', lw=3)
```

[56]: [<matplotlib.lines.Line2D at 0x7f7ba53745b0>]



```
[62]: step = 2
plt.figure(figsize=(20,5))
for i in range(0,5):
    X_poly = PolynomialFeatures(step).fit_transform(X)
    model = LinearRegression().fit(X_poly, y)
    y_pred = model.predict(X_poly)
    plt.subplot(1,5,i+1)
    plt.scatter(X, y)
    plt.plot(X, y_pred, c='r', lw=3,label=f'Nmrf={step}')
    step+=2
plt.legend()
```



### 4.3.4 Discretisation

```
[63]: from sklearn.preprocessing import Binarizer, KBinsDiscretizer
```

```
[65]: X = np.linspace(0, 5, 10).reshape((10, 1))
X
```

```
[65]: array([[0.          ],
   [0.55555556],
   [1.11111111],
   [1.66666667],
   [2.22222222],
   [2.77777778],
   [3.33333333],
   [3.88888889],
   [4.44444444],
   [5.          ]])
```

```
[67]: np.hstack((X, Binarizer(threshold=3).fit_transform(X)))
```

```
[67]: array([[0.        , 0.        ],
   [0.55555556, 0.        ],
   [1.11111111, 0.        ],
   [1.66666667, 0.        ],
   [2.22222222, 0.        ],
   [2.77777778, 0.        ],
   [3.33333333, 1.        ],
   [3.88888889, 1.        ],
   [4.44444444, 1.        ],
   [5.        , 1.        ]])
```

```
[68]: KBinsDiscretizer(n_bins=6).fit_transform(X).toarray()
```

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

## 4.4 Pipelines

```
[69]: from sklearn.pipeline import make_pipeline
      from sklearn.linear_model import SGDClassifier
      from sklearn.model_selection import train_test_split
      from sklearn.preprocessing import StandardScaler
      from sklearn.datasets import load_iris
```

```
[70]: iris = load_iris()

X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

```
[71]: # Traditional pipeline

#transformer
scaler = StandardScaler()
X_train_transformed = scaler.fit_transform(X_train)

#Estimator
model = SGDClassifier(random_state=0)
model.fit(X_train_transformed,y_train)

# test

X_test_transformed = scaler.transform(X_test)
print(model.score(X_test_transformed,y_test))
```

0.9736842105263158

#### 4.4.1 Cree une pipeline

```
[73]: model = make_pipeline(StandardScaler(), SGDClassifier(random_state=0))

model.fit(X_train, y_train)
model.score(X_test, y_test)
```

[73]: 0.9736842105263158

```
[75]: from sklearn.model_selection import GridSearchCV
```

```

params = {
    'polynomialfeatures__degree':[2, 3, 4],
    'sgdclassifier__penalty':['l1', 'l2']
}

grid = GridSearchCV(model, param_grid=params, cv=4)

grid.fit(X_train, y_train)

```

[76]: GridSearchCV(cv=4,  
estimator=Pipeline(steps=[('polynomialfeatures',  
PolynomialFeatures()),  
('standardscaler', StandardScaler()),  
('sgdclassifier',  
SGDClassifier(random\_state=0))]),  
param\_grid={'polynomialfeatures\_\_degree': [2, 3, 4],  
'sgdclassifier\_\_penalty': ['l1', 'l2']}))

[77]: grid.score(X\_test, y\_test)

[77]: 0.9736842105263158

[79]: model = SGDClassifier(random\_state=0)  
model.fit(X\_train,y\_train)  
print(model.score(X\_test,y\_test))

0.8421052631578947

#### 4.4.2 Imputers

[12]: from sklearn.impute import SimpleImputer  
import numpy as np

[23]: X= np.array([[10,3],  
[3,5],  
[5,6],  
[np.nan,3]])  
imputer = SimpleImputer(missing\_values=np.nan,strategy='mean')  
imputer2 = SimpleImputer(missing\_values=np.nan,strategy='median')  
imputer3 = SimpleImputer(missing\_values=np.nan,strategy='most\_frequent')  
imputer4 = SimpleImputer(missing\_values=np.nan,strategy='constant',  
fill\_value=99)

print(imputer.fit\_transform(X))
print(imputer2.fit\_transform(X))
print(imputer3.fit\_transform(X))
print(imputer4.fit\_transform(X))

```
[[10.  3.]
 [ 3.  5.]
 [ 5.  6.]
 [ 6.  3.]]
[[10.  3.]
 [ 3.  5.]
 [ 5.  6.]
 [ 5.  3.]]
[[10.  3.]
 [ 3.  5.]
 [ 5.  6.]
 [ 3.  3.]]
[[10.  3.]
 [ 3.  5.]
 [ 5.  6.]
 [99.  3.]]
```

#### 4.4.3 Make Column transformer

```
[10]: import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split,GridSearchCV
```

```
[11]: titanic = sns.load_dataset('titanic')
titanic = titanic.drop(['class',
                       , 'embarked', 'alive'], axis=1)
titanic.dropna(axis=0, inplace=True)
y = titanic['survived']
X = titanic.drop('survived', axis=1)
numerical_features = ['pclass', 'age', 'fare', 'sibsp']
categorical_features = ['sex', 'deck', 'alone', 'who', 'adult_male', 'embark_town']

X_train, X_test, y_train, y_test = train_test_split(X, y)
print(y.shape)
print(X.shape)
titanic.head()
```

```
(182,)
(182, 11)
```

```
[11]:    survived  pclass      sex   age  sibsp  parch      fare      who  adult_male \
1          1        1  female  38.0      1       0    71.2833  woman      False
3          1        1  female  35.0      1       0    53.1000  woman      False
6          0        1   male   54.0      0       0    51.8625   man       True
10         1        3  female   4.0      1       1   16.7000  child      False
```

```

11          1         1  female  58.0      0      0  26.5500  woman      False
          deck  embark_town  alone
1       C     Cherbourg  False
3       C   Southampton  False
6       E   Southampton  True
10      G   Southampton  False
11      C   Southampton  True

```

```
[8]: from sklearn.compose import make_column_transformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder, MinMaxScaler
from sklearn.impute import SimpleImputer
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import SGDClassifier
from sklearn.neighbors import KNeighborsClassifier
```

```
[58]: transformer = make_column_transformer((StandardScaler(), ['age', 'fare']))
transformer.fit(X)

model = make_pipeline(transformer, SGDClassifier())
model.fit(X, y)
```

```
[58]: Pipeline(steps=[('columntransformer',
                     ColumnTransformer(transformers=[('standardscaler',
                                                     StandardScaler(),
                                                     ['age', 'fare'])])),
                     ('sgdclassifier', SGDClassifier())])
```

```
[100]: numerical_pipeline = make_pipeline(SimpleImputer(), StandardScaler())
categorical_pipeline = make_pipeline(SimpleImputer(
    strategy='most_frequent'), OneHotEncoder())
preprocessor = make_column_transformer((numerical_pipeline, numerical_features),
                                      (categorical_pipeline,
                                       categorical_features))
```

```
[101]: model = make_pipeline(preprocessor, SGDClassifier())
```

```
[106]: params = {
        'sgdclassifier_penalty':['l1', 'l2']
    }
grid = GridSearchCV(model, param_grid=params, cv=10)
grid.fit(X_train, y_train)
print(grid.score(X_test,y_test))
```

```
[108]: numerical_pipeline = make_pipeline(SimpleImputer(), StandardScaler())
categorical_pipeline = make_pipeline(SimpleImputer(
    strategy='most_frequent'), OneHotEncoder())

preprocessor = make_column_transformer((numerical_pipeline, numerical_features),
                                      (categorical_pipeline, categorical_features))
model = make_pipeline(preprocessor, KNeighborsClassifier())
params = {
    'kneighborsclassifier__n_neighbors': np.arange(1, 20),
    'kneighborsclassifier__metric': ['minkowski', 'euclidean', 'manhattan']
}
grid = GridSearchCV(model, param_grid=params, cv=5)
grid.fit(X_train, y_train)
print(grid.score(X_test,y_test))
```

0.717391304347826

#### 4.4.4 Make column selector

```
[6]: from sklearn.compose import make_column_selector
```

```
[9]: numerical_features = make_column_selector(dtype_include=np.number)
categorical_features = make_column_selector(dtype_exclude=np.number)

numerical_pipeline = make_pipeline(SimpleImputer(), StandardScaler())
categorical_pipeline = make_pipeline(SimpleImputer(
    strategy='most_frequent'), OneHotEncoder())

preprocessor = make_column_transformer((numerical_pipeline, numerical_features),
                                      (categorical_pipeline, categorical_features))
model = make_pipeline(preprocessor, KNeighborsClassifier())
params = {
    'kneighborsclassifier__n_neighbors': np.arange(1, 20),
    'kneighborsclassifier__metric': ['minkowski', 'euclidean', 'manhattan']
}
grid = GridSearchCV(model, param_grid=params, cv=5)
grid.fit(X_train, y_train)
print(grid.score(X_test,y_test))
```

0.8043478260869565

#### 4.4.5 Exercice

##### Exercice 1

En utilisant le dernier exemple, essayez les différents transformateurs de données de normalisation pour trouver le pipeline le plus précis

```
[84]: modelSS = make_pipeline(PolynomialFeatures(),
                           StandardScaler(),
                           SGDClassifier(random_state=0))

params = {
    'polynomialfeatures__degree':[2, 3, 4],
    'sgdclassifier__penalty':['l1', 'l2']
}
grid = GridSearchCV(modelSS, param_grid=params, cv=4)
grid.fit(X_train, y_train)
print('StandardScaler:', grid.score(X_test, y_test))

modelMM = make_pipeline(MinMaxScaler(),
                       SGDClassifier(random_state=0))
params = {
    'sgdclassifier__penalty':['l1', 'l2']
}
grid = GridSearchCV(modelMM, param_grid=params, cv=4)
grid.fit(X_train, y_train)
print('MinMaxScaler:', grid.score(X_test, y_test))

modelRS = make_pipeline(RobustScaler(),
                       SGDClassifier(random_state=0))
params = {
    'sgdclassifier__penalty':['l1', 'l2']
}
grid = GridSearchCV(modelRS, param_grid=params, cv=4)
grid.fit(X_train, y_train)
print('RobustScaler:', grid.score(X_test, y_test))
```

StandardScaler: 0.9736842105263158

MinMaxScaler: 0.9210526315789473

RobustScaler: 0.9736842105263158

##### Exercice 2

Appliquez les différentes techniques apprises dans cette partie pour construire un pipeline sur le jeu de données du Titanic. essayez de trouver le meilleur transformateur de données avec un classificateur qui vous donne les meilleurs résultats.

```
[179]: import seaborn as sns
titanic = sns.load_dataset('titanic')
#titanic = titanic[['survived', 'pclass', 'sex', 'age']]
titanic.dropna(axis=0, inplace=True)
titanic.head()
#titanic['sex'].replace(['male', 'female'], [0, 1], inplace=True)
y= titanic['survived']
X= titanic.drop('survived',axis=1)
```

```
[180]: encoder = OrdinalEncoder()
X=encoder.fit_transform(X)
```

```
[181]: X_train,X_test,y_train,y_test = train_test_split(X,y)
print(X_train.shape,X_test.shape, y_train.shape, y_test.shape )
```

(136, 14) (46, 14) (136,) (46,)

```
[182]: modelSS = make_pipeline(PolynomialFeatures(),
                             StandardScaler(),
                             SGDClassifier(random_state=0))

params = {
    'polynomialfeatures__degree':[2, 3, 4],
    'sgdclassifier__penalty':['l1', 'l2']
}
grid = GridSearchCV(modelSS, param_grid=params, cv=5)
grid.fit(X_train, y_train)
print('StandardScaler:', grid.score(X_test, y_test))

modelMM = make_pipeline(MinMaxScaler(),
                       SGDClassifier(random_state=0))
params = {
    'sgdclassifier__penalty':['l1', 'l2']
}
grid = GridSearchCV(modelMM, param_grid=params, cv=5)
grid.fit(X_train, y_train)
print('MinMaxScaler:', grid.score(X_test, y_test))

modelRS = make_pipeline(RobustScaler(),
                       SGDClassifier(random_state=0))
params = {
    'sgdclassifier__penalty':['l1', 'l2']
}
grid = GridSearchCV(modelRS, param_grid=params, cv=5)
grid.fit(X_train, y_train)
```

```
print('RobustScaler:', grid.score(X_test, y_test))
```

StandardScaler: 1.0  
 MinMaxScaler: 1.0  
 RobustScaler: 0.9782608695652174

```
[191]: from sklearn.neighbors import KNeighborsClassifier
modelSS = make_pipeline(PolynomialFeatures(),
                       StandardScaler(),
                       KNeighborsClassifier())
params = {
    'polynomialfeatures_degree':[2, 3, 4],
    'kneighborsclassifier_n_neighbors': np.arange(1, 20),
    'kneighborsclassifier_metric': ['minkowski','euclidean', 'manhattan']}
}
grid = GridSearchCV(modelSS, param_grid=params, cv=10)
grid.fit(X_train, y_train)
print('StandardScaler:', grid.score(X_test, y_test))

modelMM = make_pipeline(MinMaxScaler(),
                       KNeighborsClassifier())
params = {
    'kneighborsclassifier_n_neighbors': np.arange(1, 20),
    'kneighborsclassifier_metric': ['minkowski','euclidean', 'manhattan']}
}
grid = GridSearchCV(modelMM, param_grid=params, cv=10)
grid.fit(X_train, y_train)
print('MinMaxScaler:', grid.score(X_test, y_test))

modelRS = make_pipeline(RobustScaler(),
                       KNeighborsClassifier())
params = {
    'kneighborsclassifier_n_neighbors': np.arange(1, 20),
    'kneighborsclassifier_metric': ['minkowski','euclidean', 'manhattan']}
}
grid = GridSearchCV(modelRS, param_grid=params, cv=10)
grid.fit(X_train, y_train)
print('RobustScaler:', grid.score(X_test, y_test))
```

StandardScaler: 0.9347826086956522  
 MinMaxScaler: 0.9565217391304348  
 RobustScaler: 0.9347826086956522

## 4.5 Feature Selection

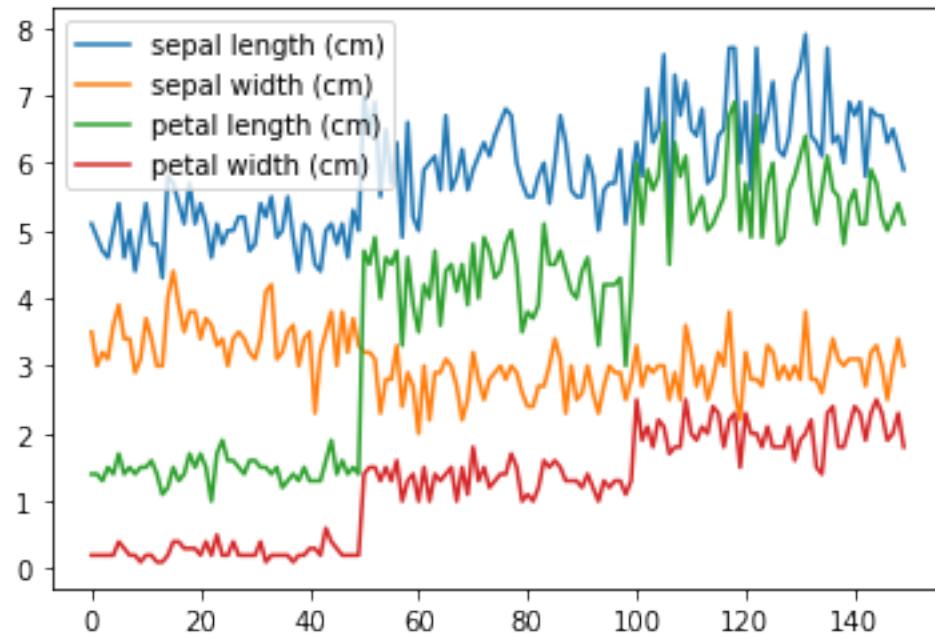
```
[1]: import numpy as np  
import matplotlib.pyplot as plt  
import pandas as pd
```

### 4.5.1 Variance Threshold

```
[2]: from sklearn.feature_selection import VarianceThreshold
```

```
[3]: from sklearn.datasets import load_iris  
  
iris = load_iris()  
X = iris.data  
y = iris.target  
  
plt.plot(X)  
plt.legend(iris.feature_names)
```

```
[3]: <matplotlib.legend.Legend at 0x7f8b2d9ccf70>
```



```
[4]: X.var(axis=0)
```

```
[4]: array([0.68112222, 0.18871289, 3.09550267, 0.57713289])
```

```
[5]: selector = VarianceThreshold(threshold=0.2)
selector.fit(X)

[5]: VarianceThreshold(threshold=0.2)

[6]: selector.get_support()

[6]: array([ True, False,  True,  True])

[7]: np.array(iris.feature_names)[selector.get_support()]

[7]: array(['sepal length (cm)', 'petal length (cm)', 'petal width (cm)'],
      dtype='|<U17')

[8]: selector.variances_

[8]: array([0.68112222, 0.18871289, 3.09550267, 0.57713289])
```

### 4.5.2 SelectKBest

```
[9]: from sklearn.feature_selection import SelectKBest, chi2, f_classif

[16]: chi2(X, y)

[16]: (array([ 10.81782088,   3.7107283 , 116.31261309,   67.0483602 ]),
      array([4.47651499e-03, 1.56395980e-01, 5.53397228e-26, 2.75824965e-15]))

[19]: selector = SelectKBest(f_classif, k=2)
selector.fit(X, y)
selector.scores_

[19]: array([ 119.26450218,   49.16004009, 1180.16118225,   960.0071468 ])

[23]: np.array(iris.feature_names)[selector.get_support()]

[23]: array(['petal length (cm)', 'petal width (cm)'], dtype='|<U17')
```

### 4.5.3 Recursive feature Elimination

```
[16]: from sklearn.feature_selection import RFECV
from sklearn.linear_model import SGDClassifier

[18]: selector = RFECV(SGDClassifier(random_state=0), step=1, ↴
      min_features_to_select=2, cv=5)
selector.fit(X, y)
print(selector.ranking_)
```

```
print(selector.grid_scores_)
```

```
[2 1 1 1]  
[0.8 0.84666667 0.77333333]
```

```
[19]: np.array(iris.feature_names)[selector.get_support()]
```

```
[19]: array(['sepal width (cm)', 'petal length (cm)', 'petal width (cm)'],
      dtype='|<U17')
```

#### 4.5.4 SelectFromModel

```
[12]: from sklearn.feature_selection import SelectFromModel  
from sklearn.linear_model import SGDClassifier
```

```
[14]: X = iris.data
y = iris.target
selector = SelectFromModel(SGDClassifier(random_state=0), threshold='mean')
selector.fit(X, y)
selector.estimator_.coef_
```

```
[14]: array([[ 8.64029104,  27.2851296 , -40.01819009, -17.73533424],  
           [-5.48888269, -58.79616709,  22.88584985, -54.14457159],  
           [-81.28026953, -75.17372078, 130.76437145, 131.39608339]])
```

```
[20]: np.array(iris.feature_names)[selector.get_support()]
```

```
[20]: array(['sepal width (cm)', 'petal length (cm)', 'petal width (cm)'],
      dtype='|<U17')
```

#### 4.5.5 Exercice

Sur la base des techniques que nous avons apprises sur la sélection des features, déterminez les caractéristiques les plus pertinentes pour l'ensemble de données du Titanic.

```
[21]: import seaborn as sns
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder, ▾
    ↪MinMaxScaler, OrdinalEncoder
from sklearn.impute import SimpleImputer
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import SGDClassifier
from sklearn.neighbors import KNeighborsClassifier
```

```
[22]: titanic = sns.load_dataset('titanic')
       titanic = titanic.drop(['class',
                               'embarked', 'alive'],
                               axis=1)
```

```
titanic.dropna(axis=0, inplace=True)
y = titanic['survived']
X = titanic.drop('survived', axis=1)
numerical_features = ['pclass', 'age', 'fare', 'sibsp', 'parch']
categorical_features = ['sex', 'deck', 'who', 'adult_male', 'embark_town', 'alone']
print(titanic.head())
print(y.shape)
print(X.shape)

numerical_pipeline = make_pipeline(SimpleImputer(), MinMaxScaler())
categorical_pipeline = make_pipeline(SimpleImputer(
    strategy='most_frequent'), OrdinalEncoder())
preprocessor = make_column_transformer((numerical_pipeline, numerical_features),
                                      (categorical_pipeline, categorical_features))

preprocessor.fit_transform(X,y)
Xx= preprocessor.transform(X)
```

	survived	pclass	sex	age	sibsp	parch	fare	who	adult_male
1	1	1	female	38.0	1	0	71.2833	woman	False
3	1	1	female	35.0	1	0	53.1000	woman	False
6	0	1	male	54.0	0	0	51.8625	man	True
10	1	3	female	4.0	1	1	16.7000	child	False
11	1	1	female	58.0	0	0	26.5500	woman	False

	deck	embark_town	alone
1	C	Cherbourg	False
3	C	Southampton	False
6	E	Southampton	True
10	G	Southampton	False
11	C	Southampton	True

(182,)  
(182, 11)

```
[23]: import sklearn.feature_selection

selector = sklearn.feature_selection.SelectKBest(chi2, k=5)
selector.fit_transform(Xx,y)
print(X.columns[selector.get_support()])

selector = sklearn.feature_selection.SelectFpr(alpha =0.01)
selector.fit_transform(Xx,y)
print(X.columns[selector.get_support()])

selector = sklearn.feature_selection.SelectFdr(alpha =0.01)
selector.fit_transform(Xx,y)
```

```

print(X.columns[selector.get_support()])

selector = sklearn.feature_selection.SelectFwe(alpha =0.01)
selector.fit_transform(Xx,y)
print(X.columns[selector.get_support()])

selector = sklearn.feature_selection.SelectPercentile(percentile= 10)
selector.fit_transform(Xx,y)
print(X.columns[selector.get_support()])

```

```

Index(['fare', 'adult_male', 'deck', 'embark_town', 'alone'], dtype='object')
Index(['sex', 'fare', 'adult_male', 'deck'], dtype='object')
Index(['sex', 'fare', 'adult_male', 'deck'], dtype='object')
Index(['sex', 'fare', 'adult_male', 'deck'], dtype='object')
Index(['deck'], dtype='object')

```

[24]:

```

from sklearn.feature_selection import SelectFromModel
from sklearn.linear_model import SGDClassifier

selector = SelectFromModel(SGDClassifier(random_state=0), threshold='mean')
selector.fit(Xx, y)
selector.estimator_.coef_

print(X.columns[selector.get_support()])

```

```
Index(['pclass', 'sex', 'parch', 'fare', 'deck'], dtype='object')
```

[ ]:

## 4.6 Apprentissage Non-Supervisé

[2]:

```

import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs

```

### 4.6.1 Clustering

#### Kmean Algorithm

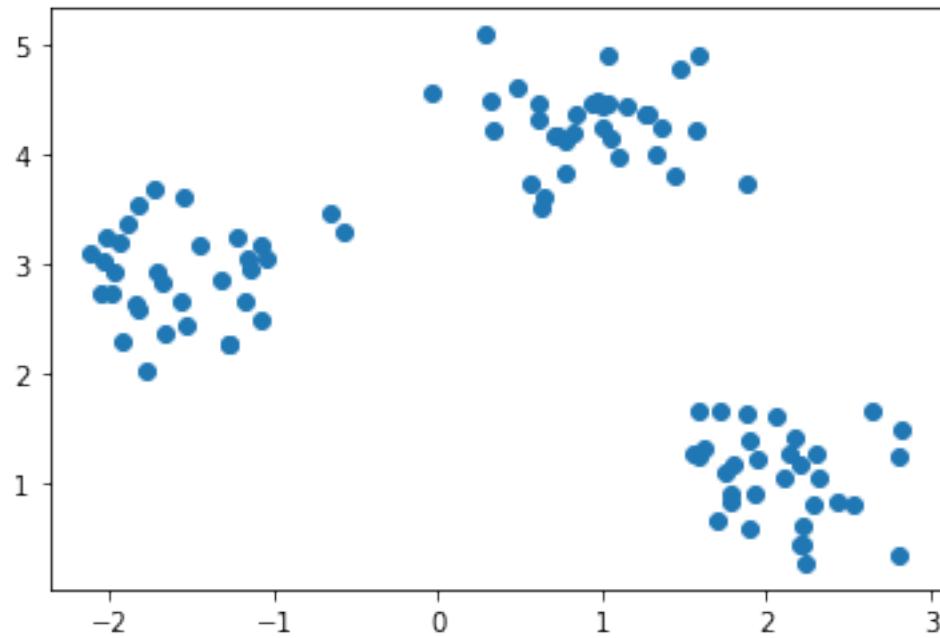
[7]:

```
from sklearn.cluster import KMeans
```

[2]:

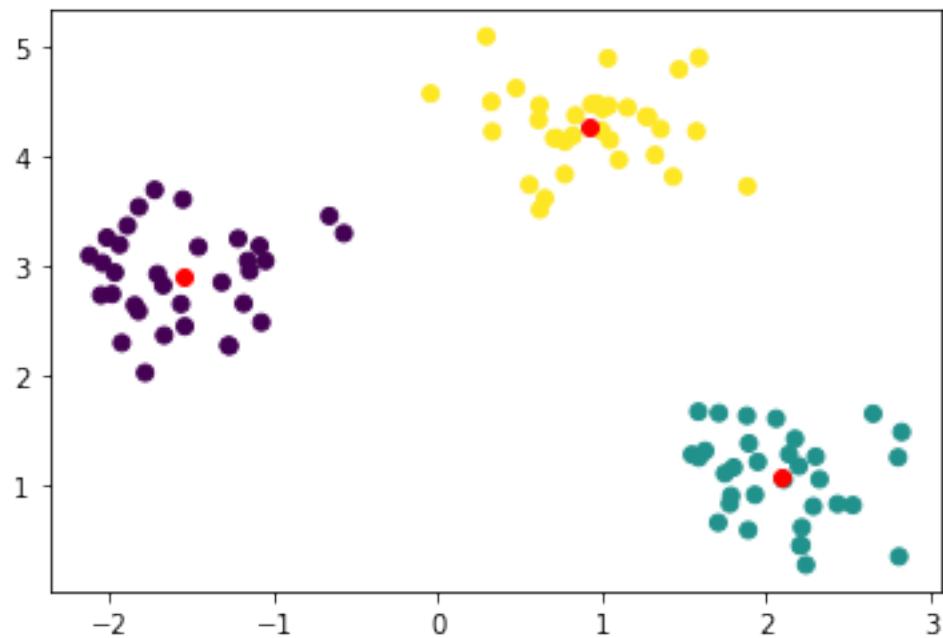
```
# Génération de données
X, y = make_blobs(n_samples=100, centers=3, cluster_std=0.4, random_state=0)
plt.scatter(X[:,0], X[:,1])
```

[2]: <matplotlib.collections.PathCollection at 0x7f2f263276d8>



```
[4]: model = KMeans(n_clusters=3)
model.fit(X)
model.predict(X)
plt.scatter(X[:,0], X[:,1], c=model.predict(X))
plt.scatter(model.cluster_centers_[:,0], model.cluster_centers_[:,1], c='red')
model.score(X)
```

[4]: -30.870531280140696

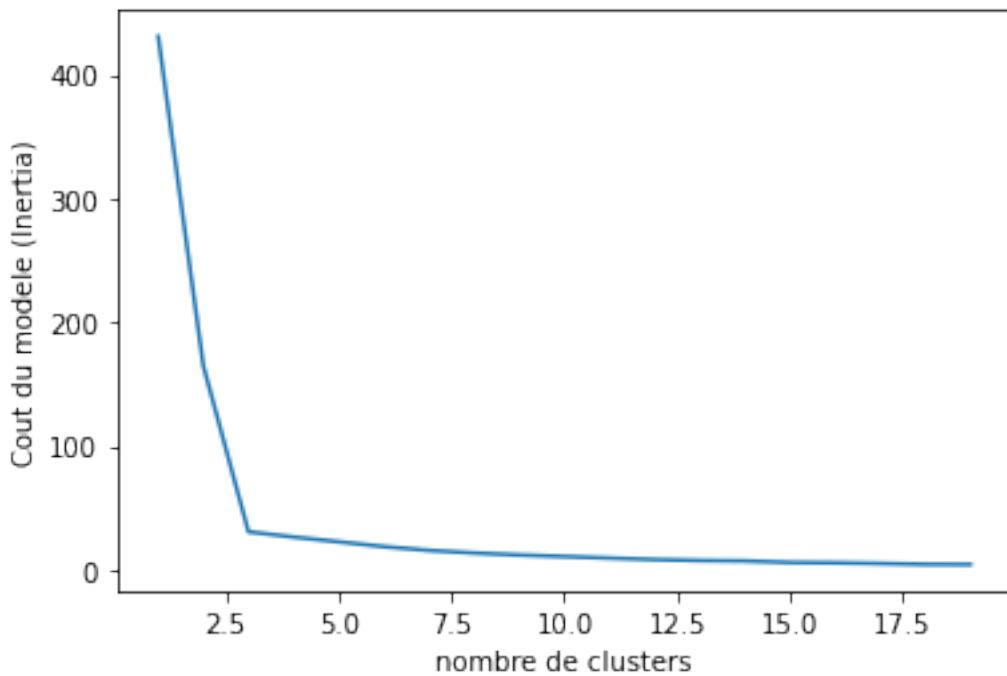


### Elbow Method

```
[6]: inertia = []
K_range = range(1, 20)
for k in K_range:
    model = KMeans(n_clusters=k).fit(X)
    inertia.append(model.inertia_)

plt.plot(K_range, inertia)
plt.xlabel('nombre de clusters')
plt.ylabel('Cout du modèle (Inertia)')
```

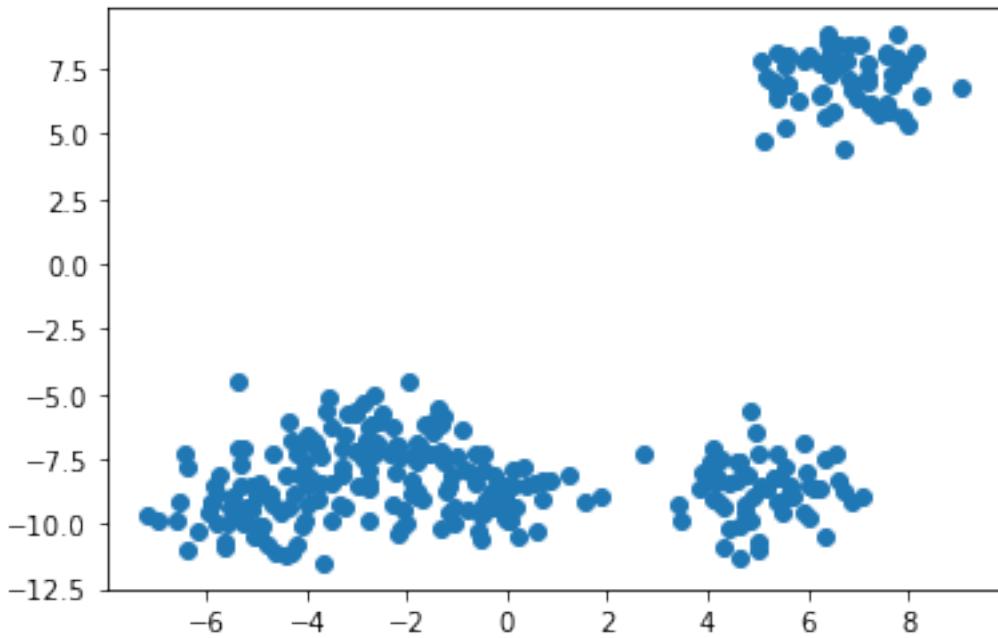
```
[6]: Text(0, 0.5, 'Cout du modèle (Inertia)')
```



### Autre méthode de clustering

```
[8]: from sklearn.cluster import*
      →KMeans,DBSCAN,AgglomerativeClustering,SpectralClustering
from sklearn import metrics
X, y = make_blobs(n_samples=300, centers=5, cluster_std=1.0)
plt.scatter(X[:,0], X[:,1])
```

```
[8]: <matplotlib.collections.PathCollection at 0x7fe95368f970>
```



```
[5]: model = DBSCAN().fit(X)

core_samples_mask = np.zeros_like(model.labels_, dtype=bool)
core_samples_mask[model.core_sample_indices_] = True
labels = model.labels_

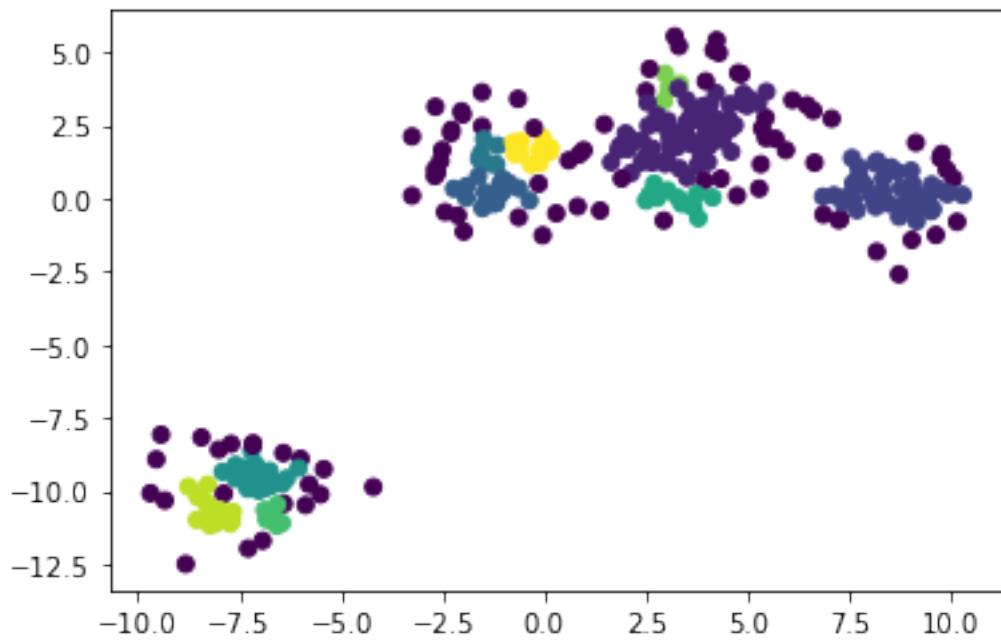
# Number of clusters in labels, ignoring noise if present.
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)

plt.scatter(X[:,0], X[:,1], c=model.fit_predict(X))

print("Estimated number of clusters: %d" % n_clusters_)
print("Estimated number of noise points: %d" % n_noise_)
print("Homogeneity: %0.3f" % metrics.homogeneity_score(y, labels))
print("Completeness: %0.3f" % metrics.completeness_score(y, labels))
print("V-measure: %0.3f" % metrics.v_measure_score(y, labels))
print("Adjusted Rand Index: %0.3f" % metrics.adjusted_rand_score(y, labels))
print(
    "Adjusted Mutual Information: %0.3f"
    % metrics.adjusted_mutual_info_score(y, labels)
)
print("Silhouette Coefficient: %0.3f" % metrics.silhouette_score(X, labels))
```

Estimated number of clusters: 10  
 Estimated number of noise points: 90

Homogeneity: 0.602  
 Completeness: 0.499  
 V-measure: 0.546  
 Adjusted Rand Index: 0.306  
 Adjusted Mutual Information: 0.526  
 Silhouette Coefficient: 0.057



```
[9]: plt.figure(figsize=(20,5))
model = DBSCAN(eps=0.6).fit(X)
plt.subplot(1,4,1)
plt.scatter(X[:,0], X[:,1], c=model.fit_predict(X))

model = AgglomerativeClustering(n_clusters=5).fit(X)
plt.subplot(1,4,2)
plt.scatter(X[:,0], X[:,1], c=model.fit_predict(X))
R = model.fit_predict(X)
s='AC:'
for i in range(0,5):
    s+= f'{R[R==i].size} '
plt.title(s)

model = SpectralClustering(n_clusters=5).fit(X)
plt.subplot(1,4,3)
plt.scatter(X[:,0], X[:,1], c=model.fit_predict(X))
R = model.fit_predict(X)
```

```
s='SC: '
for i in range(0,5):
    s+= f'{R[R[:]==i].size}'

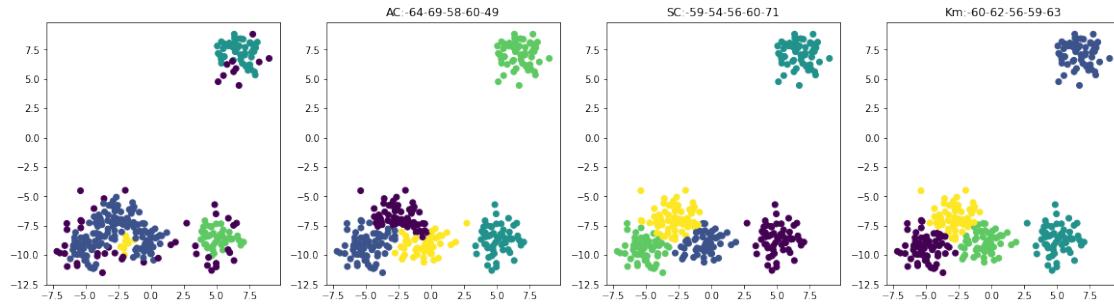
plt.title(s)

model = KMeans(n_clusters=5).fit(X)
plt.subplot(1,4,4)
plt.scatter(X[:,0], X[:,1], c=model.fit_predict(X))
R =model.fit_predict(X)
s='Km: '
for i in range(0,5):
    s+= f'{R[R[:]==i].size}'

plt.title(s)

print(R[R[:]==0].size)
```

60

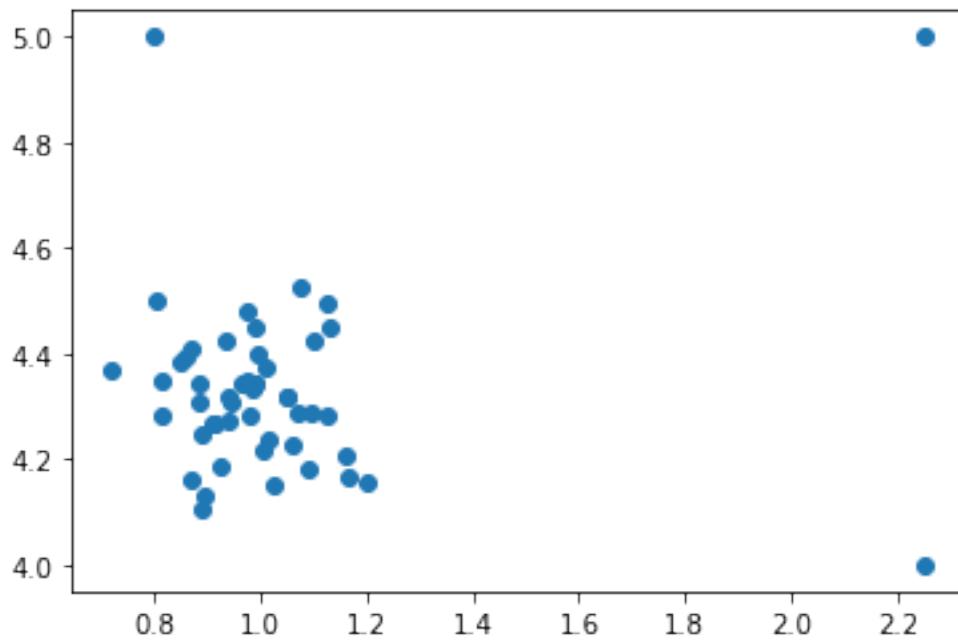


#### 4.6.2 Détection d'anomalies avec Isolation Forest

[22]: `from sklearn.ensemble import IsolationForest`

[25]: `X, y = make_blobs(n_samples=50, centers=1, cluster_std=0.1, random_state=0)`  
`X[-1,:] = np.array([2.25, 5])`  
`X[-2,:] = np.array([2.25, 4])`  
`X[-3,:] = np.array([0.8, 5])`  
`plt.scatter(X[:,0], X[:, 1])`

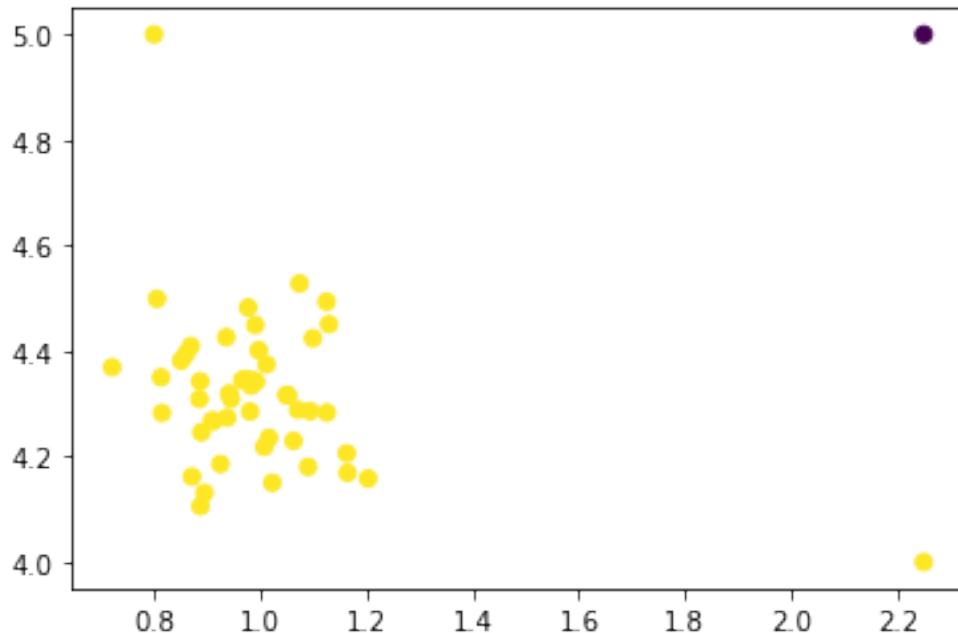
[25]: <matplotlib.collections.PathCollection at 0x7fe955c13460>



```
[28]: model = IsolationForest(contamination=3/300)
model.fit(X)

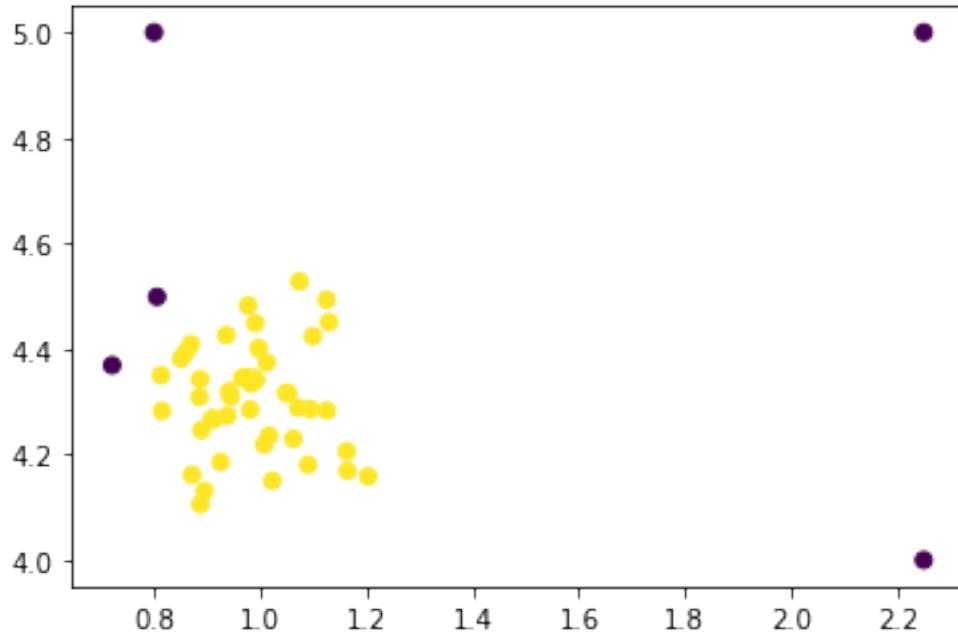
plt.scatter(X[:,0], X[:, 1], c=model.predict(X))
```

```
[28]: <matplotlib.collections.PathCollection at 0x7fe954737cd0>
```



```
[34]: from sklearn.neighbors import LocalOutlierFactor  
  
model = LocalOutlierFactor(n_neighbors=5).fit(X)  
plt.scatter(X[:,0], X[:, 1], c=model.fit_predict(X))
```

```
[34]: <matplotlib.collections.PathCollection at 0x7fe95438aa00>
```

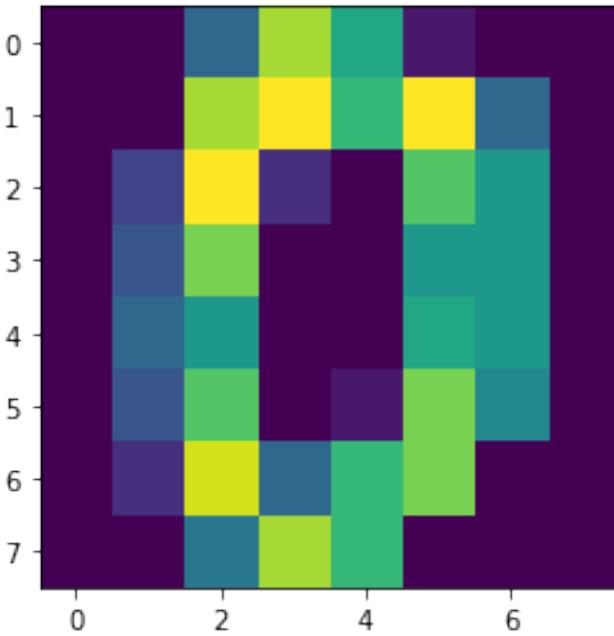


### Application : Digits Outliers

```
[38]: from sklearn.datasets import load_digits  
  
digits = load_digits()  
images = digits.images  
X = digits.data  
y = digits.target  
print(X.shape)  
plt.imshow(images[0])
```

(1797, 64)

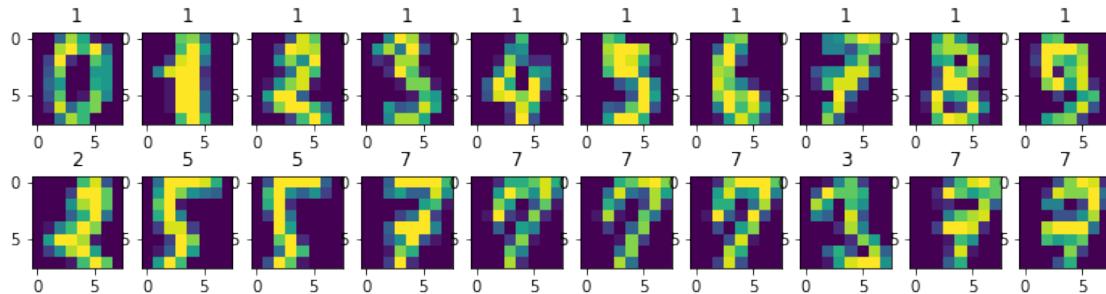
```
[38]: <matplotlib.image.AxesImage at 0x7fe955997d30>
```



```
[21]: model = IsolationForest(random_state=0, contamination=0.02)
model.fit(X)

Good = model.predict(X)
plt.figure(figsize=(12, 3))
for i in range(10):
    plt.subplot(2, 10, i+1)
    plt.imshow(images[i])
    plt.title(y[Good][i])

outliers = model.predict(X) == -1
for i in range(10):
    plt.subplot(2, 10, 10+i+1)
    plt.imshow(images[outliers][i])
    plt.title(y[outliers][i])
```



### 4.6.3 Reduction de dimension

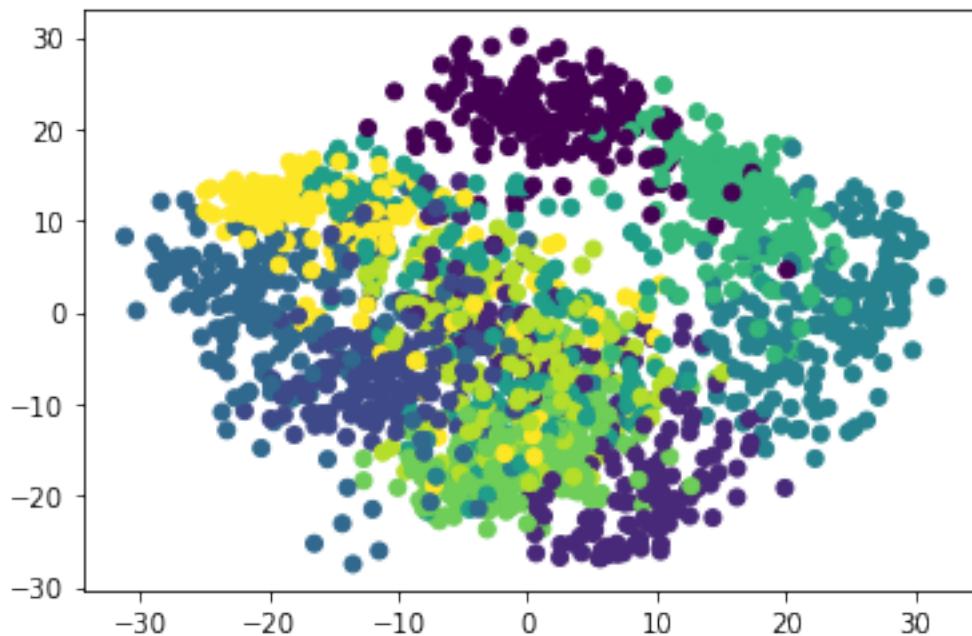
#### Visualisation 2D

```
[39]: from sklearn.decomposition import PCA  
  
model = PCA(n_components=2)  
model.fit(X)
```

```
[39]: PCA(n_components=2)
```

```
[40]: x_pca = model.transform(X)  
plt.scatter(x_pca[:,0], x_pca[:,1], c=y)
```

```
[40]: <matplotlib.collections.PathCollection at 0x7fe95476f760>
```



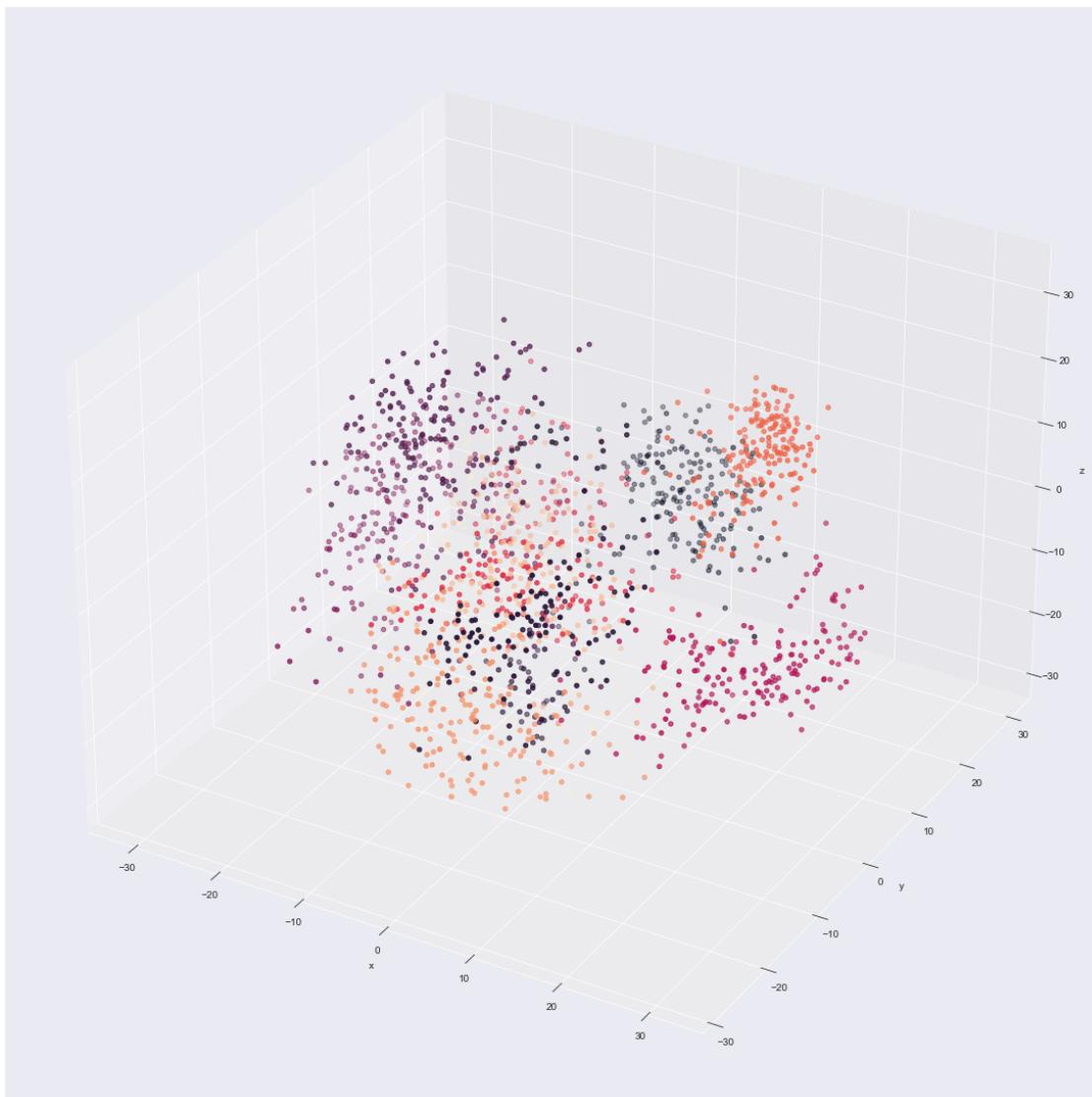
```
[51]: import seaborn as sns  
  
model = PCA(n_components=3)  
model.fit(X)  
x_pca = model.transform(X)  
  
sns.set_style("darkgrid")  
  
plt.figure(figsize=(20,20))
```

```
axes = plt.axes(projection='3d')
print(type(axes))
axes.scatter3D(x_pca[:,0], x_pca[:,1], x_pca[:,2],c=y)

axes.set_xlabel('x')
axes.set_ylabel('y')
axes.set_zlabel('z')
```

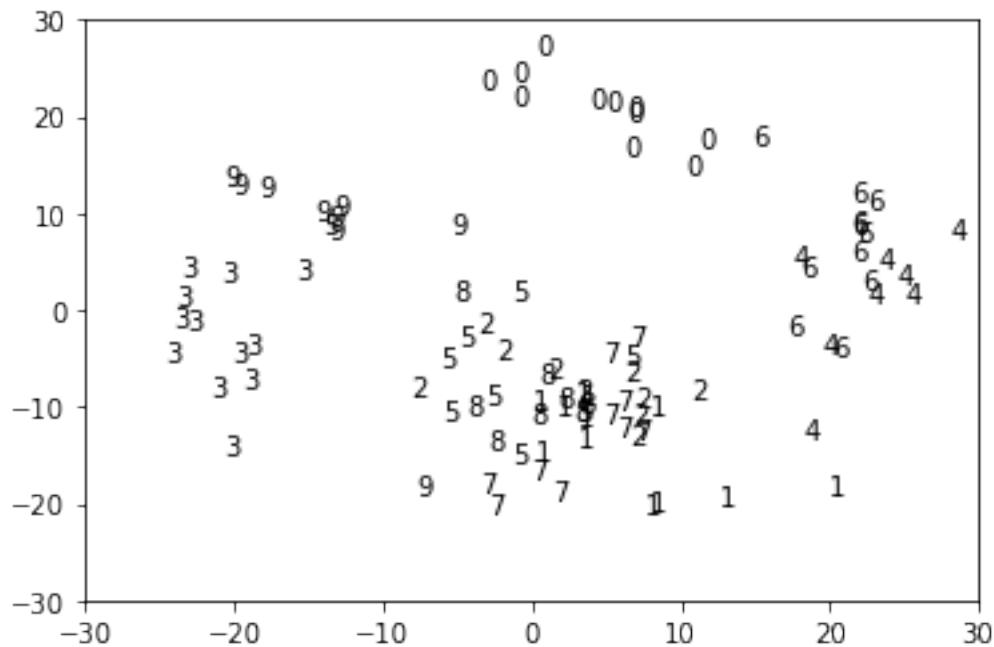
```
<class 'matplotlib.axes._subplots.Axes3DSubplot'>
```

```
[51]: Text(0.5, 0, 'z')
```



```
[41]: plt.figure()
plt.xlim(-30, 30)
plt.ylim(-30, 30)

for i in range(100):
    plt.text(x_pca[i,0], x_pca[i,1], str(y[i]))
```



### Compression de données

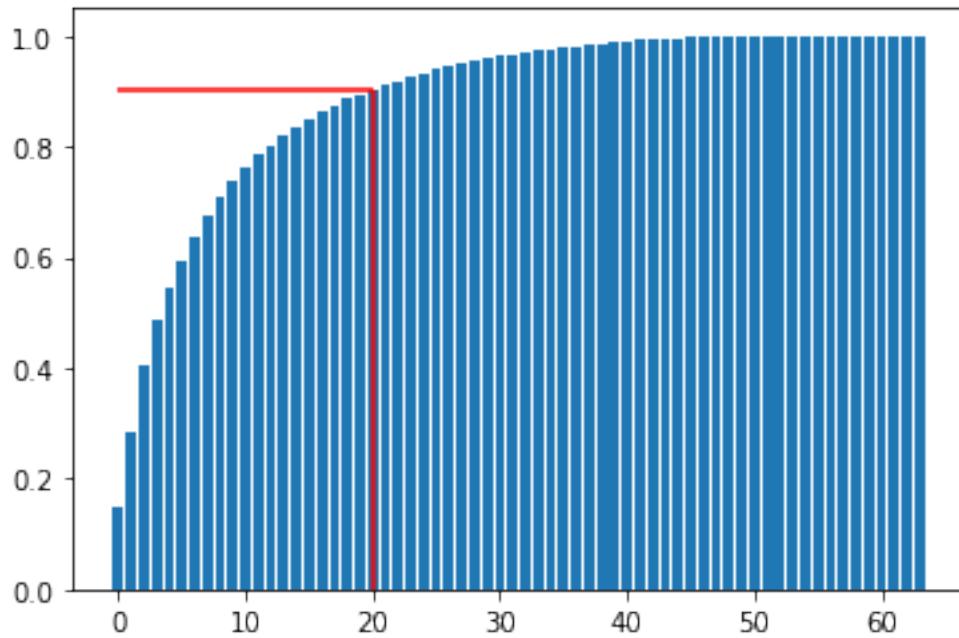
```
[42]: n_dims = X.shape[1]
model = PCA(n_components=n_dims)
model.fit(X)

variances = model.explained_variance_ratio_

meilleur_dims = np.argmax(np.cumsum(variances) > 0.90)

plt.bar(range(n_dims), np.cumsum(variances))
plt.hlines(0.90, 0, meilleur_dims, colors='r')
plt.vlines(meilleur_dims, 0, 0.90, colors='r')
```

[42]: <matplotlib.collections.LineCollection at 0x7fe955f5c1f0>



```
[43]: model = PCA(n_components=0.99)
model.fit(X)
```

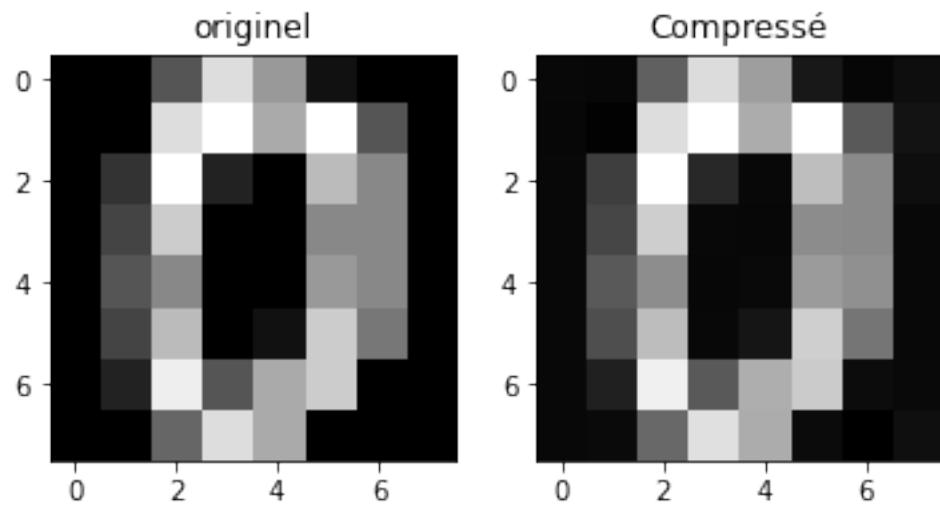
```
[43]: PCA(n_components=0.99)
```

```
[44]: X_compress = model.fit_transform(X)
X_decompress = model.inverse_transform(X_compress)

plt.subplot(1, 2, 1)
plt.imshow(X[0,:].reshape((8,8)), cmap='gray')
plt.title('originel')

plt.subplot(1, 2, 2)
plt.imshow(X_decompress[0,:].reshape((8,8)), cmap='gray')
plt.title('Compressé')
```

```
[44]: Text(0.5, 1.0, 'Compressé')
```



[ ]:



# Chapter 5

## Demarches pour un projet de science de donnees

### 5.1 Exploratory Data Analysis

#### 5.1.1 Objectif :

- Comprendre du mieux possible nos données
- Développer une première stratégie de modélisation

#### 5.1.2 Analyse de la forme des données

- **variable target** : SARS-Cov-2 exam result
- **lignes et colonnes** : 5644, 111
- **types de variables** : qualitatives : 70, quantitatives : 41
- **Analyse des valeurs manquantes** :
  - beaucoup de NaN (moitié des variables > 88% de NaN)
  - 2 groupes de données 76% -> Test viral, 89% -> taux sanguins

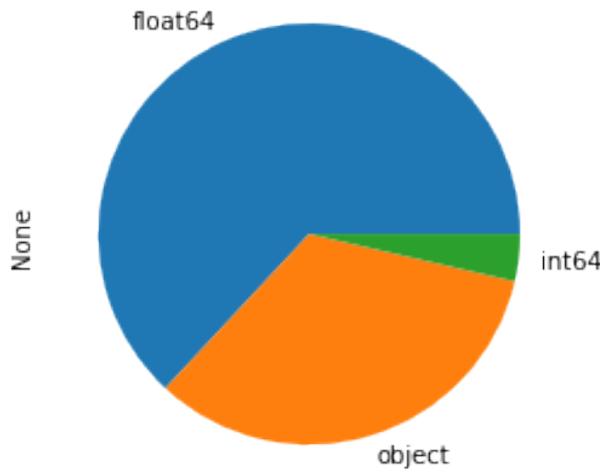
#### Visulation initiale - Elimination des colonnes inutiles

```
[5]: df = data.copy()  
df.head()
```

```
[6]: df.shape
```

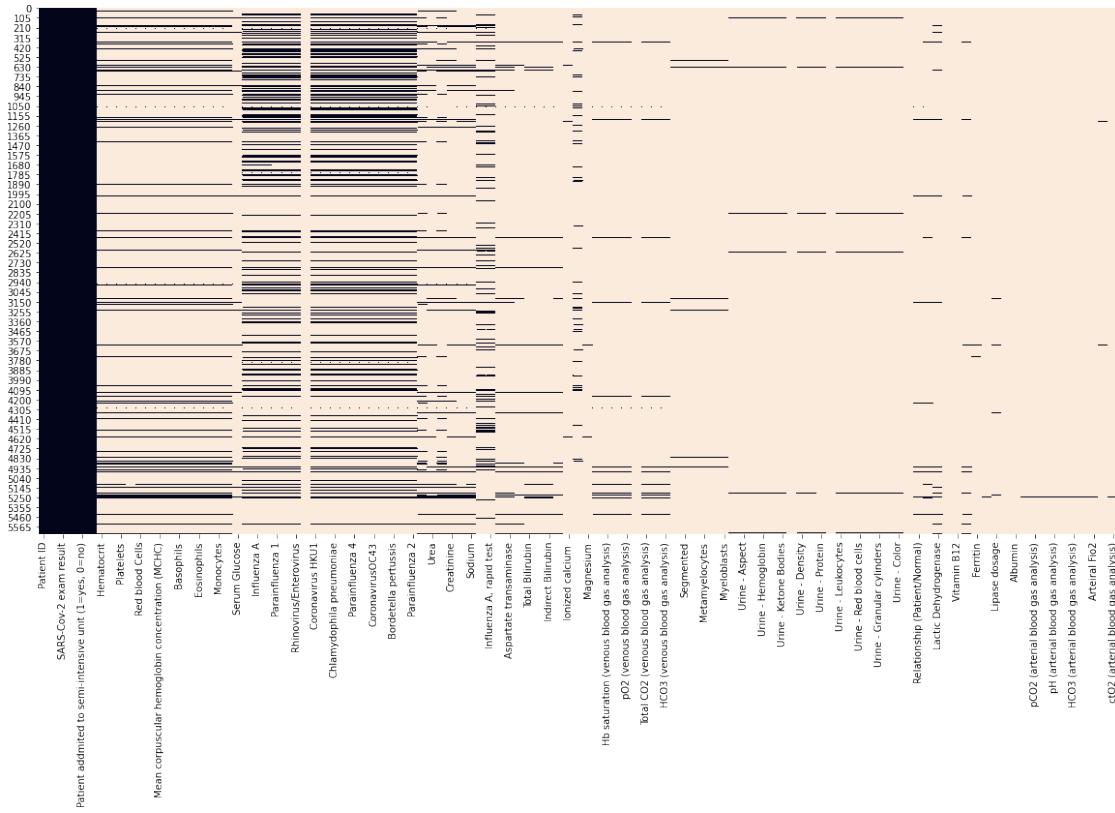
```
[6]: (5644, 111)
```

```
[7]: df.dtypes.value_counts().plot.pie()
```



```
[9]: plt.figure(figsize=(20,10))
sns.heatmap(df.isna(), cbar=False)
```

```
[9]: <AxesSubplot:>
```



```
[24]: print('Pourcentage des valeurs Null: ',df.isna().sum().sum()/(df.shape[0]*df.shape[1]))
print((df.isna().sum()/df.shape[0]).sort_values(ascending=True))
```

Pourcentage des valeurs Null:	0.8806003026414082
Patient ID	0.000000
Patient age quantile	0.000000
SARS-CoV-2 exam result	0.000000
Patient admitted to regular ward (1=yes, 0=no)	0.000000
Patient admitted to semi-intensive unit (1=yes, 0=no)	0.000000
Patient admitted to intensive care unit (1=yes, 0=no)	0.000000
Influenza B	0.760099
Respiratory Syncytial Virus	0.760099
Influenza A	0.760099
Rhinovirus/Enterovirus	0.760454
Inf A H1N1 2009	0.760454
CoronavirusOC43	0.760454
Coronavirus229E	0.760454
Parainfluenza 4	0.760454
Adenovirus	0.760454
Chlamydophila pneumoniae	0.760454
Parainfluenza 3	0.760454
Coronavirus HKU1	0.760454
CoronavirusNL63	0.760454
Parainfluenza 1	0.760454
Bordetella pertussis	0.760454
Parainfluenza 2	0.760454
Metapneumovirus	0.760454
Influenza A, rapid test	0.854713
Influenza B, rapid test	0.854713
Hemoglobin	0.893161
Hematocrit	0.893161
Red blood cell distribution width (RDW)	0.893338
Platelets	0.893338
Mean corpuscular volume (MCV)	0.893338
Eosinophils	0.893338
Mean corpuscular hemoglobin (MCH)	0.893338
Basophils	0.893338
Leukocytes	0.893338
Mean corpuscular hemoglobin concentration (MCHC)	0.893338
Lymphocytes	0.893338
Red blood Cells	0.893338
Monocytes	0.893515
Mean platelet volume	0.893870
Neutrophils	0.909107
Proteina C reactiva mg/dL	0.910347
Creatinine	0.924876

Urea	0.929660
Potassium	0.934266
Sodium	0.934444
Strepto A	0.941176
Aspartate transaminase	0.959957
Alanine transaminase	0.960135
Serum Glucose	0.963147
Total Bilirubin	0.967753
Direct Bilirubin	0.967753
Indirect Bilirubin	0.967753
Gamma-glutamyltransferase	0.972892
Alkaline phosphatase	0.974486
HC03 (venous blood gas analysis)	0.975904
pH (venous blood gas analysis)	0.975904
Total CO2 (venous blood gas analysis)	0.975904
Base excess (venous blood gas analysis)	0.975904
pO2 (venous blood gas analysis)	0.975904
pCO2 (venous blood gas analysis)	0.975904
Hb saturation (venous blood gas analysis)	0.975904
International normalized ratio (INR)	0.976435
Creatine phosphokinase (CPK)	0.981573
Lactic Dehydrogenase	0.982105
Myeloblasts	0.982814
Myelocytes	0.982814
Metamyelocytes	0.982814
Promyelocytes	0.982814
Rods #	0.982814
Segmented	0.982814
Relationship (Patient/Normal)	0.983877
Urine - Crystals	0.987597
Urine - Color	0.987597
Urine - Yeasts	0.987597
Urine - Red blood cells	0.987597
Urine - Leukocytes	0.987597
Urine - Density	0.987597
Urine - Bile pigments	0.987597
Urine - Hemoglobin	0.987597
Urine - pH	0.987597
Urine - Aspect	0.987597
Urine - Urobilinogen	0.987775
Urine - Granular cylinders	0.987775
Urine - Hyaline cylinders	0.988129
Urine - Protein	0.989369
Urine - Esterase	0.989369
Urine - Ketone Bodies	0.989901
Ionized calcium	0.991141
Magnesium	0.992913
ctO2 (arterial blood gas analysis)	0.995216

```
Hb saturation (arterial blood gases)           0.995216
pH (arterial blood gas analysis)             0.995216
Arterial Lactic Acid                         0.995216
Total CO2 (arterial blood gas analysis)       0.995216
pCO2 (arterial blood gas analysis)            0.995216
HC03 (arterial blood gas analysis)            0.995216
pO2 (arterial blood gas analysis)              0.995216
Base excess (arterial blood gas analysis)      0.995216
Ferritin                                         0.995925
Arteiral Fio2                                  0.996456
Phosphor                                         0.996456
Albumin                                          0.997697
Lipase dosage                                    0.998583
Vitamin B12                                      0.999468
Urine - Nitrite                                   0.999823
Fio2 (venous blood gas analysis)                0.999823
Partial thromboplastin time (PTT)               1.000000
Urine - Sugar                                     1.000000
Mycoplasma pneumoniae                           1.000000
D-Dimer                                           1.000000
Prothrombin time (PT), Activity                 1.000000
dtype: float64
```

### 5.1.3 Analyse du Fond

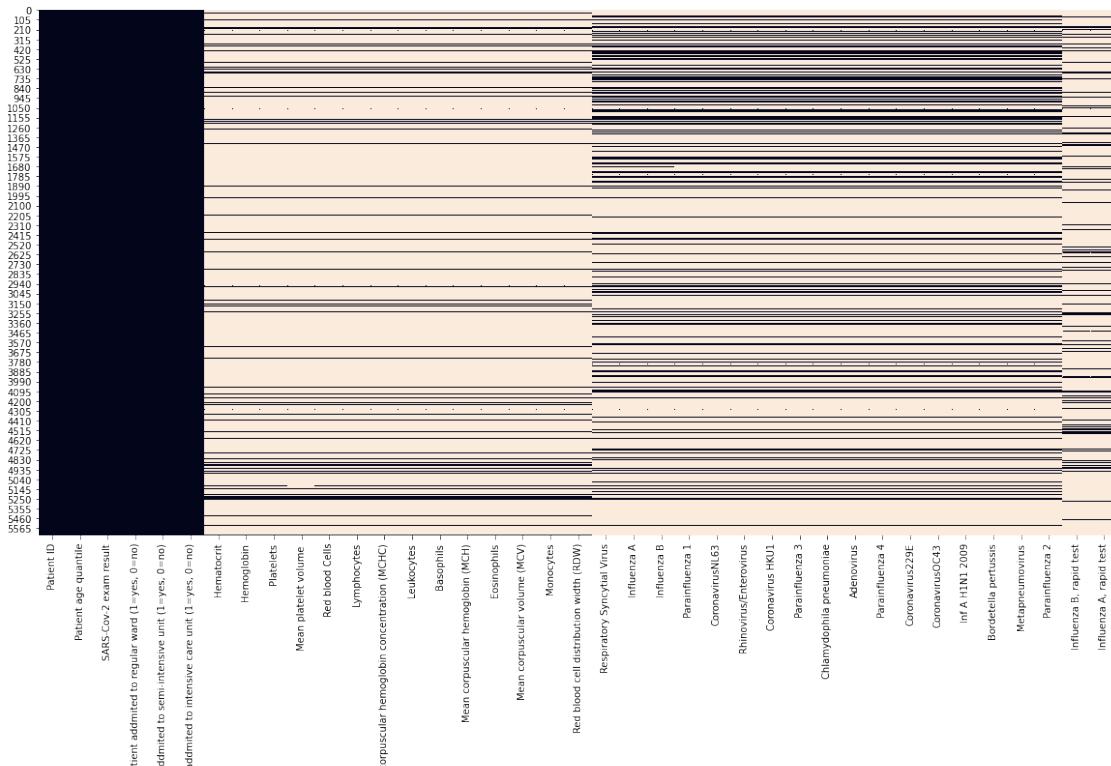
- **Visualisation de la target :**
  - 10% de positifs (558 / 5000)
- **Signification des variables :**
  - variables continues standardisées, skewed (asymétriques), test sanguin
  - age quantile : difficile d'interpréter ce graphique, clairement ces données ont été traitées, on pourrait penser 0-5, mais cela pourrait aussi être une transformation mathématique. On peut pas savoir car la personne qui a mis ce dataset ne le précise nul part. Mais ça n'est pas très important
  - variable qualitative : binaire (0, 1), viral, Rhinovirus qui semble très élevée

```
[26]: df = df[df.columns[df.isna().sum()/df.shape[0] <0.9]]
print('Pourcentage des valeurs Null: ', df.isna().sum().sum()/(df.shape[0]*df.
    .shape[1]))
df.head()
```

Pourcentage des valeurs Null: 0.6959784840720348

```
[27]: plt.figure(figsize=(20,10))
sns.heatmap(df.isna(), cbar=False)
```

```
[27]: <AxesSubplot:>
```

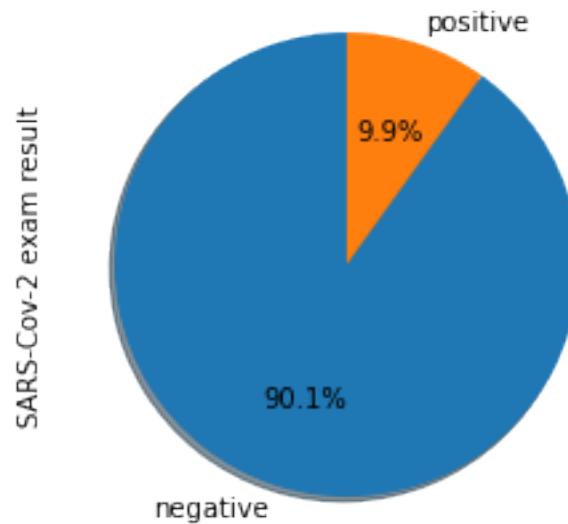


```
[28]: df = df.drop('Patient ID', axis=1)
```

### Examen de la colonne target

```
[36]: print(df['SARS-Cov-2 exam result'].value_counts(normalize=True))
ax=df['SARS-Cov-2 exam result'].value_counts(normalize=True).plot.
    →pie(shadow=True, autopct='%.1.1f%%', startangle=90)
```

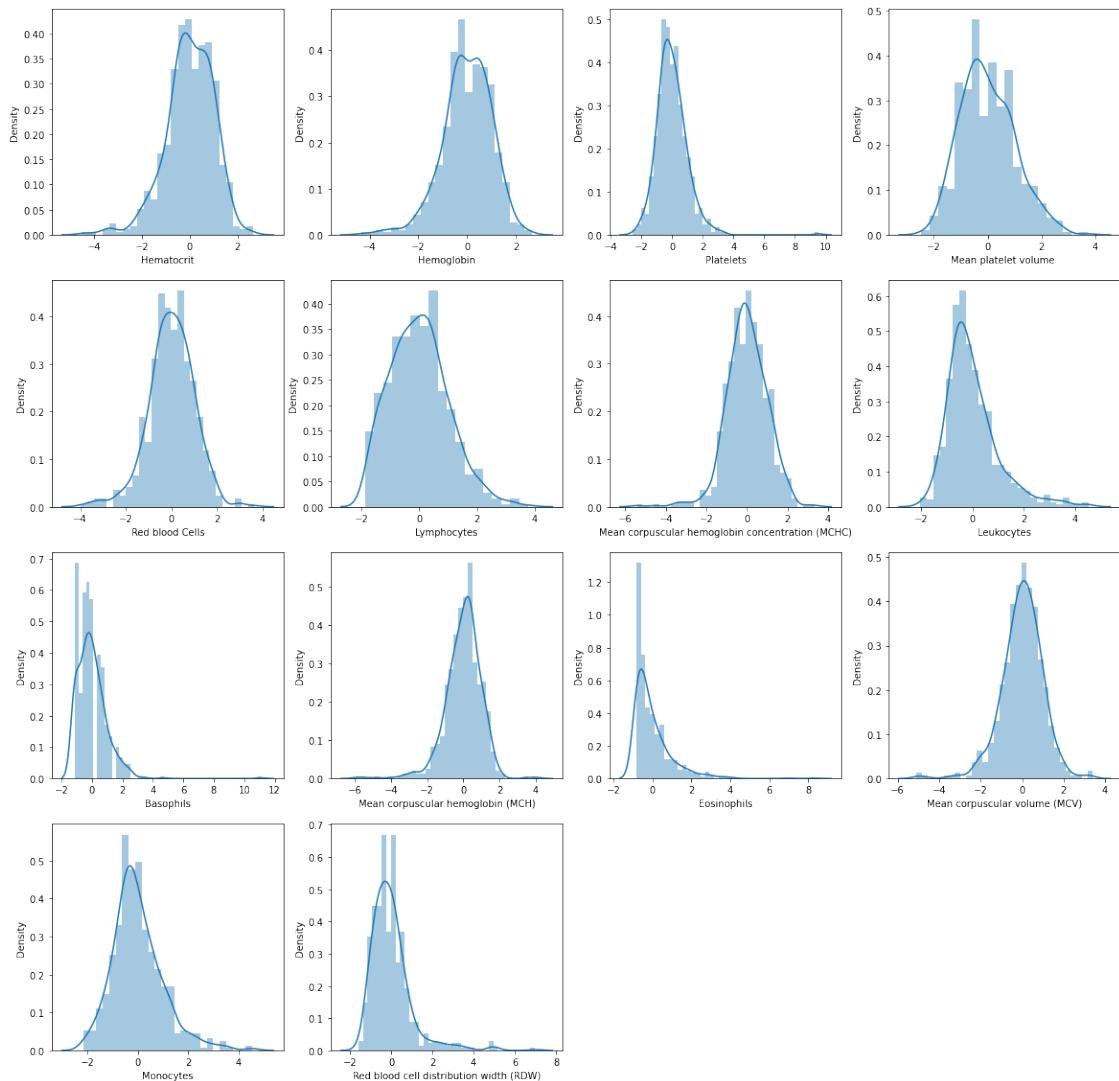
```
negative      0.901134
positive      0.098866
Name: SARS-Cov-2 exam result, dtype: float64
```



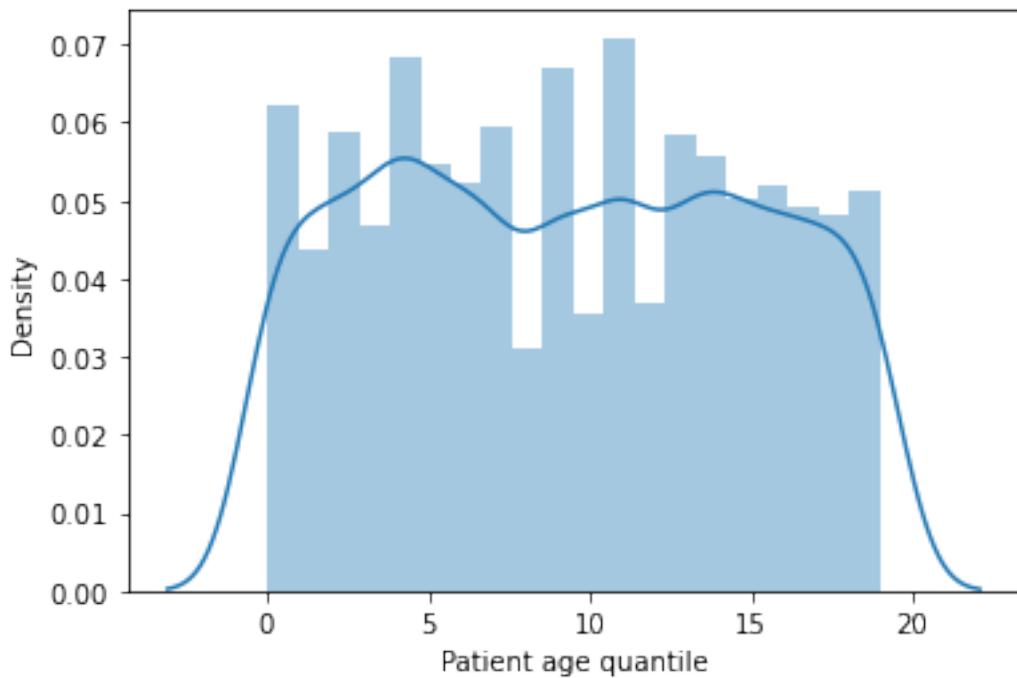
### histogrammes des variables continues

```
[60]: import math
def number_of_line(N,Col):
    return math.ceil(N/Col)
```

```
[45]: Col=4
N_lines = number_of_line(df.select_dtypes('float').shape[1],Col)
f = plt.figure(figsize=(20,20))
for i, col in zip(range(1,df.select_dtypes('float').shape[1]+1),df.
    ↪select_dtypes('float')):
    f.add_subplot(N_lines, Col, i)
    sns.distplot(df[col])
```



```
[46]: sns.distplot(df['Patient age quantile'], bins=20)
```



```
[47]: df['Patient age quantile'].value_counts()
```

```
[47]: 11    380
      4     366
      9     359
      0     334
      7     319
      2     315
      13    313
      14    299
      5     294
      6     281
      16    279
      19    275
      15    269
      17    263
      18    259
      3     251
      1     234
      12   197
      10   190
      8    167
Name: Patient age quantile, dtype: int64
```

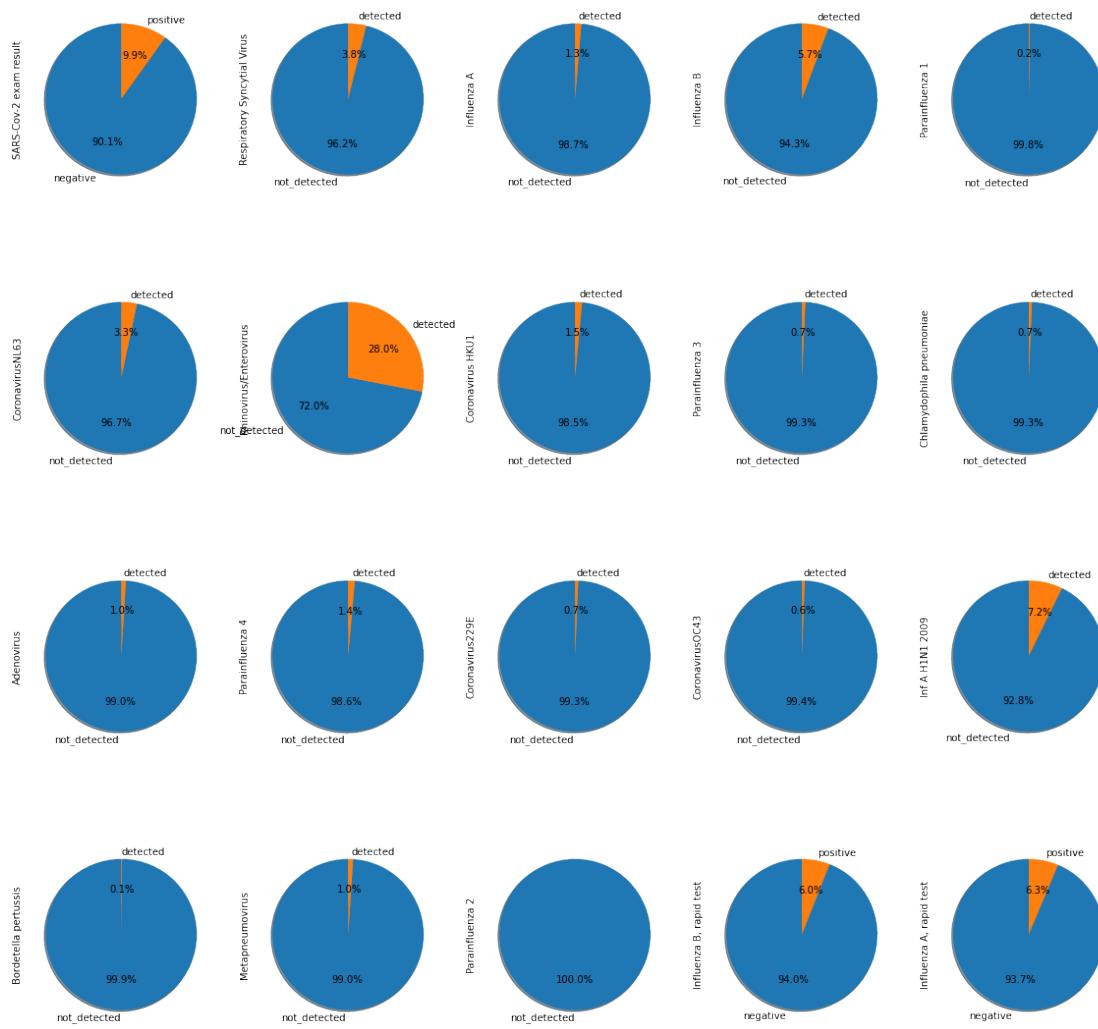
## Variables Qualitatives

```
[48]: for col in df.select_dtypes('object'):
    print(f'{col} : {df[col].unique()}')
```

```
SARS-Cov-2 exam result----- ['negative' 'positive']
Respiratory Syncytial Virus----- [nan 'not_detected'
'detected']
Influenza A----- [nan 'not_detected'
'detected']
Influenza B----- [nan 'not_detected'
'detected']
Parainfluenza 1----- [nan 'not_detected'
'detected']
CoronavirusNL63----- [nan 'not_detected'
'detected']
Rhinovirus/Enterovirus----- [nan 'detected'
'not_detected']
Coronavirus HKU1----- [nan 'not_detected'
'detected']
Parainfluenza 3----- [nan 'not_detected'
'detected']
Chlamydophila pneumoniae----- [nan 'not_detected'
'detected']
Adenovirus----- [nan 'not_detected'
'detected']
Parainfluenza 4----- [nan 'not_detected'
'detected']
Coronavirus229E----- [nan 'not_detected'
'detected']
CoronavirusOC43----- [nan 'not_detected'
'detected']
Inf A H1N1 2009----- [nan 'not_detected'
'detected']
Bordetella pertussis----- [nan 'not_detected'
'detected']
Metapneumovirus----- [nan 'not_detected'
'detected']
Parainfluenza 2----- [nan 'not_detected']
Influenza B, rapid test----- [nan 'negative' 'positive']
Influenza A, rapid test----- [nan 'negative' 'positive']
```

```
[63]: Col=5
n= df.select_dtypes('object').shape[1]
N_lines = number_of_line(n,Col)
f = plt.figure(figsize=(20,20))
for i, col in zip(range(1,n+1),df.select_dtypes('object')):
```

```
f.add_subplot(N_lines, Col, i)
df[col].value_counts().plot.pie(shadow=True, autopct='%.1f%%', startangle=90)
```



### 5.1.4 Relation Target / Variables

- **Relation Variables / Target :**

- target / blood : les taux de Monocytes, Platelets, Leukocytes semblent liés au covid-19  
-> hypothese a tester
- target/age : les individus de faible age sont tres peu contaminés ? -> attention on ne connaît pas l'age, et on ne sait pas de quand date le dataset (s'il s'agit des enfants on sait que les enfants sont touchés autant que les adultes). En revanche cette variable pourra être intéressante pour la comparer avec les résultats de tests sanguins
- target / viral : les doubles maladies sont tres rares. Rhinovirus/Enterovirus positif - covid-19 négatif ? -> hypothese a tester ? mais il est possible que la région est subie une épidémie de ce virus. De plus on peut tres bien avoir 2 virus en meme temps. Tout ca

n'a aucun lien avec le covid-19

### Création de sous-ensembles positifs et négatifs

```
[65]: positive_df = df[df['SARS-Cov-2 exam result'] == 'positive']
```

```
[66]: negative_df = df[df['SARS-Cov-2 exam result'] == 'negative']
```

### Création des ensembles Blood et viral

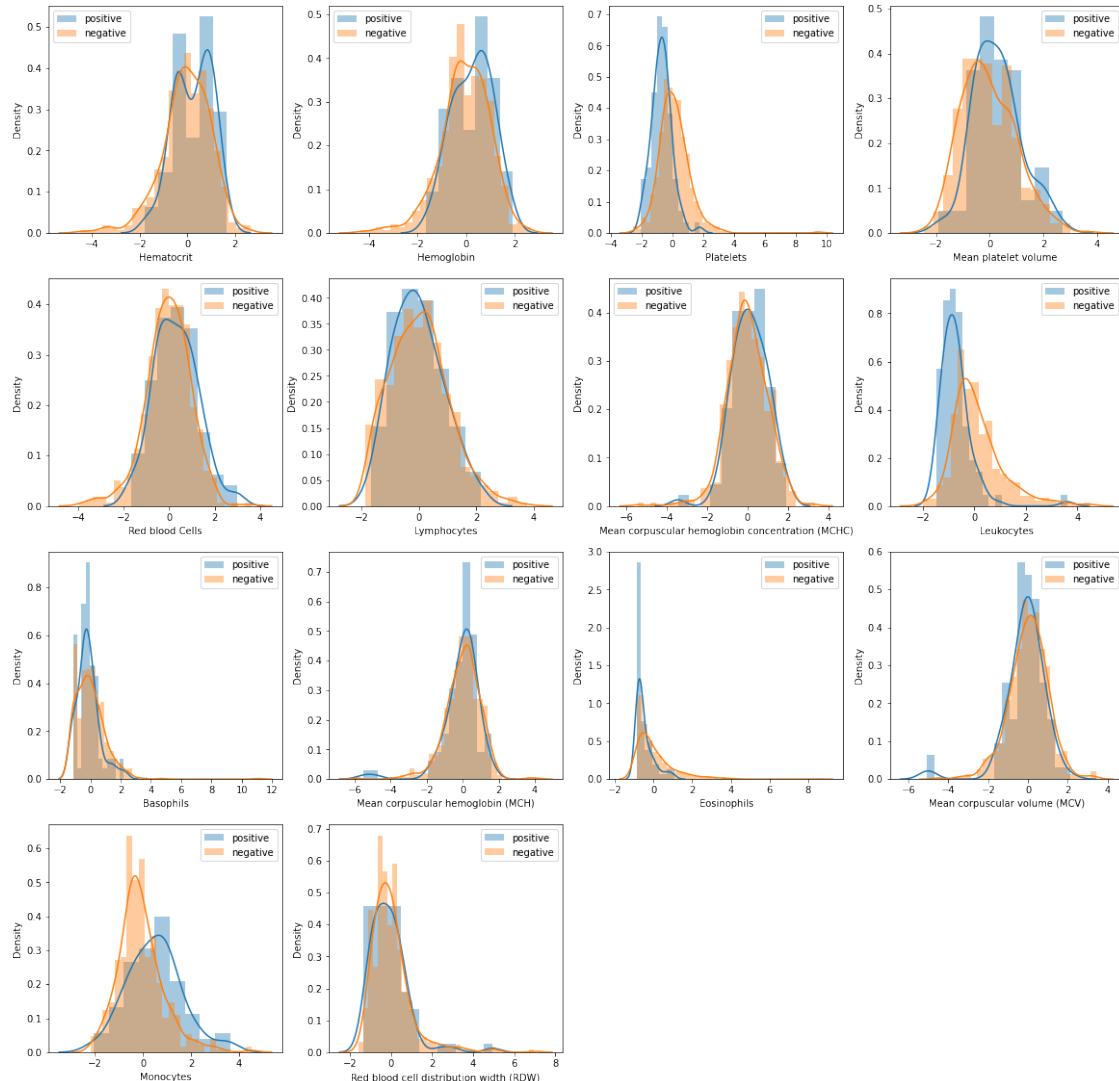
```
[64]: missing_rate = df.isna().sum()/df.shape[0]
```

```
[67]: blood_columns = df.columns[(missing_rate < 0.9) & (missing_rate > 0.88)]
```

```
[68]: viral_columns = df.columns[(missing_rate < 0.88) & (missing_rate > 0.75)]
```

### Target / Blood

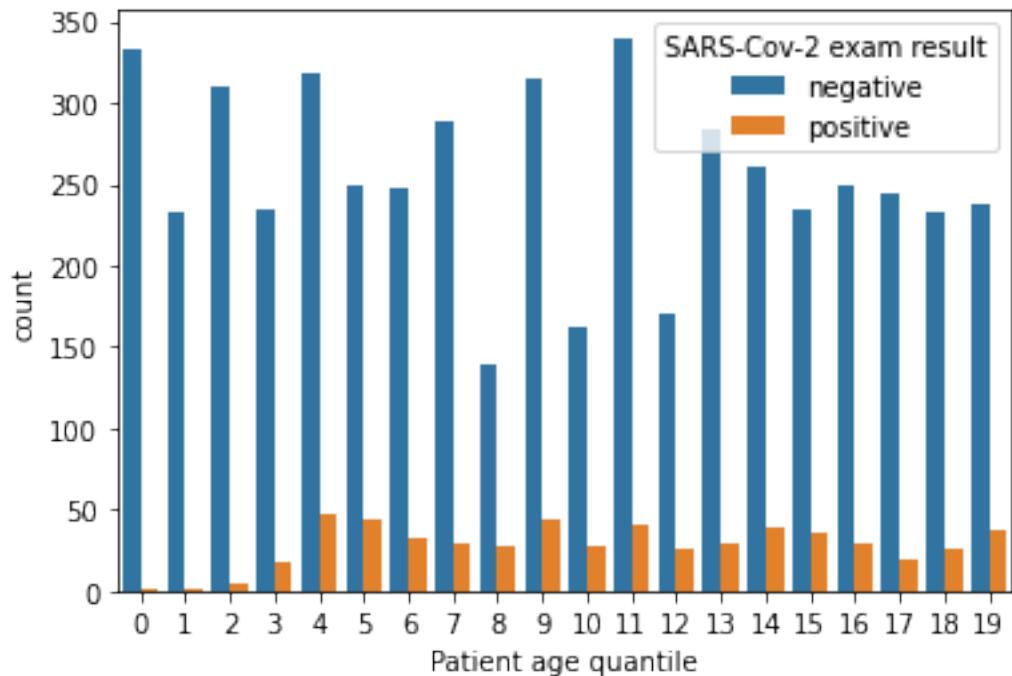
```
[72]: Col=4
n= blood_columns.shape[0]
N_lines = number_of_line(n,Col)
f = plt.figure(figsize=(20,20))
for i, col in zip(range(1,n+1),blood_columns):
    f.add_subplot(N_lines, Col, i)
    sns.distplot(positive_df[col], label='positive')
    sns.distplot(negative_df[col], label='negative')
    plt.legend()
```



### Relation Target / age

```
[50]: sns.countplot(x='Patient age quantile', hue='SARS-CoV-2 exam result', data=df)
```

```
[50]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1cfb18ccc0>
```



## Relation Target / Viral

```
[51]: pd.crosstab(df['SARS-Cov-2 exam result'], df['Influenza A'])
```

```
[51]: Influenza A           detected  not_detected
      SARS-Cov-2 exam result
      negative                  18       1224
      positive                   0        112
```

```
[73]: Col=4
n= viral_columns.shape[0]
N_lines = number_of_line(n,Col)
f = plt.figure(figsize=(20,20))
for i, col in zip(range(1,n+1),viral_columns):
    f.add_subplot(N_lines, Col, i)
    sns.heatmap(pd.crosstab(df['SARS-Cov-2 exam result'], df[col]), annot=True, cbar=False, fmt='d')
```



### 5.1.5 Analyse Avancée

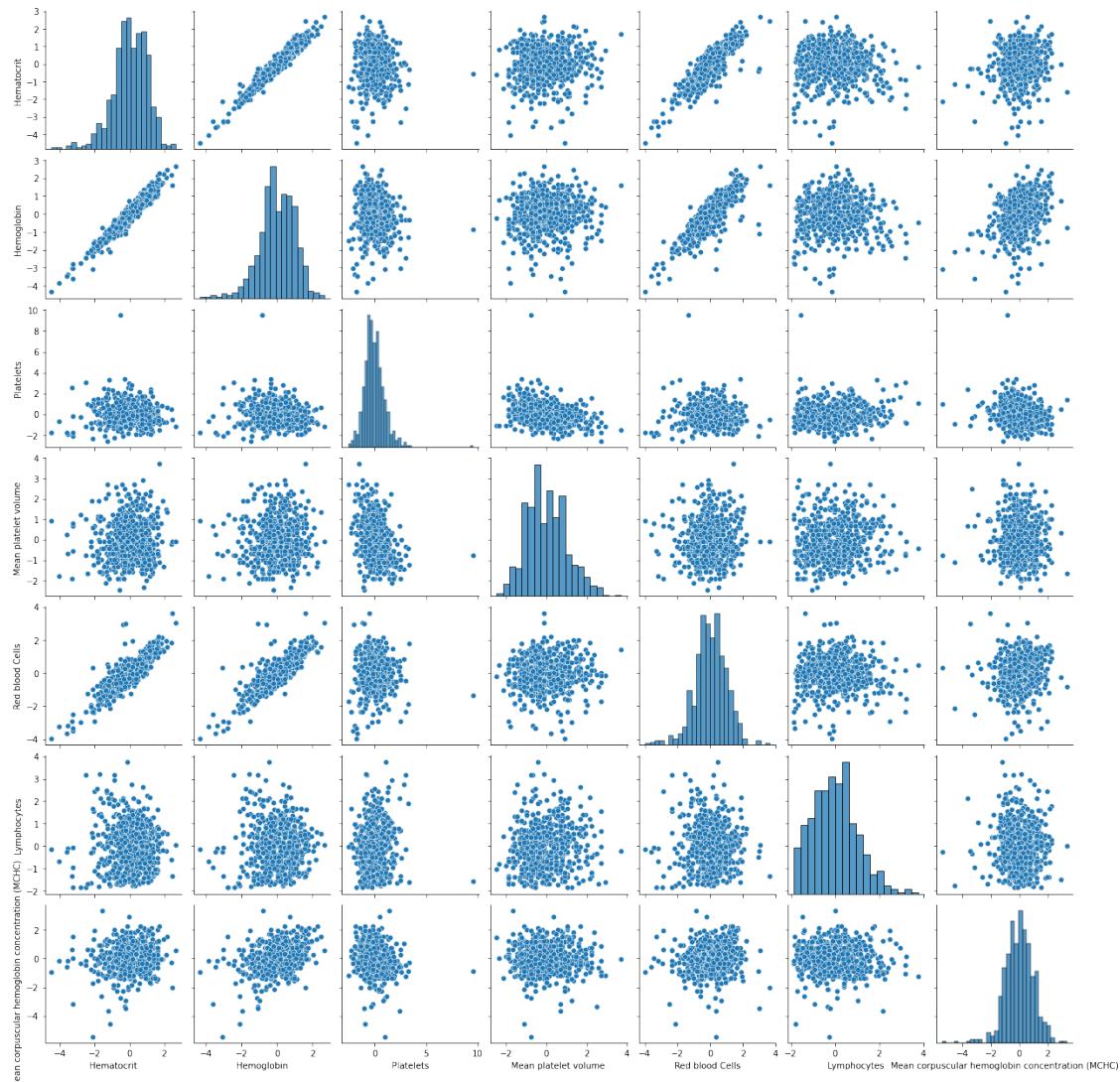
- blood\_data / blood\_data : certaines variables sont très corrélées : +0.9
- blood\_data / age : très faible corrélation entre age et taux sanguins
- viral / viral : influenza rapid test donne de mauvais résultats, il faudrait peut-être la laisser tomber
- relation maladie / blood data : Les taux sanguins entre malades et covid-19 sont différents
- relation hospitalisation / est malade :
- relation hospitalisation / blood : intéressant dans le cas où on voudrait prédire dans quelle service un patient devrait aller
- **NaN analyse** : viral : 1350(92/8), blood : 600(87/13), both : 90

## Relation Variables / Variables

### relations Taux Sanguin

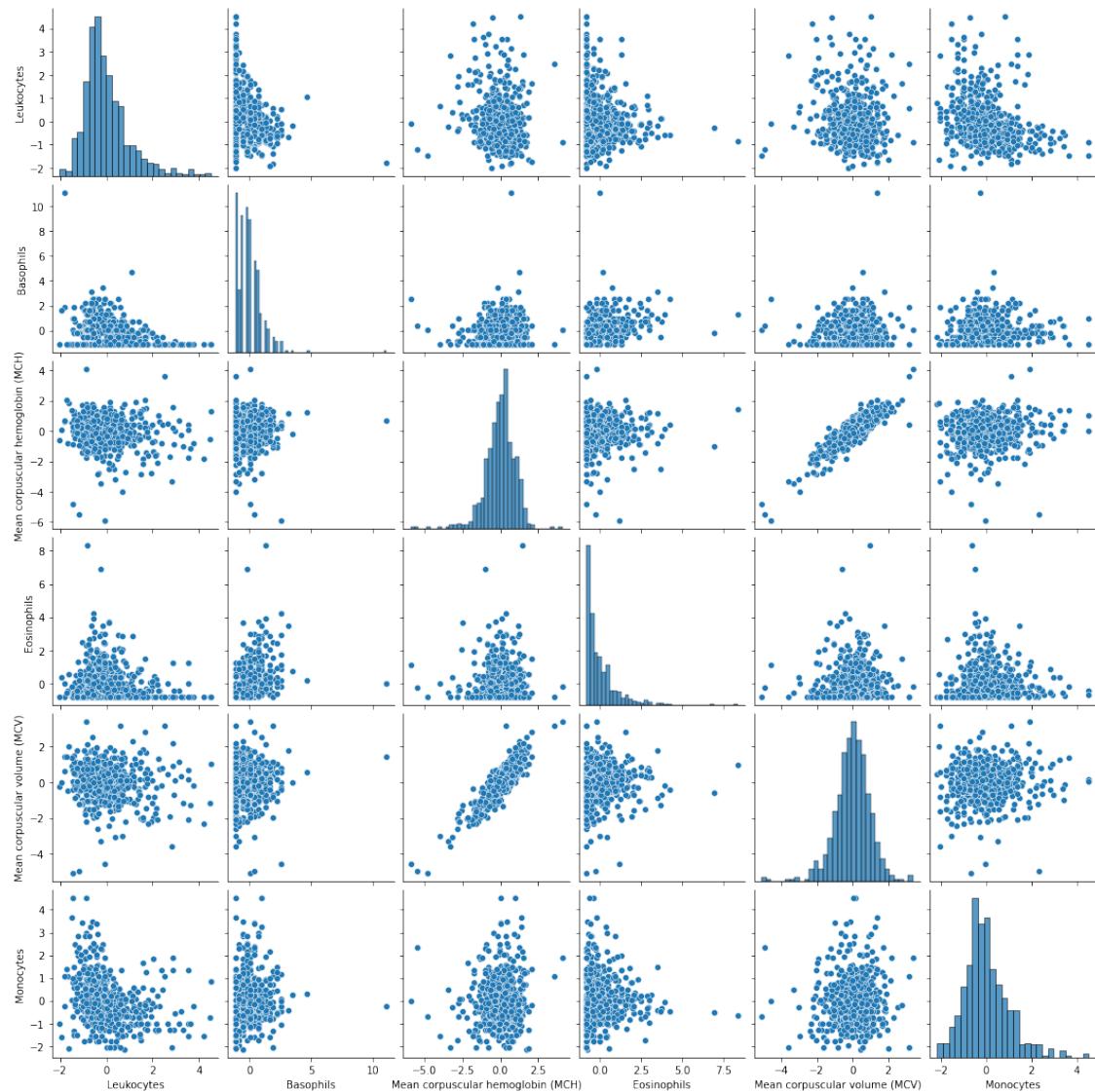
```
[75]: sns.pairplot(df[blood_columns[0:blood_columns.shape[0]//2]])
```

```
[75]: <seaborn.axisgrid.PairGrid at 0x7fe9ffa99100>
```



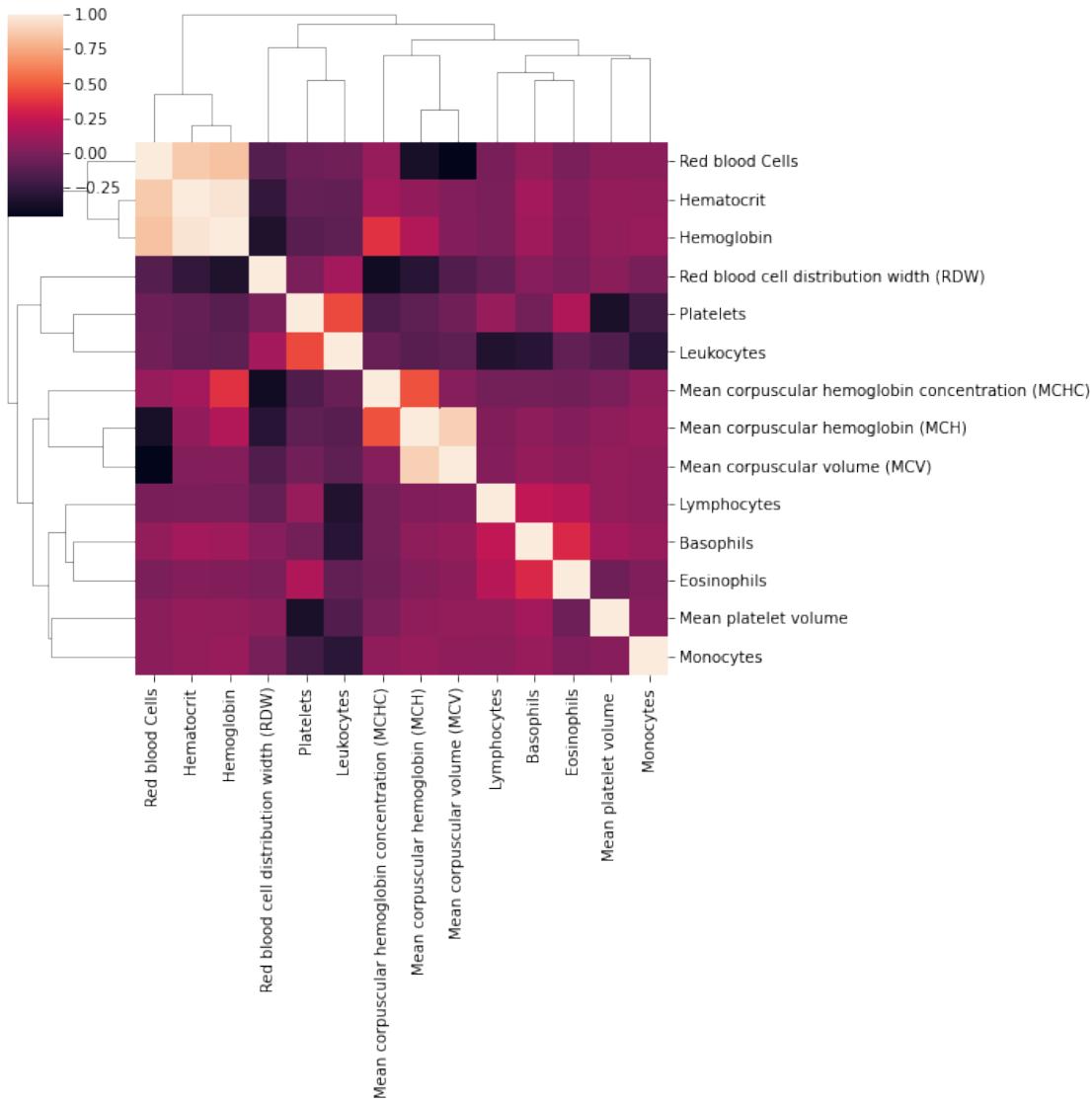
```
[78]: sns.pairplot(df[blood_columns[(blood_columns.shape[0]//2):-1]])
```

```
[78]: <seaborn.axisgrid.PairGrid at 0x7fe9ff2335b0>
```



```
[54]: sns.clustermap(df[blood_columns].corr())
```

```
[54]: <seaborn.matrix.ClusterGrid at 0x7f1cf841eb70>
```



### Relation Age / Blood

```
[56]: df.corr()['Patient age quantile'].sort_values()
```

[56]: Leukocytes	-0.166386
Platelets	-0.158683
Lymphocytes	-0.125935
Mean corpuscular hemoglobin concentration (MCHC)	-0.124671
Red blood Cells	-0.037510
Patient addmited to intensive care unit (1=yes, 0=no)	-0.035772
Patient addmited to semi-intensive unit (1=yes, 0=no)	0.015736
Eosinophils	0.022085

```
Patient addmited to regular ward (1=yes, 0=no)          0.046166
Monocytes                                         0.050962
Hemoglobin                                       0.060320
Hematocrit                                       0.096808
Basophils                                         0.107525
Mean platelet volume                           0.119449
Red blood cell distribution width (RDW)        0.166429
Mean corpuscular hemoglobin (MCH)            0.197394
Mean corpuscular volume (MCV)                 0.281655
Patient age quantile                         1.000000
Name: Patient age quantile, dtype: float64
```

### Relation entre Influenza et rapid test

```
[84]: pd.crosstab(df['Influenza A'], df['Influenza A, rapid test'])
```

```
[84]: Influenza A, rapid test  negative  positive
Influenza A
detected           2          4
not_detected      245         15
```

```
[85]: pd.crosstab(df['Influenza B'], df['Influenza B, rapid test'])
```

```
[85]: Influenza B, rapid test  negative  positive
Influenza B
detected           18          11
not_detected      233          4
```

### relation Viral / sanguin

#### Création d'une nouvelle variable "est malade"

```
[86]: df['est malade'] = np.sum(df[viral_columns[:-2]] == 'detected', axis=1) >=1
```

```
[60]: df.head()
```

```
[60]:             Patient age quantile SARS-CoV-2 exam result \
Patient ID
44477f75e8169d2           13          negative
126e9dd13932f68           17          negative
a46b4402a0e5696             8          negative
f7d619a94f97c45             5          negative
d9e41465789c2b5           15          negative

Patient addmited to regular ward (1=yes, 0=no) \
Patient ID
44477f75e8169d2               0
```

126e9dd13932f68				0
a46b4402a0e5696				0
f7d619a94f97c45				0
d9e41465789c2b5				0
 Patient addmited to semi-intensive unit (1=yes, 0=no) \				
Patient ID				
44477f75e8169d2				0
126e9dd13932f68				0
a46b4402a0e5696				0
f7d619a94f97c45				0
d9e41465789c2b5				0
 Patient addmited to intensive care unit (1=yes, 0=no) \				
Patient ID				
44477f75e8169d2				0
126e9dd13932f68				0
a46b4402a0e5696				0
f7d619a94f97c45				0
d9e41465789c2b5				0
 Hematocrit Hemoglobin Platelets Mean platelet volume \				
Patient ID				
44477f75e8169d2	NaN	NaN	NaN	NaN
126e9dd13932f68	0.236515	-0.02234	-0.517413	0.010677
a46b4402a0e5696	NaN	NaN	NaN	NaN
f7d619a94f97c45	NaN	NaN	NaN	NaN
d9e41465789c2b5	NaN	NaN	NaN	NaN
 Red blood Cells Lymphocytes \				
Patient ID				
44477f75e8169d2		NaN	NaN	
126e9dd13932f68	0.102004	0.318366		
a46b4402a0e5696	NaN	NaN		
f7d619a94f97c45	NaN	NaN		
d9e41465789c2b5	NaN	NaN		
 Mean corpuscular hemoglobin concentration (MCHC) Leukocytes \				
Patient ID				
44477f75e8169d2			NaN	NaN
126e9dd13932f68			-0.95079	-0.09461
a46b4402a0e5696			NaN	NaN
f7d619a94f97c45			NaN	NaN
d9e41465789c2b5			NaN	NaN
 Basophils Mean corpuscular hemoglobin (MCH) Eosinophils \				
Patient ID				

44477f75e8169d2	NaN		NaN	NaN
126e9dd13932f68	-0.223767		-0.292269	1.482158
a46b4402a0e5696	NaN		NaN	NaN
f7d619a94f97c45	NaN		NaN	NaN
d9e41465789c2b5	NaN		NaN	NaN

Mean corpuscular volume (MCV) Monocytes \

Patient ID

44477f75e8169d2		NaN	NaN
126e9dd13932f68		0.166192	0.357547
a46b4402a0e5696		NaN	NaN
f7d619a94f97c45		NaN	NaN
d9e41465789c2b5		NaN	NaN

Red blood cell distribution width (RDW) \

Patient ID

44477f75e8169d2			NaN
126e9dd13932f68			-0.625073
a46b4402a0e5696			NaN
f7d619a94f97c45			NaN
d9e41465789c2b5			NaN

Respiratory Syncytial Virus Influenza A Influenza B \

Patient ID

44477f75e8169d2		NaN	NaN	NaN
126e9dd13932f68		not_detected	not_detected	not_detected
a46b4402a0e5696		NaN	NaN	NaN
f7d619a94f97c45		NaN	NaN	NaN
d9e41465789c2b5		not_detected	not_detected	not_detected

Parainfluenza 1 CoronavirusNL63 Rhinovirus/Enterovirus \

Patient ID

44477f75e8169d2		NaN	NaN	NaN
126e9dd13932f68		not_detected	not_detected	detected
a46b4402a0e5696		NaN	NaN	NaN
f7d619a94f97c45		NaN	NaN	NaN
d9e41465789c2b5		not_detected	not_detected	detected

Coronavirus HKU1 Parainfluenza 3 Chlamydophila pneumoniae \

Patient ID

44477f75e8169d2		NaN	NaN	NaN
126e9dd13932f68		not_detected	not_detected	not_detected
a46b4402a0e5696		NaN	NaN	NaN
f7d619a94f97c45		NaN	NaN	NaN
d9e41465789c2b5		not_detected	not_detected	not_detected

Adenovirus Parainfluenza 4 Coronavirus229E CoronavirusOC43 \

```
Patient ID
44477f75e8169d2      NaN      NaN      NaN      NaN
126e9dd13932f68  not_detected  not_detected  not_detected  not_detected
a46b4402a0e5696      NaN      NaN      NaN      NaN
f7d619a94f97c45      NaN      NaN      NaN      NaN
d9e41465789c2b5  not_detected  not_detected  not_detected  not_detected

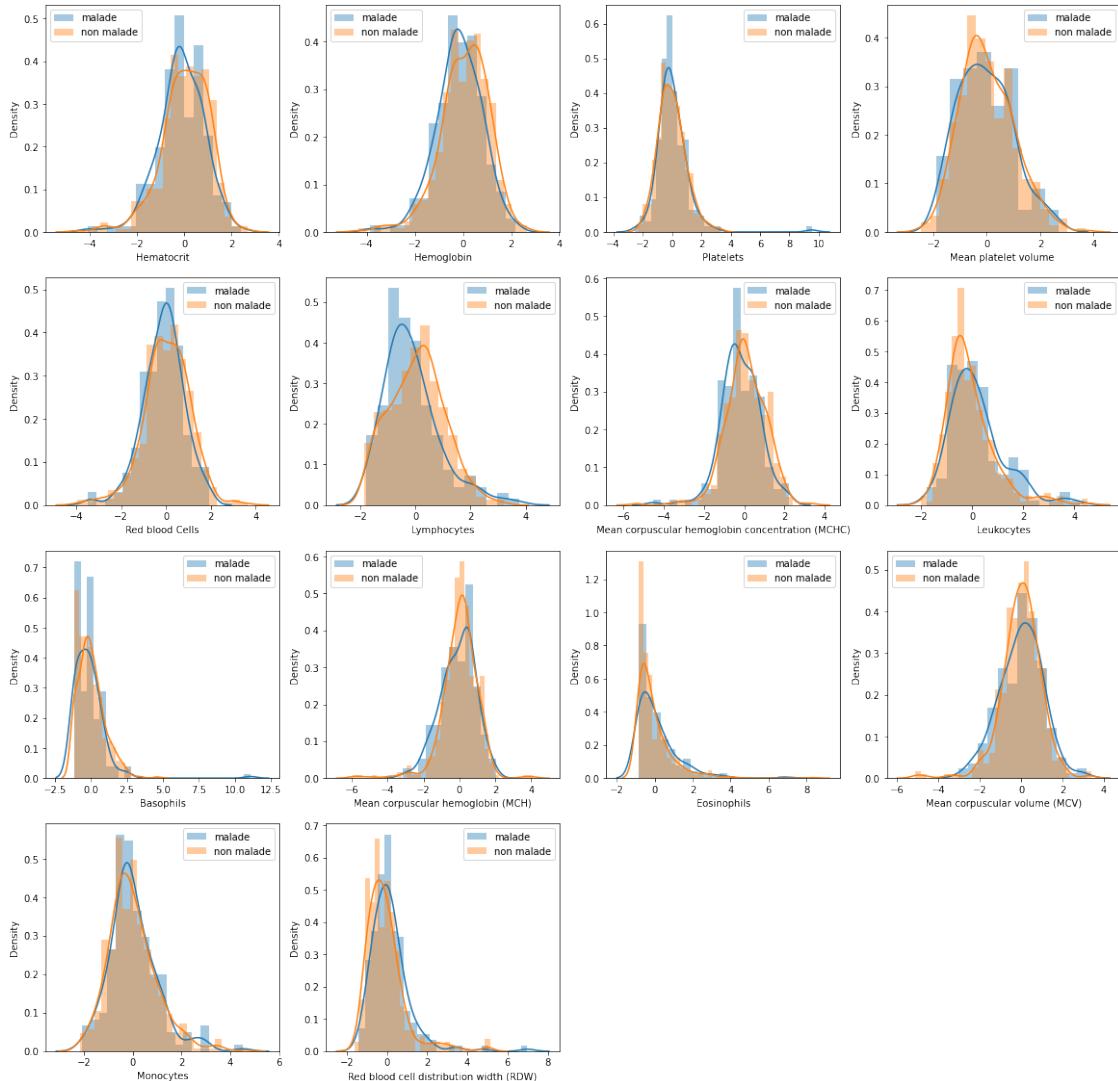
Inf A H1N1 2009 Bordetella pertussis Metapneumovirus \
Patient ID
44477f75e8169d2      NaN      NaN      NaN      NaN
126e9dd13932f68  not_detected  not_detected  not_detected
a46b4402a0e5696      NaN      NaN      NaN      NaN
f7d619a94f97c45      NaN      NaN      NaN      NaN
d9e41465789c2b5  not_detected  not_detected  not_detected

Parainfluenza 2 Influenza B, rapid test \
Patient ID
44477f75e8169d2      NaN      NaN
126e9dd13932f68  not_detected  negative
a46b4402a0e5696      NaN      NaN
f7d619a94f97c45      NaN      NaN
d9e41465789c2b5  not_detected  NaN

Influenza A, rapid test  statut  est malade
Patient ID
44477f75e8169d2      NaN  inconnu  False
126e9dd13932f68  negative  inconnu  True
a46b4402a0e5696      NaN  inconnu  False
f7d619a94f97c45      NaN  inconnu  False
d9e41465789c2b5      NaN  inconnu  True
```

```
[87]: malade_df = df[df['est malade'] == True]
non_malade_df = df[df['est malade'] == False]
```

```
[95]: Col=4
n= blood_columns.shape[0]
N_lines = number_of_line(n,Col)
f = plt.figure(figsize=(20,20))
for i, col in zip(range(1,n+1),blood_columns):
    f.add_subplot(N_lines, Col, i)
    sns.distplot(malade_df[col], label='malade')
    sns.distplot(non_malade_df[col], label='non malade')
    plt.legend()
```



```
[91]: def hospitalisation(df):
    if df['Patient addmited to regular ward (1=yes, 0=no)'] == 1:
        return 'surveillance'
    elif df['Patient addmited to semi-intensive unit (1=yes, 0=no)'] == 1:
        return 'soins semi-intensives'
    elif df['Patient addmited to intensive care unit (1=yes, 0=no)'] == 1:
        return 'soins intensifs'
    else:
        return 'inconnu'
```

```
[92]: df['statut'] = df.apply(hospitalisation, axis=1)
```

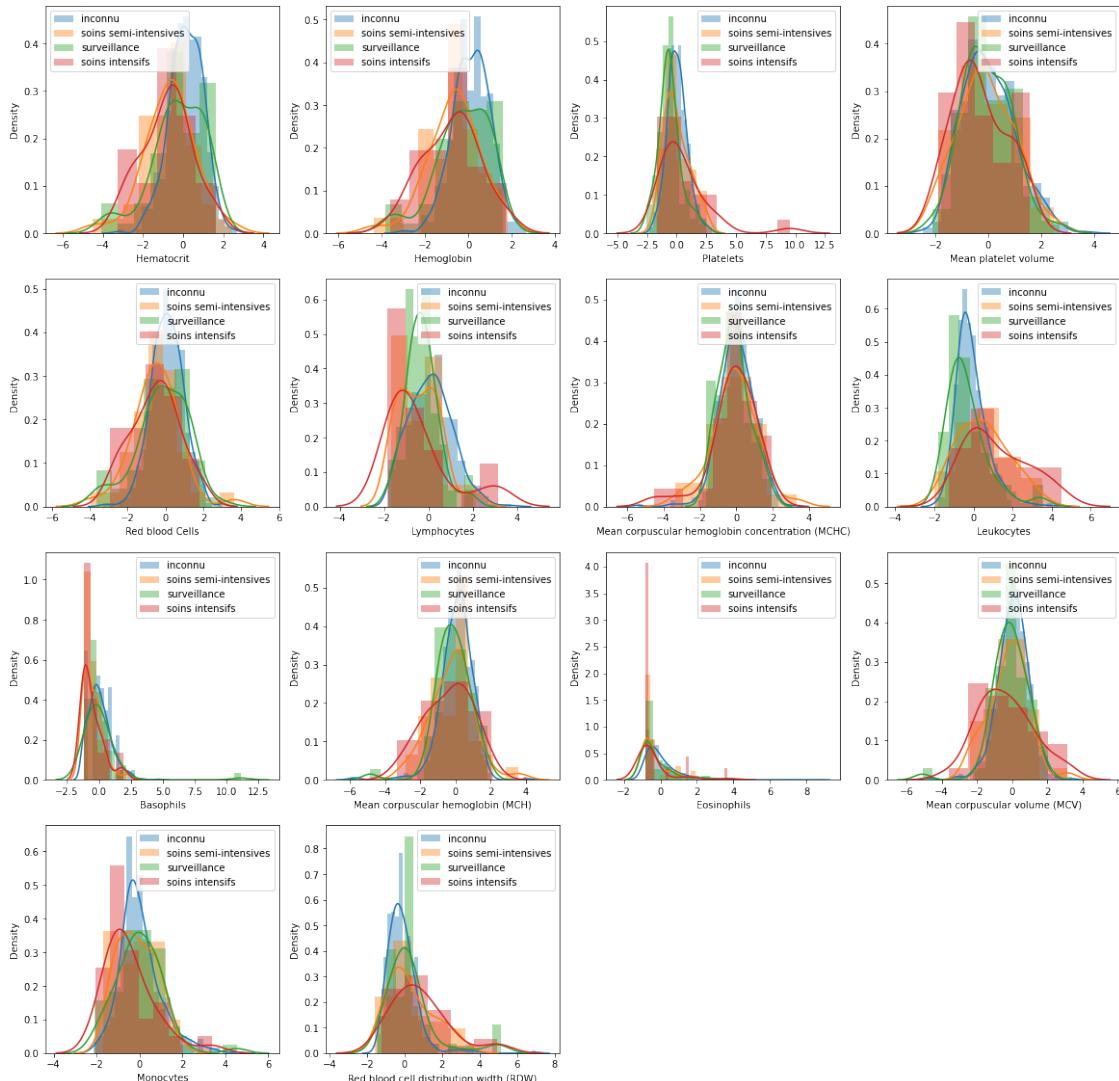
```
[93]: df.head()
```

```
[99]: df['statut'].unique()
```

```
[99]: array(['inconnu', 'soins semi-intensives', 'surveillance',
       'soins intensifs'], dtype=object)
```

```
[96]: Col=4
n= blood_columns.shape[0]
N_lines = number_of_line(n,Col)
f = plt.figure(figsize=(20,20))
for i, col in zip(range(1,n+1),blood_columns):
    f.add_subplot(N_lines, Col, i)

    for cat in df['statut'].unique():
        sns.distplot(df[df['statut']==cat][col], label=cat)
plt.legend()
```



```
[97]: df[blood_columns].count()
```

```
[97]: Hematocrit           603
Hemoglobin            603
Platelets              602
Mean platelet volume  599
Red blood Cells       602
Lymphocytes            602
Mean corpuscular hemoglobin concentration (MCHC) 602
Leukocytes              602
Basophils                602
Mean corpuscular hemoglobin (MCH)      602
Eosinophils             602
Mean corpuscular volume (MCV)        602
Monocytes                601
Red blood cell distribution width (RDW) 602
dtype: int64
```

```
[98]: df[viral_columns].count()
```

```
[98]: Respiratory Syncytial Virus    1354
Influenza A               1354
Influenza B               1354
Parainfluenza 1            1352
CoronavirusNL63            1352
Rhinovirus/Enterovirus     1352
Coronavirus HKU1            1352
Parainfluenza 3            1352
Chlamydophila pneumoniae   1352
Adenovirus                 1352
Parainfluenza 4            1352
Coronavirus229E            1352
CoronavirusOC43            1352
Inf A H1N1 2009            1352
Bordetella pertussis       1352
Metapneumovirus            1352
Parainfluenza 2            1352
Influenza B, rapid test    820
Influenza A, rapid test    820
dtype: int64
```

```
[100]: df1 = df[viral_columns[:-2]]
df1['covid'] = df['SARS-CoV-2 exam result']
df1.dropna()['covid'].value_counts(normalize=True)
```

```
[100]: negative    0.91716
positive     0.08284
```

```
Name: covid, dtype: float64
```

```
[101]: df2 = df[blood_columns]
df2['covid'] = df['SARS-Cov-2 exam result']
df2.dropna()['covid'].value_counts(normalize=True)
```

```
[101]: negative    0.864548
positive     0.135452
Name: covid, dtype: float64
```

### 5.1.6 Test des hypothèses

- Les individus atteints du covid-19 ont des taux de Leukocytes, Monocytes, Platelets significativement différents
  - $H_0$  = Les taux moyens sont ÉGAUX chez les individus positifs et négatifs
- Les individus atteints d'une quelconque maladie ont des taux significativement différents

```
[102]: from scipy.stats import ttest_ind
```

```
[103]: positive_df
```

```
[105]: balanced_neg = negative_df.sample(positive_df.shape[0])

print(positive_df.shape,balanced_neg.shape)
```

```
(558, 38) (558, 38)
```

```
[110]: def t_test(col):
    alpha = 0.02
    stat, p = ttest_ind(balanced_neg[col].dropna(), positive_df[col].dropna())
    if p < alpha:
        return 'H0 Rejetée'
    else :
        return 0
```

- Les individus atteints du covid-19 ont des taux de Leukocytes, Monocytes, Platelets, Hémotocrit, Hemoglobin, Eosinophils significativement différents
  - $H_0$  = Les taux moyens sont ÉGAUX chez les individus positifs et négatifs ( $H_0$  Rejetée)
- Les personnes atteintes de n'importe quelle maladie ont des taux significativement différents
  - $H_0$  = Les niveaux moyens sont ÉGAUX chez les individus malades et non malades ( $H_0$  accepté)

```
[111]: for col in blood_columns:
    print(f'{col} : {t_test(col)}')
```

```
Hematocrit----- H0 Rejetée
Hemoglobin----- H0 Rejetée
Platelets----- H0 Rejetée
Mean platelet volume ----- 0
Red blood Cells----- H0 Rejetée
Lymphocytes----- 0
Mean corpuscular hemoglobin concentration (MCHC)--- 0
Leukocytes----- H0 Rejetée
Basophils----- 0
Mean corpuscular hemoglobin (MCH)----- 0
Eosinophils----- H0 Rejetée
Mean corpuscular volume (MCV)----- 0
Monocytes----- H0 Rejetée
Red blood cell distribution width (RDW)----- 0
```

[112]: `print(malade_df.shape ,non_malade_df.shape )`

```
(692, 39) (4952, 39)
```

[113]: `balanced_malad = non_malade_df.sample(malade_df.shape[0])  
print(balanced_malad.shape, malade_df.shape)`

```
(692, 39) (692, 39)
```

[114]: `def t_test(col):  
 alpha = 0.02  
 stat, p = ttest_ind(balanced_malad[col].dropna(), malade_df[col].dropna())  
 if p < alpha:  
 return 'H0 Rejetée'  
 else :  
 return 0`

[115]: `for col in blood_columns:  
 print(f'{col} : {t_test(col)}')`

```
Hematocrit----- 0
Hemoglobin----- 0
Platelets----- 0
Mean platelet volume ----- 0
Red blood Cells----- 0
Lymphocytes----- 0
Mean corpuscular hemoglobin concentration (MCHC)--- H0 Rejetée
Leukocytes----- 0
Basophils----- 0
Mean corpuscular hemoglobin (MCH)----- 0
Eosinophils----- 0
Mean corpuscular volume (MCV)----- 0
Monocytes----- 0
```

```
Red blood cell distribution width (RDW)----- 0
```

[1]:

```
%%javascript
IPython.OutputArea.auto_scroll_threshold = 9999
```

```
<IPython.core.display.Javascript object>
```

[2]:

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

[5]:

```
data = pd.read_excel('Datasets/dataset.xlsx')
df = data.copy()
```

## 5.2 PRE-PROCESSING

- Création du Train set / Test set
- Élimination des NaN : dropna(), imputation
- Encodage
- Suppression des valeurs aberrantes
- Sélection de caractéristique (Feature Selection)
- Ingénierie des caractéristique (Feature Engineering)
- Mise à l'échelle des caractéristique (Feature Scaling)

### 5.2.1 Crédation des sous-ensembles

[27]:

```
missing_rate = df.isna().sum()/df.shape[0]
```

[28]:

```
blood_columns = list(df.columns[(missing_rate < 0.9) & (missing_rate > 0.88)])
viral_columns = list(df.columns[(missing_rate < 0.80) & (missing_rate > 0.75)])
```

[29]:

```
key_columns = ['Patient age quantile', 'SARS-CoV-2 exam result']
```

[30]:

```
df = df[key_columns + blood_columns + viral_columns]
```

### 5.2.2 Data split - Nettoyage - Encodage

[31]:

```
from sklearn.model_selection import train_test_split
```

[32]:

```
trainset, testset = train_test_split(df, test_size=0.2, random_state=0)
```

[33]:

```
trainset['SARS-CoV-2 exam result'].value_counts()
```

[33]:

negative	4068
positive	447

```
Name: SARS-Cov-2 exam result, dtype: int64
```

```
[34]: testset['SARS-Cov-2 exam result'].value_counts()
```

```
[34]: negative    1018  
positive      111  
Name: SARS-Cov-2 exam result, dtype: int64
```

```
[35]: def encodage(df):  
    code = {'negative':0,  
            'positive':1,  
            'not_detected':0,  
            'detected':1}  
  
    for col in df.select_dtypes('object').columns:  
        df.loc[:,col] = df[col].map(code)  
  
    return df
```

```
[36]: def feature_engineering(df):  
    df['est malade'] = df[viral_columns].sum(axis=1) >= 1  
    df = df.drop(viral_columns, axis=1)  
    return df
```

```
[37]: def imputation(df):  
    df = df.dropna(axis=0)  
    return df
```

```
[38]: def preprocessing(df):  
    df = encodage(df)  
    df = imputation(df)  
  
    X = df.drop('SARS-Cov-2 exam result', axis=1)  
    y = df['SARS-Cov-2 exam result']  
    print(y.value_counts())  
  
    return X, y
```

```
[39]: X_train, y_train = preprocessing(trainset)
```

```
0      258  
1      38  
Name: SARS-Cov-2 exam result, dtype: int64
```

```
[40]: X_test, y_test = preprocessing(testset)
```

```
0      54
```

```
1      12
Name: SARS-CoV-2 exam result, dtype: int64
```

### 5.2.3 Modellisation

```
[41]: from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import make_pipeline
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.preprocessing import PolynomialFeatures
from sklearn.decomposition import PCA

[42]: model_1 = RandomForestClassifier(random_state=0)

[53]: model_2 = make_pipeline(PolynomialFeatures(2), SelectKBest(f_classif, k=10),
                           RandomForestClassifier(random_state=0))
```

### 5.2.4 Procédure d'évaluation

```
[44]: from sklearn.metrics import f1_score, confusion_matrix, classification_report
from sklearn.model_selection import learning_curve

[45]: def evaluation(model):

    model.fit(X_train, y_train)
    ypred = model.predict(X_test)

    print(confusion_matrix(y_test, ypred))
    print(classification_report(y_test, ypred))

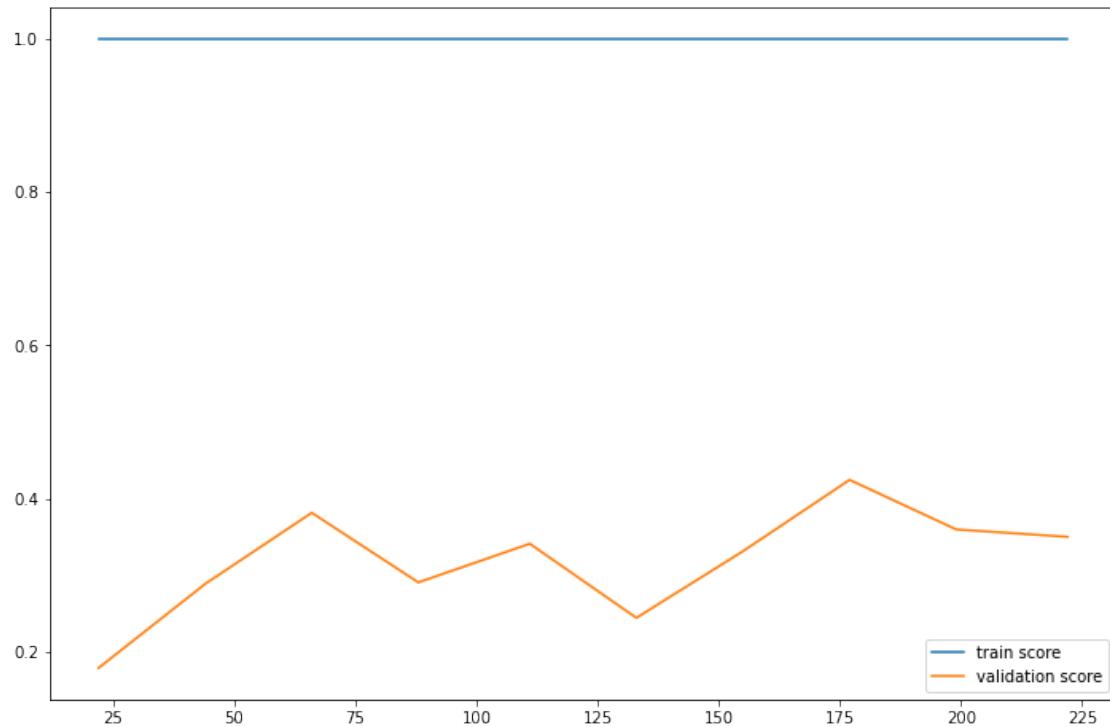
    N, train_score, val_score = learning_curve(model, X_train, y_train,
                                                cv=4, scoring='f1',
                                                train_sizes=np.linspace(0.1, 1, 10))

    plt.figure(figsize=(12, 8))
    plt.plot(N, train_score.mean(axis=1), label='train score')
    plt.plot(N, val_score.mean(axis=1), label='validation score')
    plt.legend()

[46]: evaluation(model_1)
```

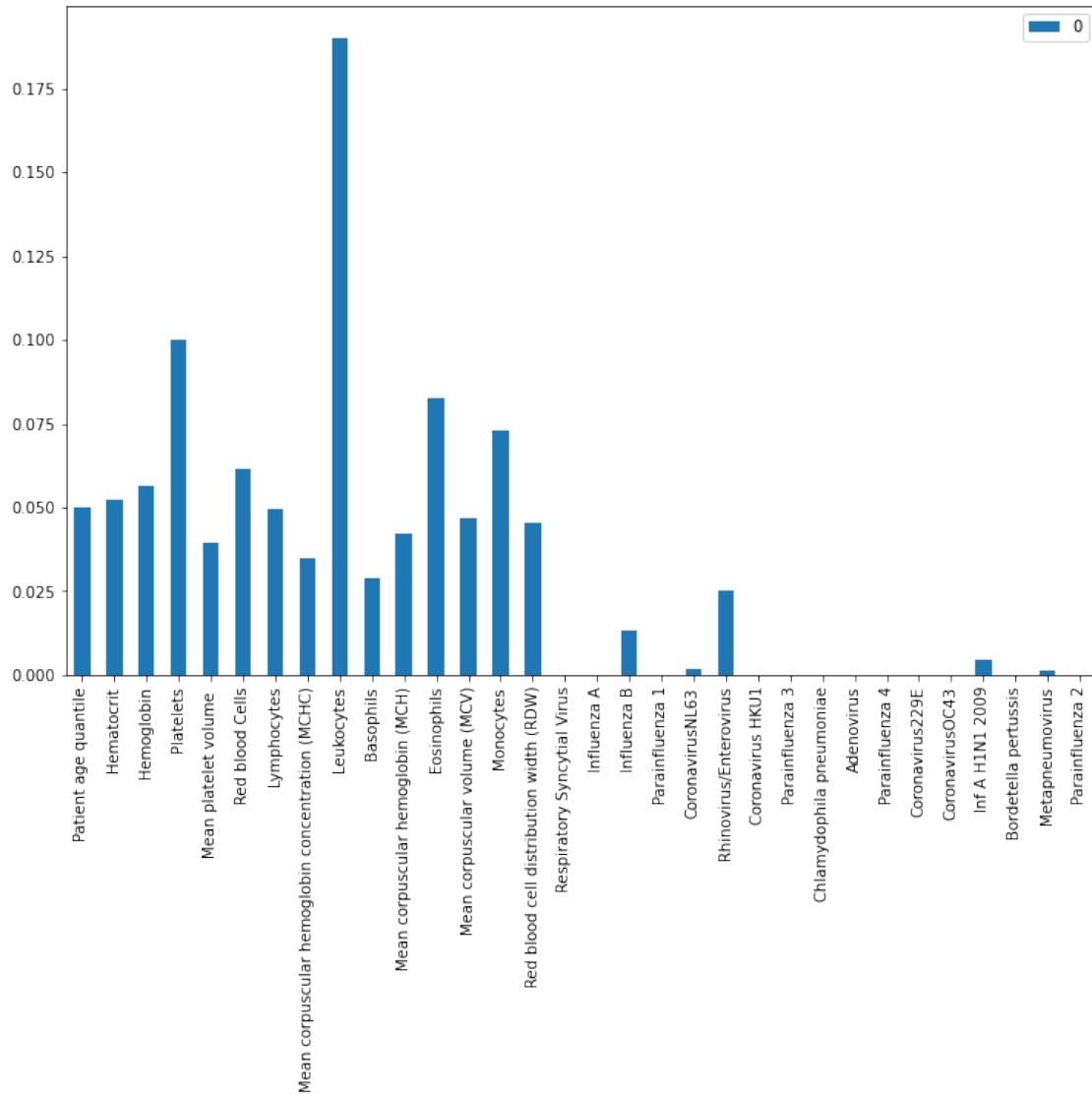
```
[[54  0]
 [11  1]]
```

	precision	recall	f1-score	support
0	0.83	1.00	0.91	54
1	1.00	0.08	0.15	12
accuracy			0.83	66
macro avg	0.92	0.54	0.53	66
weighted avg	0.86	0.83	0.77	66



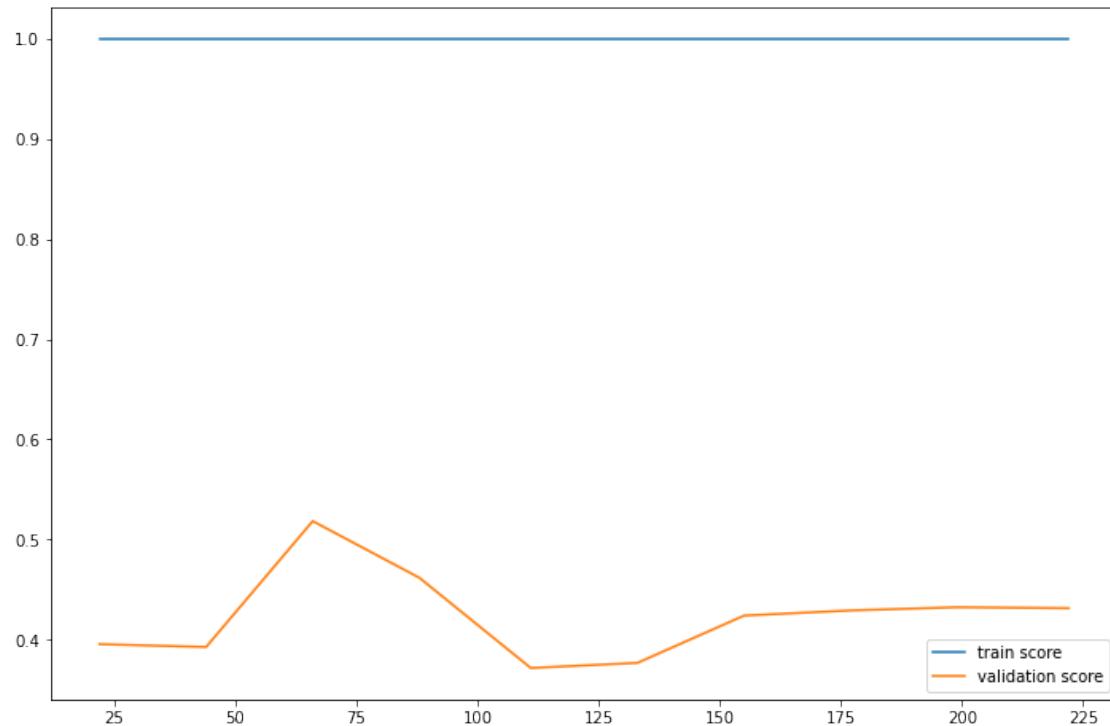
```
[47]: pd.DataFrame(model_1.feature_importances_, index=X_train.columns).plot.  
      ↪bar(figsize=(12, 8))
```

```
[47]: <AxesSubplot:>
```



```
[55]: evaluation(model_2)
```

	precision	recall	f1-score	support
0	0.85	0.94	0.89	54
1	0.50	0.25	0.33	12
accuracy			0.82	66
macro avg	0.68	0.60	0.61	66
weighted avg	0.79	0.82	0.79	66



## 5.3 Modelisation

### 5.3.1 Creation des sous-ensembles

```
[53]: missing_rate = df.isna().sum()/df.shape[0]
```

```
[54]: blood_columns = list(df.columns[(missing_rate < 0.9) & (missing_rate > 0.88)])
viral_columns = list(df.columns[(missing_rate < 0.80) & (missing_rate > 0.75)])
```

```
[8]: key_columns = ['Patient age quantile', 'SARS-CoV-2 exam result']
```

```
[55]: df = df[key_columns + blood_columns + viral_columns]
```

### 5.3.2 TrainTest - Nettoyage - Encodage

```
[9]: from sklearn.model_selection import train_test_split
```

```
[56]: trainset, testset = train_test_split(df, test_size=0.2, random_state=0)
```

```
[57]: trainset['SARS-CoV-2 exam result'].value_counts()
```

```
[57]: negative    4068
      positive     447
      Name: SARS-Cov-2 exam result, dtype: int64
```

```
[58]: testset['SARS-Cov-2 exam result'].value_counts()
```

```
[58]: negative    1018
      positive     111
      Name: SARS-Cov-2 exam result, dtype: int64
```

```
[39]: def encodage(df):
        code = {'negative':0,
                 'positive':1,
                 'not_detected':0,
                 'detected':1}

        for col in df.select_dtypes('object').columns:
            df.loc[:,col] = df[col].map(code)

        return df
```

```
[40]: def feature_engineering(df):
        df['est malade'] = df[viral_columns].sum(axis=1) >= 1
        df = df.drop(viral_columns, axis=1)
        return df
```

```
[41]: def imputation(df):
        #df['is na'] = (df['Parainfluenza 3'].isna()) | (df['Leukocytes'].isna())
        #df = df.fillna(-999)
        df = df.dropna(axis=0)
        return df
```

```
[59]: def preprocessing(df):

    df = encodage(df)
    df = feature_engineering(df)
    df = imputation(df)

    X = df.drop('SARS-Cov-2 exam result', axis=1)
    y = df['SARS-Cov-2 exam result']

    print(y.value_counts())

    return X, y
```

```
[60]: X_train, y_train = preprocessing(trainset)
      print(X_train.shape, y_train.shape)
```

```
0    422
1     65
Name: SARS-CoV-2 exam result, dtype: int64
(487, 16) (487,)
```

```
[61]: X_test, y_test = preprocessing(testset)
print(X_test.shape, y_test.shape)
```

```
0    95
1     16
Name: SARS-CoV-2 exam result, dtype: int64
(111, 16) (111,)
```

### 5.3.3 Procédure d'évaluation

```
[19]: from sklearn.metrics import f1_score, confusion_matrix, classification_report
from sklearn.model_selection import learning_curve
```

```
[20]: def evaluation(model):

    model.fit(X_train, y_train)
    ypred = model.predict(X_test)

    print(confusion_matrix(y_test, ypred))
    print(classification_report(y_test, ypred))

    N, train_score, val_score = learning_curve(model, X_train, y_train,
                                                cv=4, scoring='f1',
                                                train_sizes=np.linspace(0.1, 1, ↴10))

    plt.figure(figsize=(12, 8))
    plt.plot(N, train_score.mean(axis=1), label='train score')
    plt.plot(N, val_score.mean(axis=1), label='validation score')
    plt.legend()
```

### 5.3.4 Modellisation

```
[21]: from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import make_pipeline
from sklearn.feature_selection import SelectKBest, f_classif
```

```
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
```

```
[22]: preprocessor = make_pipeline(PolynomialFeatures(2, include_bias=False),  
    SelectKBest(f_classif, k=10))
```

```
[23]: RandomForest = make_pipeline(preprocessor,  
    RandomForestClassifier(random_state=0))  
AdaBoost = make_pipeline(preprocessor, AdaBoostClassifier(random_state=0))  
SVM = make_pipeline(preprocessor, StandardScaler(), SVC(random_state=0))  
KNN = make_pipeline(preprocessor, StandardScaler(), KNeighborsClassifier())
```

```
[24]: dict_of_models = {'RandomForest': RandomForest,  
                     'AdaBoost' : AdaBoost,  
                     'SVM' : SVM,  
                     'KNN' : KNN  
}
```

```
[62]: for name, model in dict_of_models.items():  
    print(name)  
    evaluation(model)
```

RandomForest

[[91 4]
[11 5]]
precision
recall
f1-score
support

0	0.89	0.96	0.92	95
1	0.56	0.31	0.40	16

accuracy			0.86	111
macro avg	0.72	0.64	0.66	111
weighted avg	0.84	0.86	0.85	111

AdaBoost

[[91 4]
[ 9 7]]
precision
recall
f1-score
support

0	0.91	0.96	0.93	95
1	0.64	0.44	0.52	16

accuracy			0.88	111
macro avg	0.77	0.70	0.73	111
weighted avg	0.87	0.88	0.87	111

SVM

[[92 3]]
----------

[10 6]]

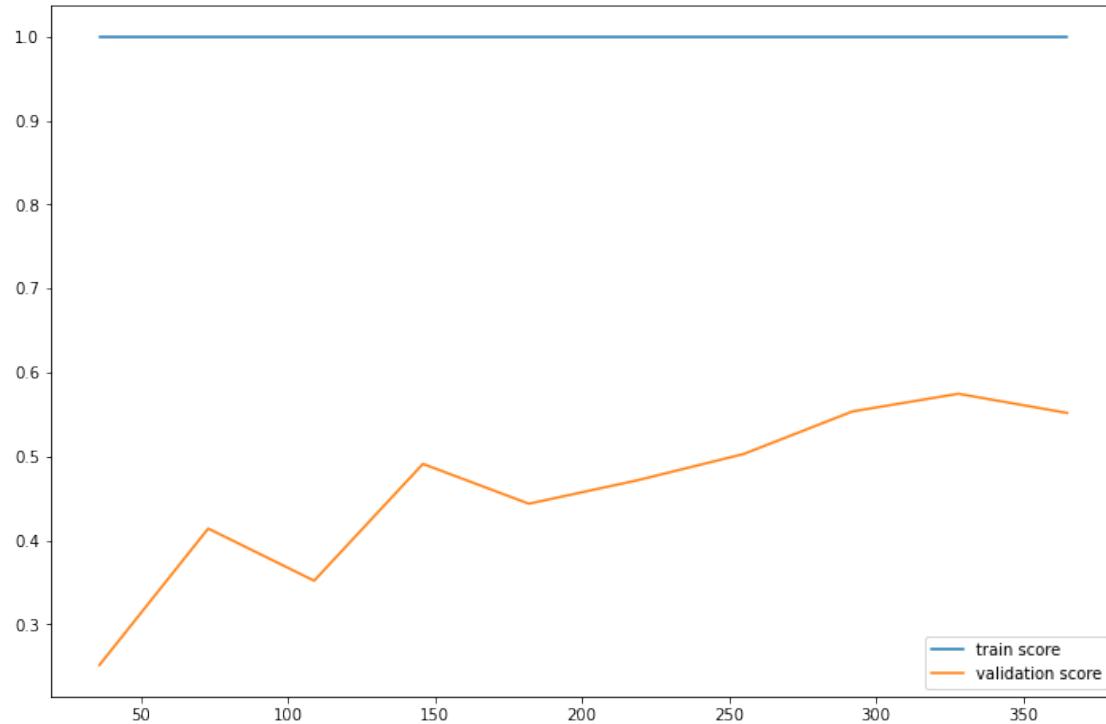
	precision	recall	f1-score	support
0	0.90	0.97	0.93	95
1	0.67	0.38	0.48	16
accuracy			0.88	111
macro avg	0.78	0.67	0.71	111
weighted avg	0.87	0.88	0.87	111

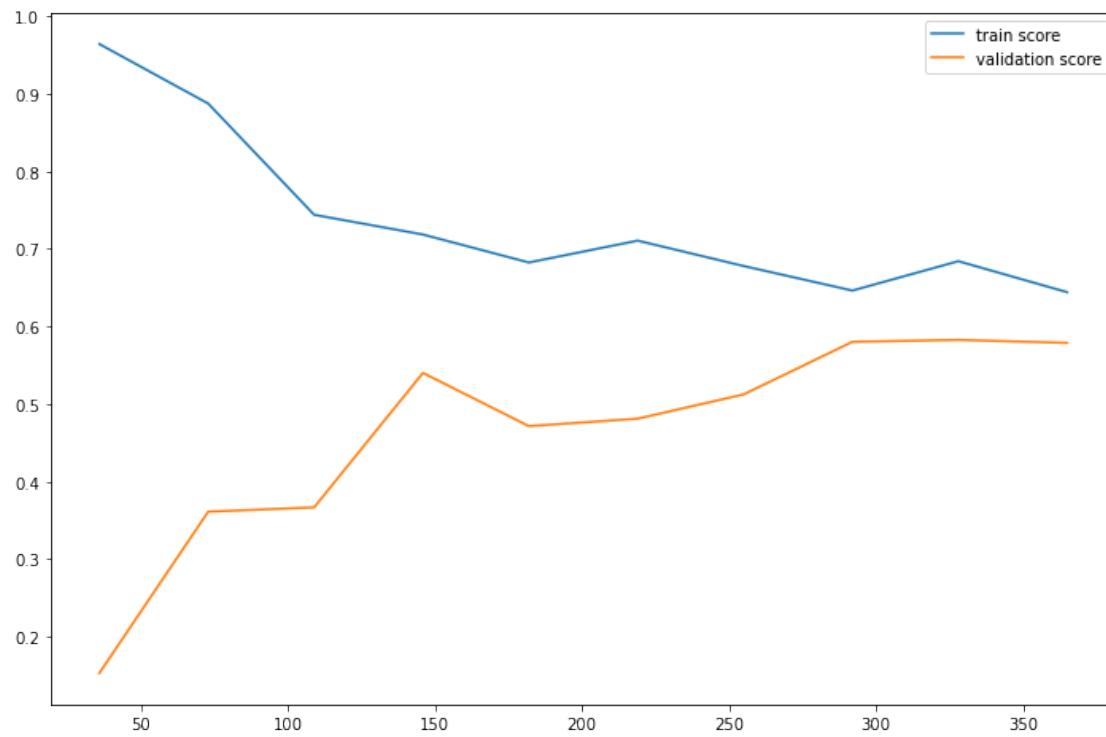
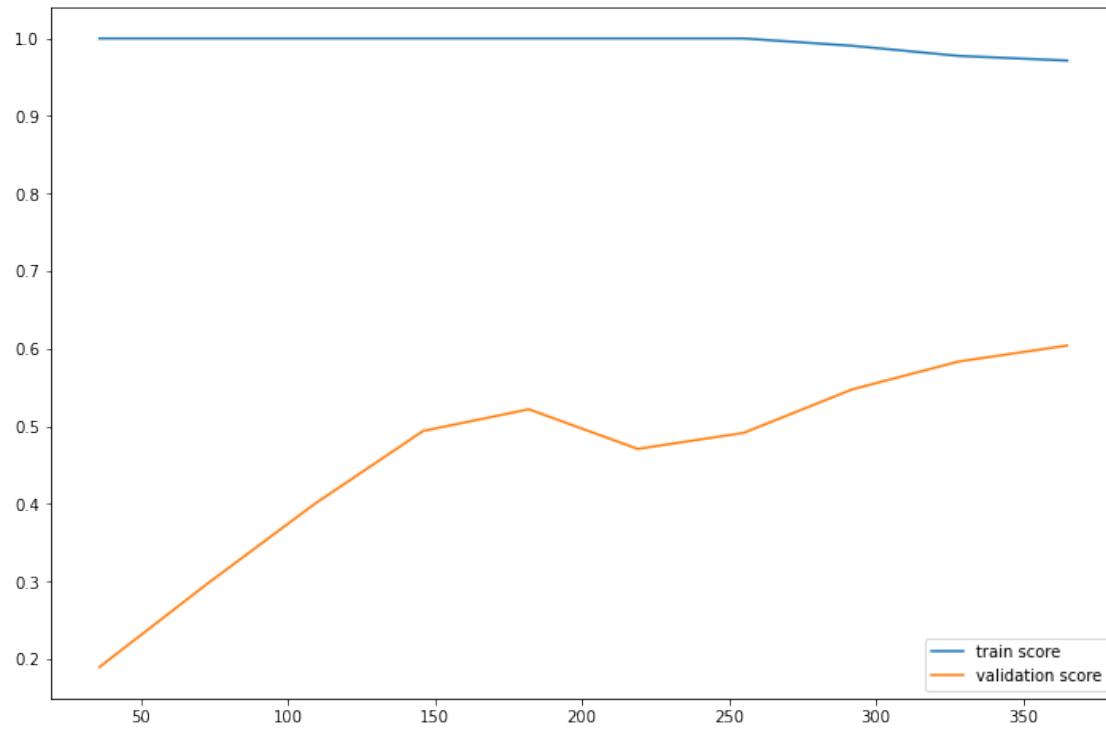
KNN

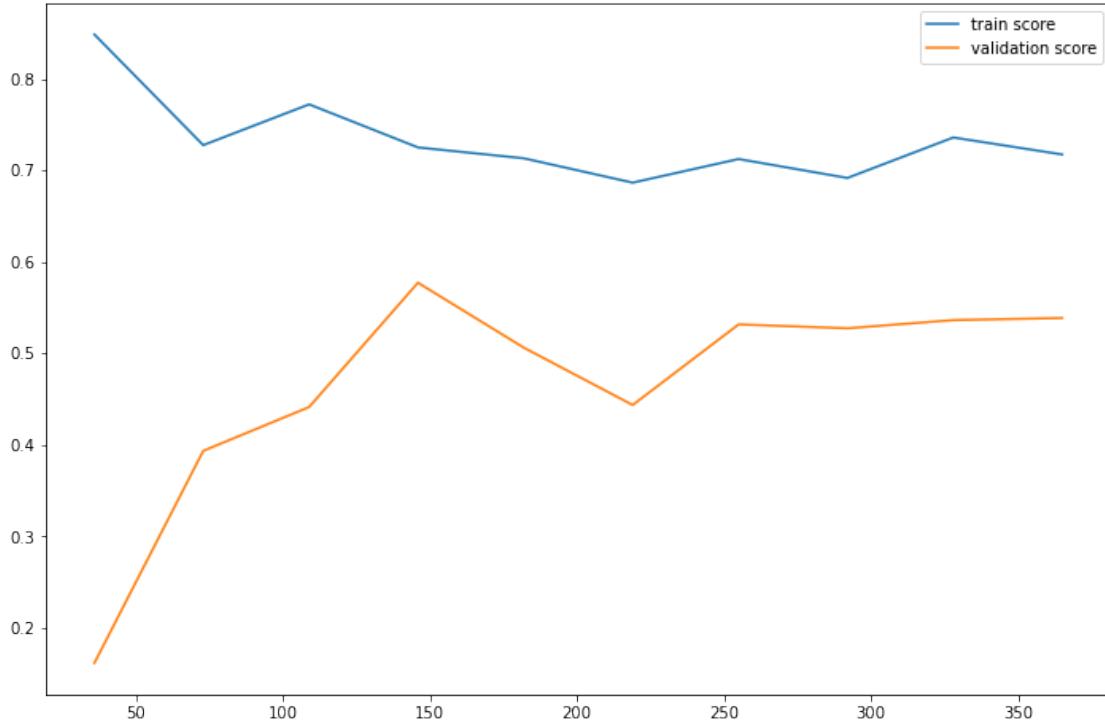
[[88 7]

[ 8 8]]

	precision	recall	f1-score	support
0	0.92	0.93	0.92	95
1	0.53	0.50	0.52	16
accuracy			0.86	111
macro avg	0.72	0.71	0.72	111
weighted avg	0.86	0.86	0.86	111







### 5.3.5 OPTIMISATION

```
[ ]: from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
```

[32] : SVM

```
[32]: Pipeline(memory=None,
              steps=[('pipeline',
                      Pipeline(memory=None,
                               steps=[('polynomialfeatures',
                                       PolynomialFeatures(degree=2,
                                                       include_bias=False,
                                                       interaction_only=False,
                                                       order='C')),
                               ('selectkbest',
                                   SelectKBest(k=10,
                                               score_func=<function f_classif at
0x7f453d76d158>))],
                               verbose=False)),
              ('standardscaler',
                  StandardScaler(copy=True, with_mean=True, with_std=True)),
              ('svc',
                  SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None,
```

```
coef0=0.0, decision_function_shape='ovr', degree=3,
gamma='scale', kernel='rbf', max_iter=-1,
probability=False, random_state=0, shrinking=True,
tol=0.001, verbose=False)],
verbose=False)
```

```
[ ]: hyper_params = {'svc__gamma':[1e-3, 1e-4, 0.0005],
'svc__C':[1, 10, 100, 1000, 3000],
'pipeline__polynomialfeatures__degree':[2, 3],
'pipeline__selectkbest__k': range(45, 60)}
```

```
[34]: grid = RandomizedSearchCV(SVM, hyper_params, scoring='recall', cv=4,
n_iter=40)

grid.fit(X_train, y_train)

print(grid.best_params_)

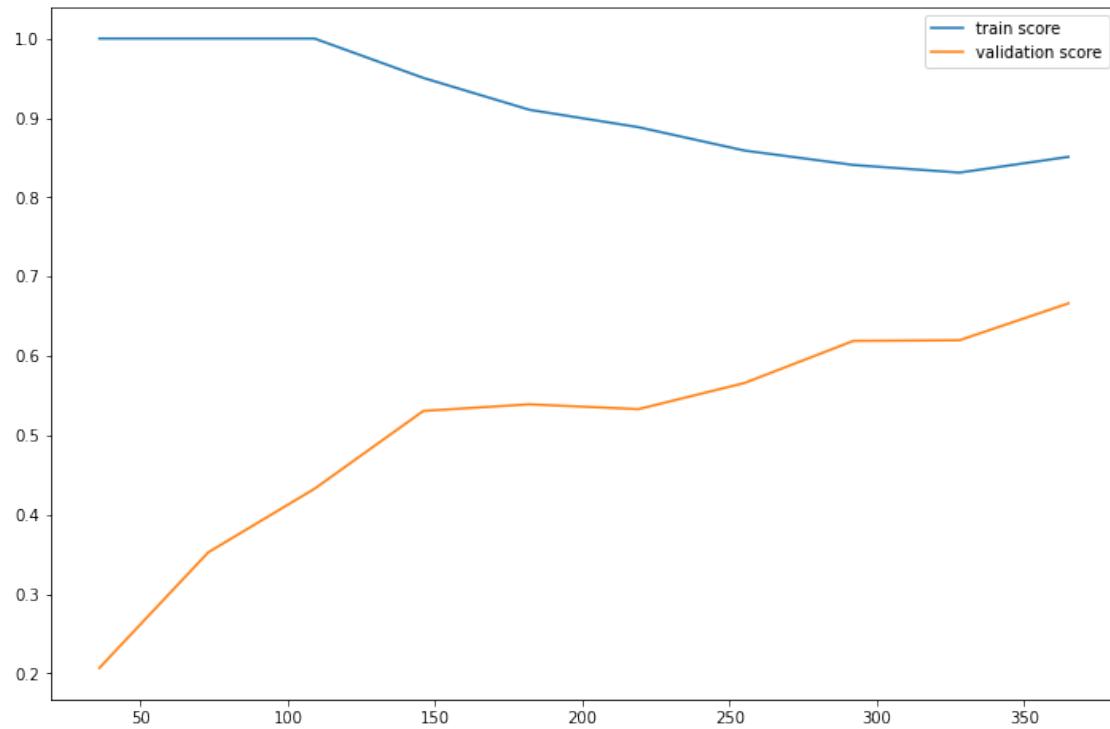
y_pred = grid.predict(X_test)

print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.92	0.95	0.93	95
1	0.62	0.50	0.55	16
accuracy			0.88	111
macro avg	0.77	0.72	0.74	111
weighted avg	0.87	0.88	0.88	111

```
[35]: evaluation(grid.best_estimator_)
```

	precision	recall	f1-score	support
0	0.92	0.95	0.93	95
1	0.62	0.50	0.55	16
accuracy			0.88	111
macro avg	0.77	0.72	0.74	111
weighted avg	0.87	0.88	0.88	111



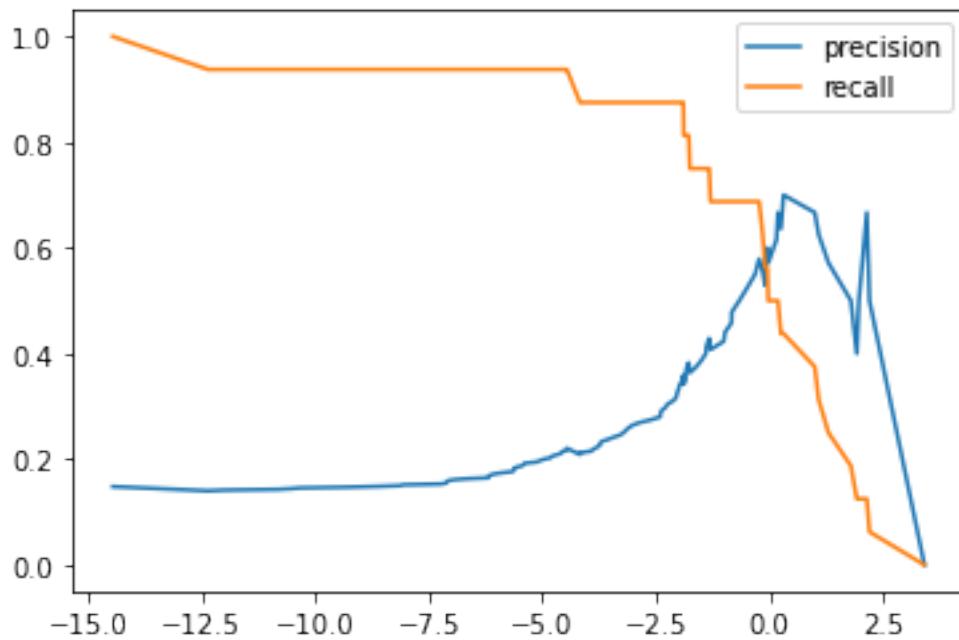
### 5.3.6 Precision Recall Curve

```
[ ]: from sklearn.metrics import precision_recall_curve
```

```
[ ]: precision, recall, threshold = precision_recall_curve(y_test, grid.  
             ↪best_estimator_.decision_function(X_test))
```

```
[38]: plt.plot(threshold, precision[:-1], label='precision')  
plt.plot(threshold, recall[:-1], label='recall')  
plt.legend()
```

```
[38]: <matplotlib.legend.Legend at 0x7f453ce71f28>
```



```
[ ]: def model_final(model, X, threshold=0):
    return model.decision_function(X) > threshold

[ ]: y_pred = model_final(grid.best_estimator_, X_test, threshold=-1)

[ ]: from sklearn.metrics import recall_score

[42]: f1_score(y_test, y_pred)

[42]: 0.5365853658536586

[43]: recall_score(y_test, y_pred)

[43]: 0.6875
```

# Chapter 6

## Projets Global

### 6.1 Description du problème :

Une société automobile chinoise Geely Auto aspire à pénétrer le marché américain en y installant son unité de fabrication et en produisant des voitures localement pour faire concurrence à ses homologues américains et européens.

Ils ont engagé une société de conseil automobile pour comprendre les facteurs dont dépend le prix des voitures. Plus précisément, ils veulent comprendre les facteurs affectant le prix des voitures sur le marché américain, car ceux-ci peuvent être très différents du marché chinois. L'entreprise veut savoir :

- Quelles variables sont significatives pour prédire le prix d'une voiture
- Dans quelle mesure ces variables décrivent bien le prix d'une voiture

### 6.2 Travail à faire

Nous sommes amenés à modéliser le prix des voitures avec les variables indépendantes disponibles. Il sera utilisé par la direction pour comprendre comment exactement les prix varient avec les variables indépendantes. Ils peuvent donc manipuler la conception des voitures, la stratégie commerciale, etc. pour atteindre certains niveaux de prix. De plus, le modèle sera un bon moyen pour la direction de comprendre la dynamique des prix d'un nouveau marché.

```
[ ]: import numpy as np
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

pd.options.display.float_format='{: .4f}'.format
```

## 6.3 Exploratory Data Analysis

### 6.3.1 Analyse de la forme des données

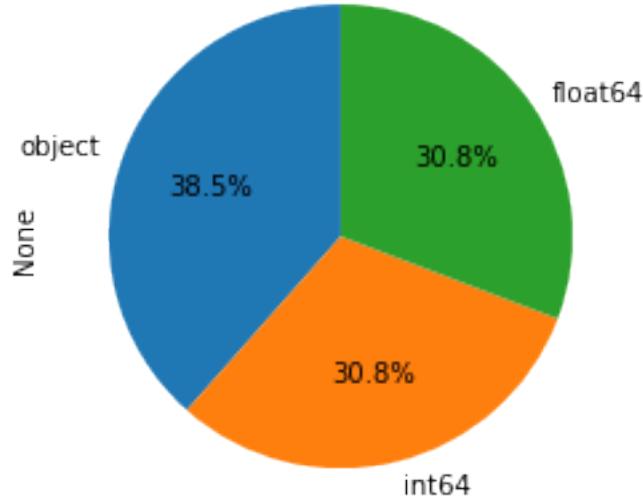
- **variable target** : price
- **lignes et colonnes** : 205, 26
- **types de variables** : qualitatives : 10, quantitatives : 16
- **Analyse des valeurs manquantes** : pas de valeurs NaN dans le dataset

```
[199]: file="CarPrice_Assignment.csv"
df_auto = pd.read_csv(file)
df_auto.head()
df = df_auto.copy()
df.shape
```

[199]: (205, 26)

```
[200]: df.dtypes.value_counts().plot.pie(autopct='%.1f%%', startangle=90)
df.dtypes.value_counts()
```

```
[200]: object      10
       int64       8
       float64     8
       dtype: int64
```



```
[201]: df_auto.info()
```

<class 'pandas.core.frame.DataFrame'>

```
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column            Non-Null Count Dtype  
 --- 
 0   car_ID             205 non-null    int64  
 1   symboling          205 non-null    int64  
 2   CarName            205 non-null    object  
 3   fueltype           205 non-null    object  
 4   aspiration         205 non-null    object  
 5   doornumber         205 non-null    object  
 6   carbbody           205 non-null    object  
 7   drivewheel         205 non-null    object  
 8   enginelocation     205 non-null    object  
 9   wheelbase          205 non-null    float64 
 10  carlength          205 non-null    float64 
 11  carwidth           205 non-null    float64 
 12  carheight          205 non-null    float64 
 13  curbweight         205 non-null    int64  
 14  enginetype         205 non-null    object  
 15  cylindernumber     205 non-null    object  
 16  enginesize          205 non-null    int64  
 17  fuelsystem          205 non-null    object  
 18  boreratio           205 non-null    float64 
 19  stroke              205 non-null    float64 
 20  compressionratio    205 non-null    float64 
 21  horsepower          205 non-null    int64  
 22  peakrpm             205 non-null    int64  
 23  citympg             205 non-null    int64  
 24  highwaympg          205 non-null    int64  
 25  price               205 non-null    float64 

dtypes: float64(8), int64(8), object(10)
memory usage: 41.8+ KB
```

### 6.3.2 Analyse du Fond

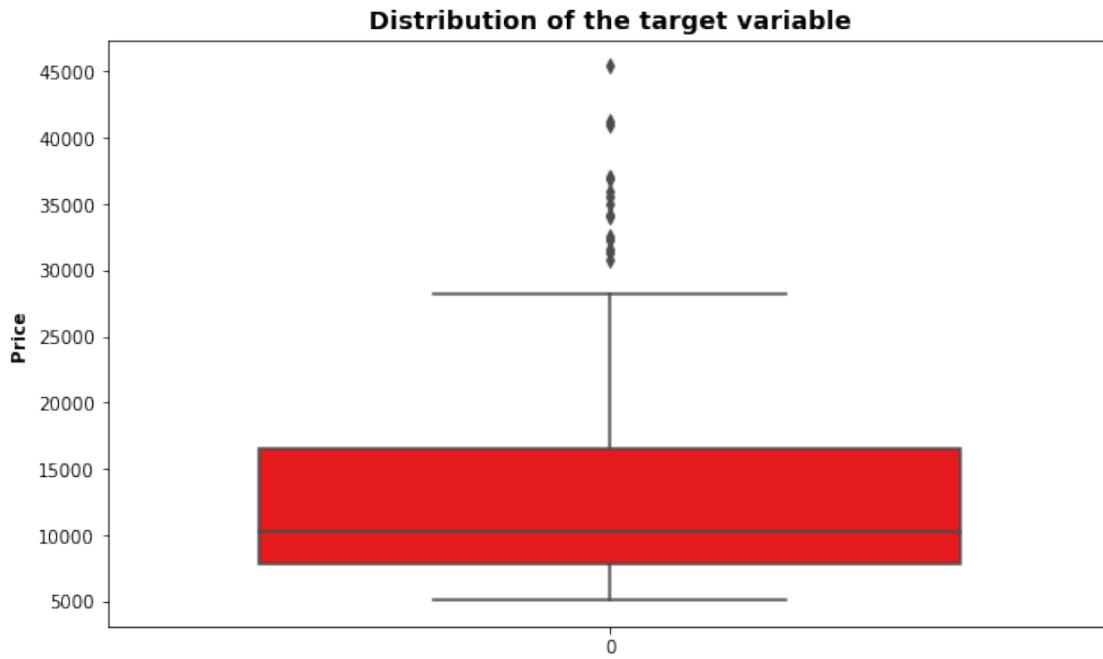
#### Examen de la colonne target

- Visualisation de la target :
  - Certaines fourchettes de prix supérieures à 36 000 peuvent être qualifiées de valeurs aberrantes, mais ne les supprimons pas, nous utiliserons plutôt une mise à l'échelle de normalisation.
  - Les parcelles semblent être asymétriques, les prix de presque toutes les voitures semblent inférieurs à 18 000.

```
[11]: Target = 'price'
plt.figure(figsize=(10,6))
sns.boxplot(data=df_auto[Target], orient="v", palette="Set1" ,whis=1.
             →5,saturation=1, width=0.7)
```

```
plt.title("Distribution of the target variable", fontsize = 14, fontweight = 'bold')
plt.ylabel("Price ", fontweight = 'bold')
```

[11]: Text(0, 0.5, 'Price ')

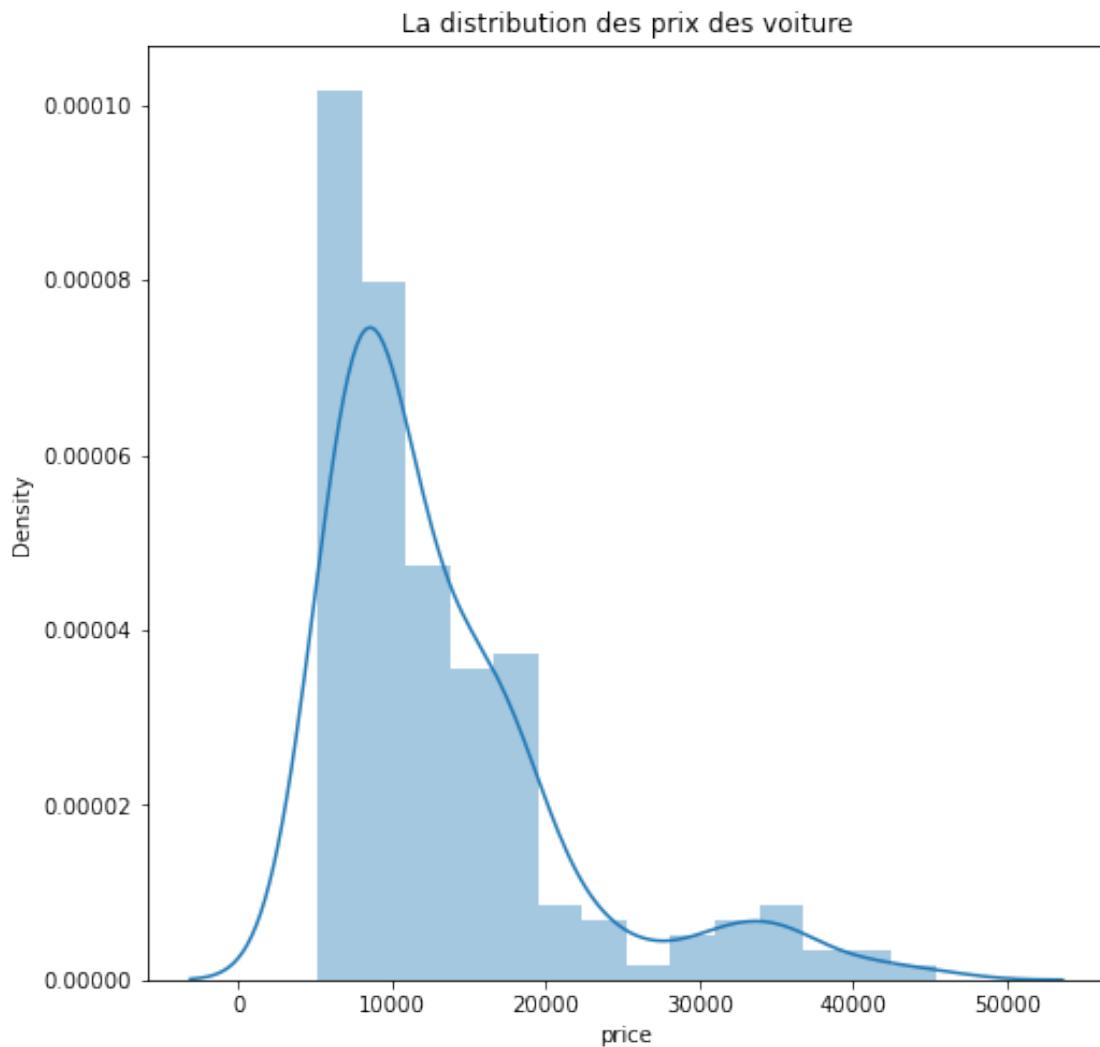


[12]: plt.figure(figsize=(8,8))

```
plt.title('La distribution des prix des voitures')
sns.distplot(df_auto['price'])
```

```
/Users/mac/opt/anaconda3/lib/python3.8/site-
packages/seaborn/distributions.py:2551: FutureWarning: `distplot` is a
deprecated function and will be removed in a future version. Please adapt your
code to use either `displot` (a figure-level function with similar flexibility)
or `histplot` (an axes-level function for histograms).
warnings.warn(msg, FutureWarning)
```

[12]: <AxesSubplot:title={'center':'La distribution des prix des voitures'},
 xlabel='price', ylabel='Density'>



### Examen des variables

- Colone NameCar:
  - Toyota semble être la voiture la plus appréciée.
  - Mercury semble être la voiture la moins appréciée.
- Observation globale des variable quantitative:
  - carwidth , carlength, curbweight ,enginesize ,horsepower semble avoir une corrélation poitive avec le prix.
  - carheight ne montre aucune tendance significative avec le prix.
  - citympg , highwaympg - semblent avoir une corrélation négative significative avec le prix.
- Observation globale des variable qualitative:
  - Les voitures avec fueltype comme diesel sont relativement chères que les voitures avec fueltype comme gas.
  - Tous les types de carrosseries sont relativement moins chers que les carrosseries “con-

vertibles".

- Les voitures avec " rear enginelocation sont beaucoup plus chères que les voitures avec front enginelocation.
- Le prix de la voiture est directement proportionnel à no. de cylindres dans la plupart des cas.
- Le type de moteur ohcv entre dans les voitures de gamme de prix plus élevée.
- DoorNumber n'affecte pas beaucoup le prix.
- Les voitures HigerEnd semblent avoir une roue motrice "rwd"

```
[202]: df['CarName']=df['CarName'].str.split(' ',expand=True).loc[:,0]
df['CarName'].unique()
```

```
[202]: array(['alfa-romero', 'audi', 'bmw', 'chevrolet', 'dodge', 'honda',
       'isuzu', 'jaguar', 'maxda', 'mazda', 'buick', 'mercury',
       'mitsubishi', 'Nissan', 'nissan', 'peugeot', 'plymouth', 'porsche',
       'porcshce', 'renault', 'saab', 'subaru', 'toyota', 'toyoutu',
       'vokswagen', 'volkswagen', 'vw', 'volvo'], dtype=object)
```

**Erreurs de frappe dans le nom de l'entreprise automobile** - maxda = mazda - Nissan = Nissan - porsche = porsche - toyota = toyota - volkswagen = volkswagen = vw

```
[203]: df['CarName'] = df['CarName'].replace({'maxda': 'mazda', 'nissan': 'Nissan',
                                         'porcshce': 'porsche', 'toyoutu': 'toyota',
                                         'vokswagen': 'volkswagen', 'vw': 'volkswagen'})
```

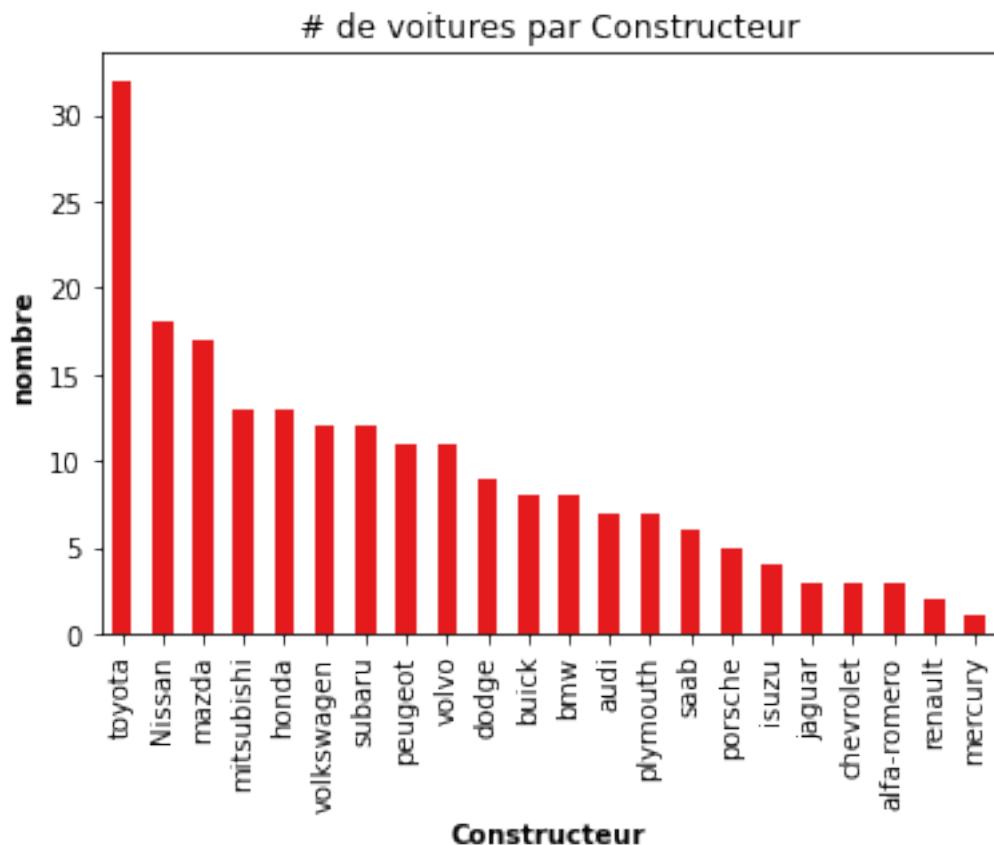
```
[204]: df = df.drop('car_ID',axis=1)
```

```
[205]: df = df.copy()
df['symboling'] = df['symboling'].astype(str)
```

```
[206]: cat_col = df.select_dtypes(include=['object']).columns
num_col = df.select_dtypes(exclude=['object']).columns
df_cat = df[cat_col]
df_num = df[num_col]
```

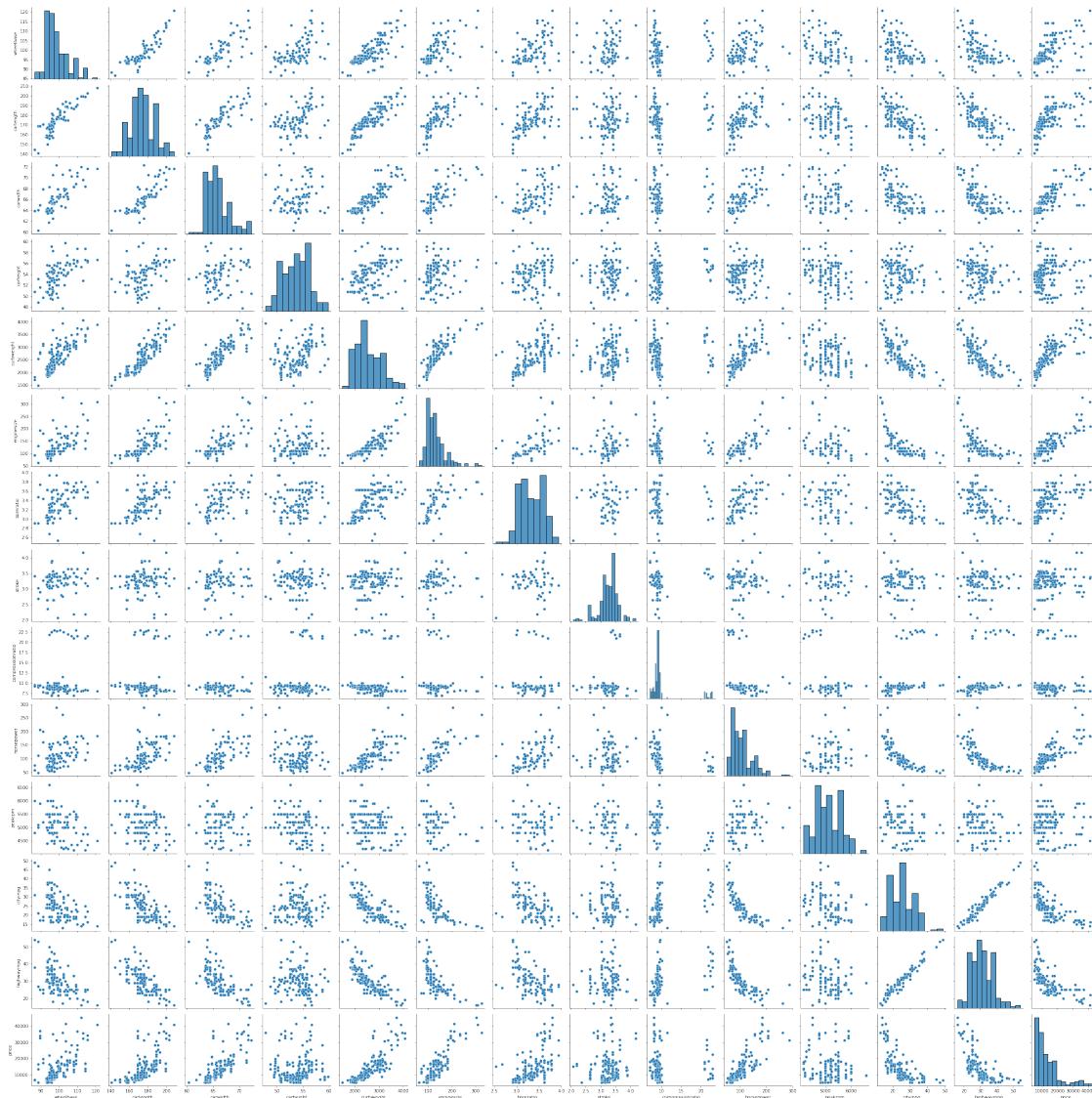
```
[190]: ax=df['CarName'].value_counts().plot(kind='bar',stacked=True, colormap = 'Set1')
ax.title.set_text('# de voitures par Constructeur')
plt.xlabel("Constructeur",fontweight = 'bold')
plt.ylabel("nombre",fontweight = 'bold')
```

```
[190]: Text(0, 0.5, 'nombre')
```



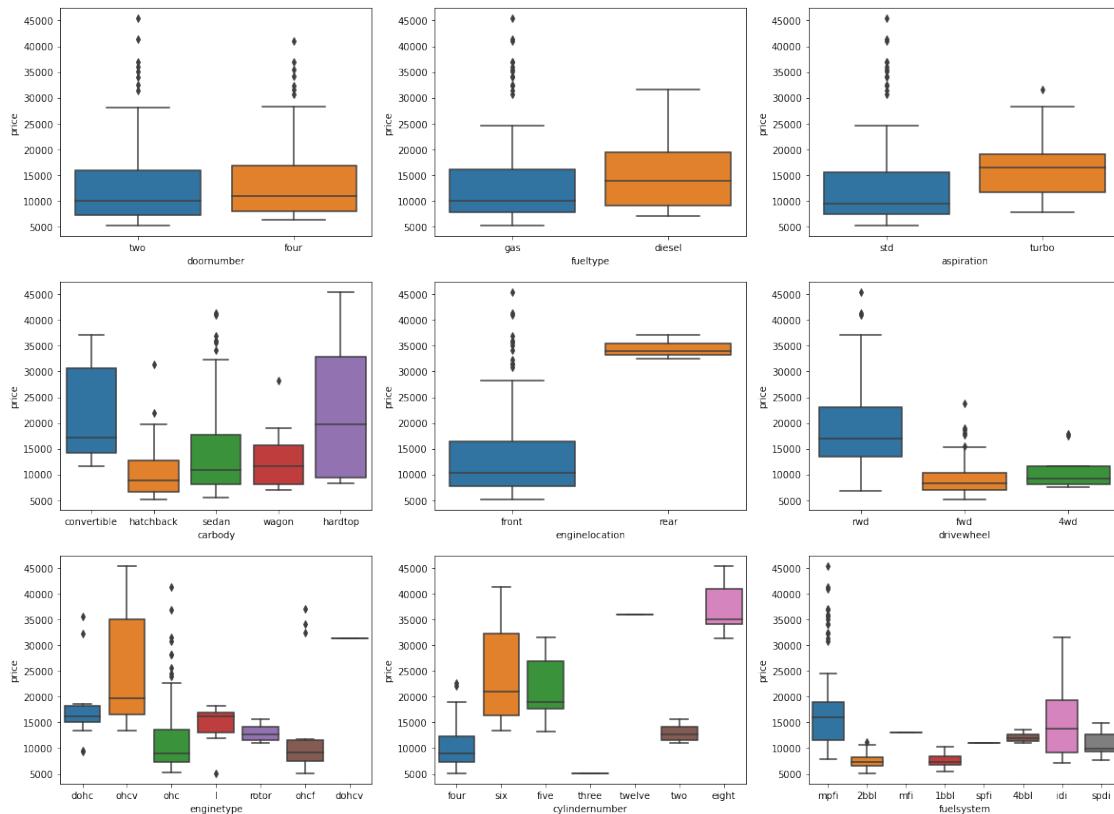
### histogrammes des variables continues

```
[33]: sns.pairplot(df[num_col])
```



```
[34]: plt.figure(figsize=(20, 15))
plt.subplot(3,3,1)
sns.boxplot(x = 'doornumber', y = 'price', data = df)
plt.subplot(3,3,2)
sns.boxplot(x = 'fueltype', y = 'price', data = df)
plt.subplot(3,3,3)
sns.boxplot(x = 'aspiration', y = 'price', data = df)
plt.subplot(3,3,4)
sns.boxplot(x = 'carbody', y = 'price', data = df)
plt.subplot(3,3,5)
sns.boxplot(x = 'enginelocation', y = 'price', data = df)
plt.subplot(3,3,6)
sns.boxplot(x = 'drivewheel', y = 'price', data = df_auto)
plt.subplot(3,3,7)
sns.boxplot(x = 'enginetype', y = 'price', data = df)
```

```
plt.subplot(3,3,8)
sns.boxplot(x = 'cylindernumber', y = 'price', data = df)
plt.subplot(3,3,9)
sns.boxplot(x = 'fuelsystem', y = 'price', data = df)
plt.show()
```



### Relation entre variables et Target

- Le nombre de cylindres utilisés dans la plupart des voitures est de "quatre".
- Le nombre de voitures à essence est bien supérieur à celui des voitures à carburant "diesel".
- "Sedan" est le type de voiture préféré.
- Jaguar, Buick et Porsche sont les voitures les plus chères.
- hardtop et convertible ont le prix le plus élevé.

```
[38]: plt.figure(figsize=(25, 6))

plt.subplot(1,3,1)
plt1 = df['cylindernumber'].value_counts().plot.bar()
plt.title('Nombre de cylindres')
plt1.set(xlabel = 'Number of cylinders', ylabel='Frequency de Nombre of cylinders')
```

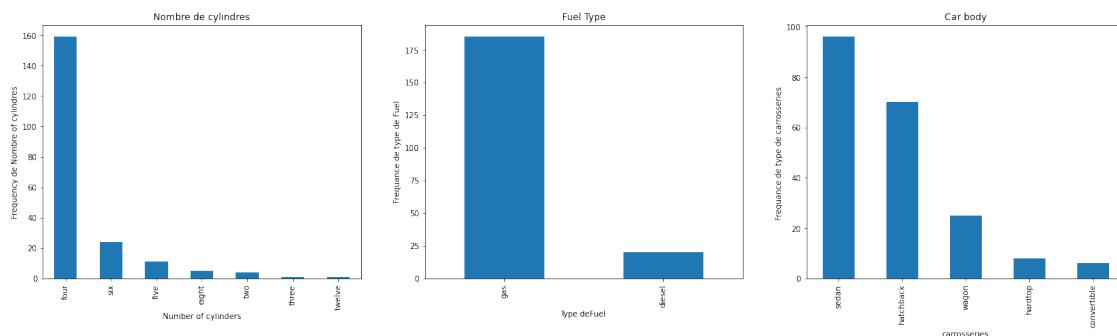
```

plt.subplot(1,3,2)
plt1 = df['fueltype'].value_counts().plot.bar()
plt.title('Fuel Type')
plt1.set(xlabel = 'Type de Fuel ', ylabel='Frequence de type de Fuel')

plt.subplot(1,3,3)
plt1 = df['carbody'].value_counts().plot.bar()
plt.title('Car body')
plt1.set(xlabel = 'carrosseries', ylabel='Frequence de type de carrosseries ')

```

[38]: [Text(0.5, 0, 'carrosseries'),  
Text(0, 0.5, 'Frequence de type de carrosseries ')]



[41]: plt.figure(figsize = (30, 6))

```

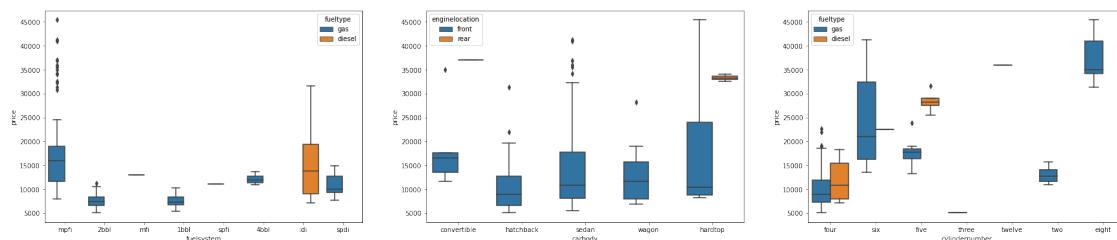
plt.subplot(1,3,1)
sns.boxplot(x = 'fuelsystem', y = 'price', hue = 'fueltype', data = df)

plt.subplot(1,3,2)
sns.boxplot(x = 'carbody', y = 'price', hue = 'enginelocation', data = df_auto)

plt.subplot(1,3,3)
sns.boxplot(x = 'cylindernumber', y = 'price', hue = 'fueltype', data = df_auto)

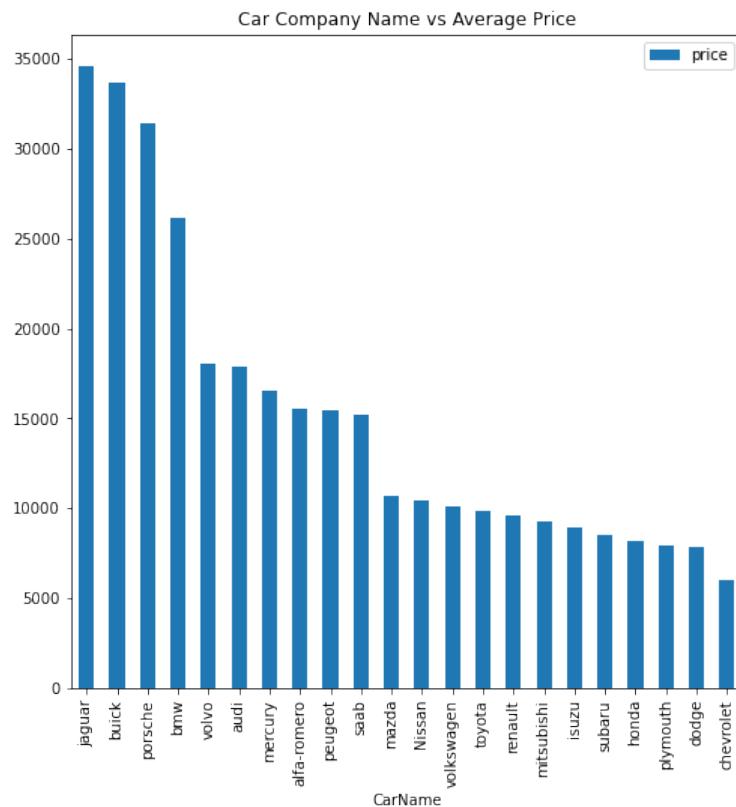
```

[41]: <AxesSubplot:xlabel='cylindernumber', ylabel='price'>

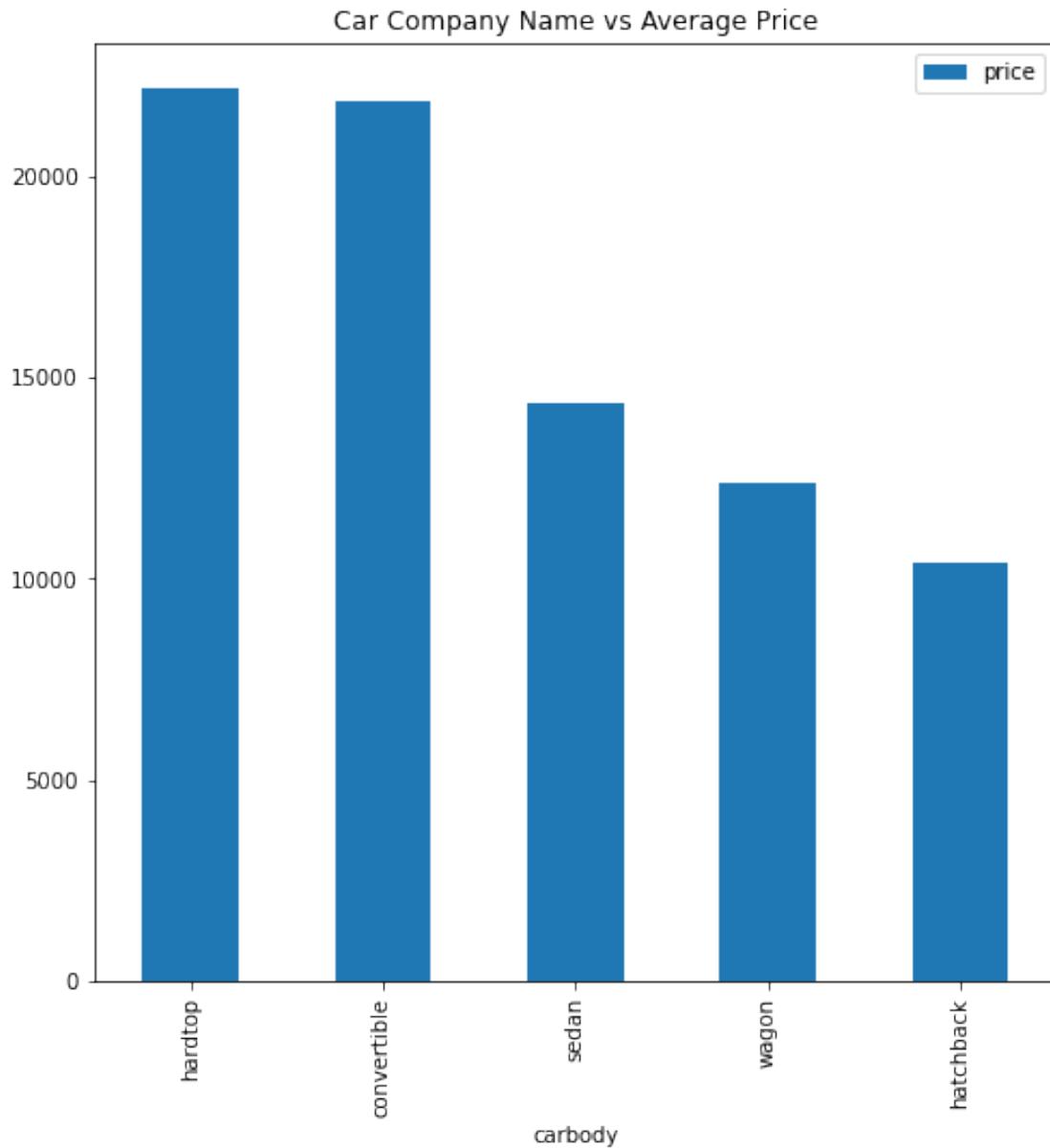


[47]: plt.figure(figsize=(20, 6))
df\_autox = pd.DataFrame(df.groupby(['CarName'])['price'].mean()).
→sort\_values(ascending = False)

```
df_autox.plot.bar()  
plt.title('Car Company Name vs Average Price')
```



```
[48]: df_autoy = pd.DataFrame(df.groupby(['carbody'])['price'].mean().  
                           sort_values(ascending = False))  
df_autoy.plot.bar()  
plt.title('Car Company Name vs Average Price')  
plt.show()
```



[207] : #Binning the Car Companies based on avg prices of each car Company.

```
df['price'] = df['price'].astype('int')
df_temp = df.copy()
t = df_temp.groupby(['CarName'])['price'].mean()
df_temp = df_temp.merge(t.reset_index(), how='left', on='CarName')
bins = [0,10000,20000,40000]
label =['Budget_Friendly','Medium_Range','TopNotch_Cars']
df['Cars_Category'] = pd.cut(df_temp['price_y'],bins,right=False,labels=label)
df.head()
```

```
[207]: symboling      CarName fueltype aspiration doornumber      carbody \
0      3 alfa-romero    gas      std      two convertible
1      3 alfa-romero    gas      std      two convertible
2      1 alfa-romero    gas      std      two hatchback
3      2 audi           gas      std      four sedan
4      2 audi           gas      std      four sedan

drivewheel enginelocation wheelbase carlength ... fuelsystem boreratio \
0     rwd       front   88.6000 168.8000 ... mpfi      3.4700
1     rwd       front   88.6000 168.8000 ... mpfi      3.4700
2     rwd       front   94.5000 171.2000 ... mpfi      2.6800
3     fwd       front   99.8000 176.6000 ... mpfi      3.1900
4     4wd      front   99.4000 176.6000 ... mpfi      3.1900

stroke compressionratio horsepower peakrpm citympg highwaympg price \
0 2.6800         9.0000      111      5000      21      27 13495
1 2.6800         9.0000      111      5000      21      27 16500
2 3.4700         9.0000      154      5000      19      26 16500
3 3.4000        10.0000      102      5500      24      30 13950
4 3.4000        8.0000      115      5500      18      22 17450

Cars_Category
0 Medium_Range
1 Medium_Range
2 Medium_Range
3 Medium_Range
4 Medium_Range

[5 rows x 26 columns]
```

## Conclusion

Les variables significatives sont: - Cars\_Category , Engine Type, Fuel Type - Car Body , Aspiration , Cylinder Number - Drivewheel , Curbweight , Car Length - Car Length , Car width , Engine Size - Boreratio , Horse Power , Wheel base - citympg , highwaympg , symboling

### 6.3.3 Preprocessing

#### Preparation des donnees

```
[208]: sig_col = ['price', 'Cars_Category', 'enginetype', 'fueltype', 'aspiration', 'carbody', 'cylindernumber', 'drivewheel', 'wheelbase', 'curbweight', 'enginesize', 'boreratio', 'horsepower', 'citympg', 'highwaympg', 'carlength', 'carwidth']

df = df[sig_col]
```

```
[210]: sig_cat_col = [
    →['Cars_Category', 'fueltype', 'aspiration', 'carbody', 'drivewheel', 'enginetype', 'cylindernumber']
dummies = pd.get_dummies(df[sig_cat_col])
dummies.shape
```

[210]: (205, 29)

```
[211]: dummies = pd.get_dummies(df[sig_cat_col], drop_first = True)
dummies.shape
```

[211]: (205, 22)

```
[212]: df = pd.concat([df, dummies], axis = 1)
df.drop( sig_cat_col, axis = 1, inplace = True)
df.shape
```

[212]: (205, 32)

```
[214]: df.head()
```

	price	wheelbase	curbweight	enginesize	boreratio	horsepower	citympg	\
0	13495	88.6000	2548	130	3.4700	111	21	
1	16500	88.6000	2548	130	3.4700	111	21	
2	16500	94.5000	2823	152	2.6800	154	19	
3	13950	99.8000	2337	109	3.1900	102	24	
4	17450	99.4000	2824	136	3.1900	115	18	

	highwaympg	carlength	carwidth	...	enginetype_ohc	enginetype_ohcf	\
0	27	168.8000	64.1000	...	0	0	
1	27	168.8000	64.1000	...	0	0	
2	26	171.2000	65.5000	...	0	0	
3	30	176.6000	66.2000	...	1	0	
4	22	176.6000	66.4000	...	1	0	

	enginetype_ohcv	enginetype_rotor	cylindernumber_five	\
0	0	0	0	
1	0	0	0	
2	1	0	0	
3	0	0	0	
4	0	0	1	

	cylindernumber_four	cylindernumber_six	cylindernumber_three	\
0	1	0	0	
1	1	0	0	
2	0	1	0	
3	1	0	0	
4	0	0	0	

```
cylindernumber_twelve cylindernumber_two  
0 0 0  
1 0 0  
2 0 0  
3 0 0  
4 0 0  
  
[5 rows x 32 columns]
```

## Splitting date

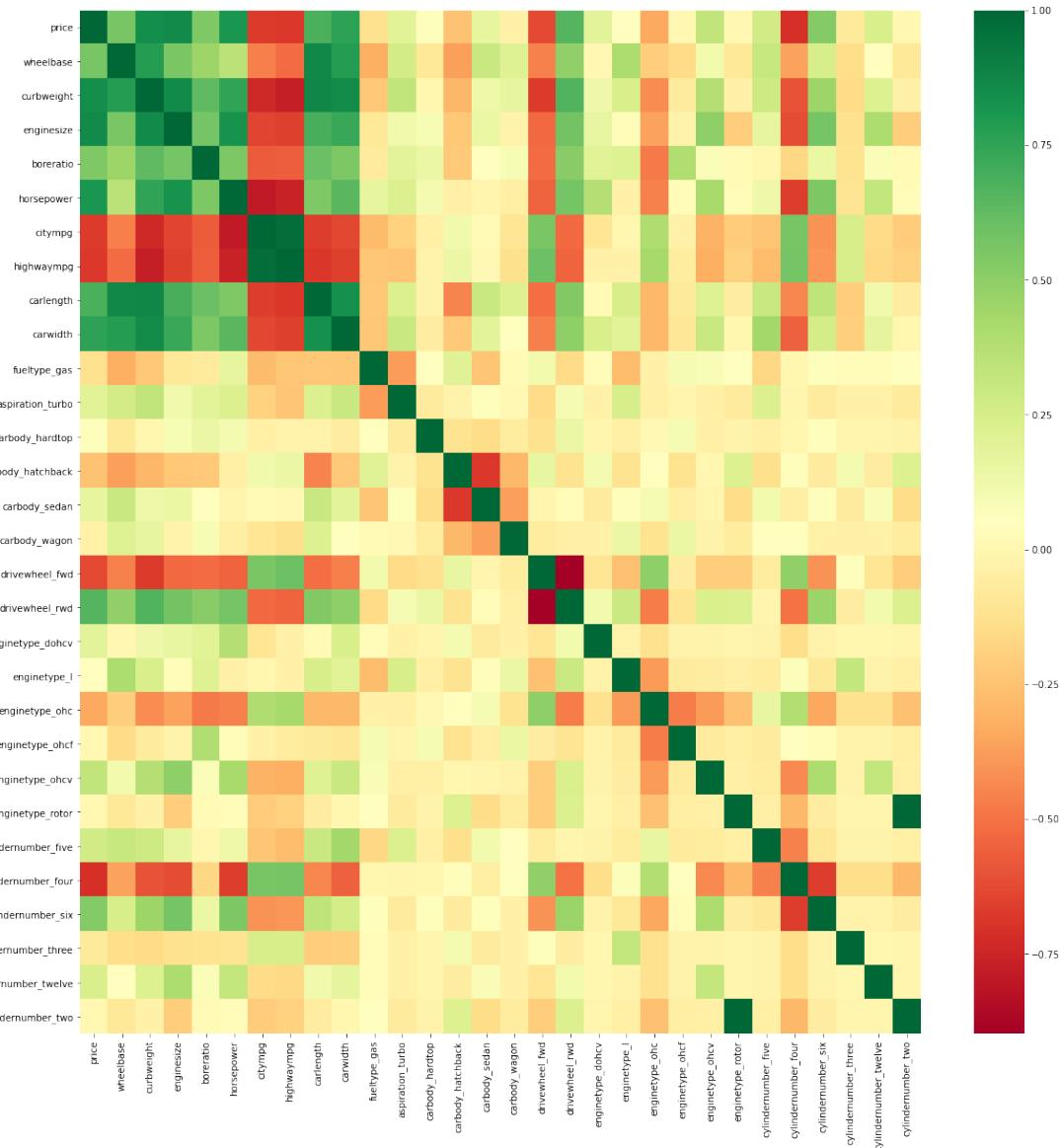
```
[215]: from sklearn.model_selection import train_test_split  
from sklearn.preprocessing import StandardScaler  
  
np.random.seed(0)  
df_train, df_test = train_test_split(df, train_size = 0.8, test_size = 0.2,  
                                     random_state = 100)
```

## Standarization des donnees

```
[216]: scaler = StandardScaler()
```

```
[217]: sig_num_col = ['wheelbase', 'carlength', 'carwidth', 'curbweight', 'enginesize', 'boreratio', 'horsepower', 'citympg', 'highwaympg']  
df_train[sig_num_col] = scaler.fit_transform(df_train[sig_num_col])
```

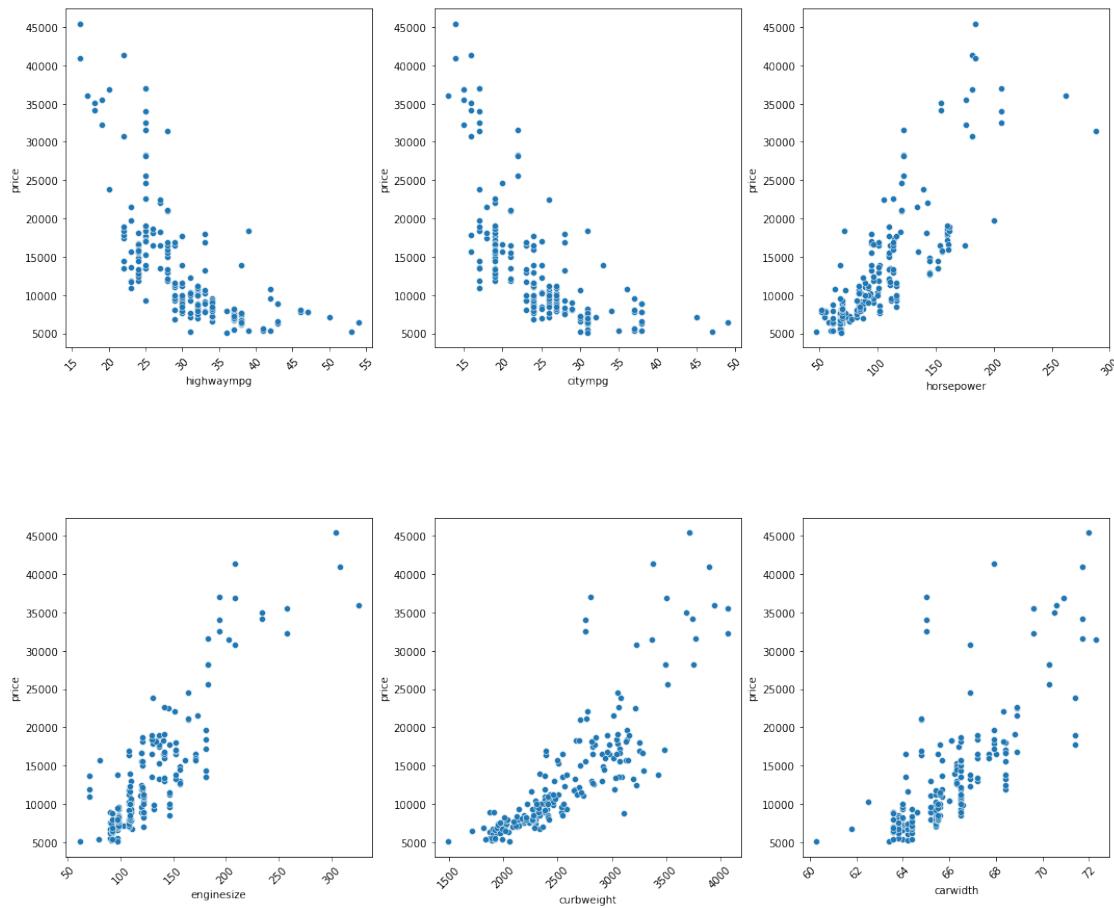
```
[41]: plt.figure(figsize = (20, 20))  
sns.heatmap(df_train.corr(), cmap="RdYlGn")  
plt.show()
```



```
[70]: col = ['highwaympg', 'citympg', 'horsepower', 'enginesize', 'curbweight', 'carwidth']

fig,axes = plt.subplots(2,3,figsize=(18,15))
for seg,col in enumerate(col):
    x,y = seg//3,seg%3
    an=sns.scatterplot(x=col, y='price' ,data=df_auto, ax=axes[x,y])
    plt.setp(an.get_xticklabels(), rotation=45)

plt.subplots_adjust(hspace=0.5)
```



```
[218]: y_train = df_train.pop('price')
X_train = df_train
```

## 6.4 Modelisation du probleme

### 6.4.1 test des Hypothese

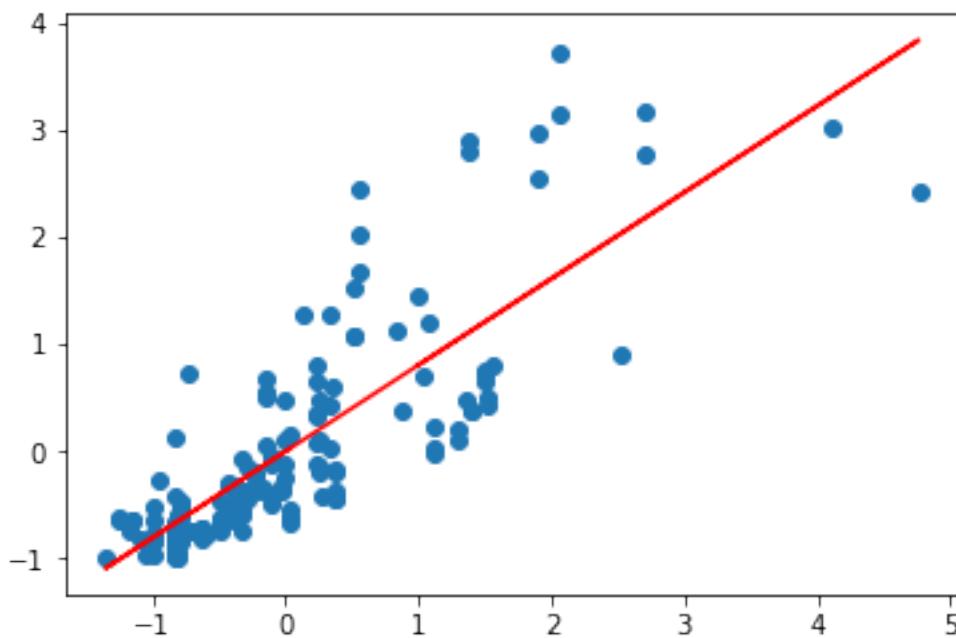
```
[219]: import statsmodels.api as sm
X_train_1 = X_train['horsepower']
# Add a constant
X_train_1c = sm.add_constant(X_train_1)

# Create a first fitted model
lr_1 = sm.OLS(y_train, X_train_1c).fit()

lr_1.params
```

```
[219]: const      0.0000
horsepower  0.8126
dtype: float64
```

```
[46]: plt.scatter(X_train_1c.iloc[:, 1], y_train)
plt.plot(X_train_1c.iloc[:, 1], 0.8062*X_train_1c.iloc[:, 1], 'r')
plt.show()
```



```
[220]: print(lr_1.summary())
```

### OLS Regression Results

```
=====
Dep. Variable:                  price      R-squared:                 0.660
Model:                          OLS        Adj. R-squared:            0.658
Method:                         Least Squares   F-statistic:             314.9
Date:                          Sun, 04 Sep 2022   Prob (F-statistic):       7.96e-40
Time:                           11:31:36          Log-Likelihood:           -144.16
No. Observations:                  164          AIC:                      292.3
Df Residuals:                     162          BIC:                      298.5
Df Model:                          1
Covariance Type:                nonrobust
=====
```

	coef	std err	t	P> t	[0.025	0.975]
const	5.378e-17	0.046	1.17e-15	1.000	-0.090	0.090
horsepower	0.8126	0.046	17.746	0.000	0.722	0.903

```
=====
Omnibus:                      36.811    Durbin-Watson:            1.855
Prob(Omnibus):                  0.000    Jarque-Bera (JB):         61.090
Skew:                           1.126    Prob(JB):                  5.42e-14
Kurtosis:                        4.967    Cond. No.                   1.00
=====
```

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

La valeur R-carré obtenue est '0,65'. Puisque nous avons tellement de variables, nous pouvons clairement faire mieux que cela. Alors allons-y et ajoutons l'autre variable hautement corrélée, c'est-à-dire le 'curbweight'.

```
[221]: X_train_2 = X_train[['horsepower', 'curbweight']]
# Add a constant
X_train_2c = sm.add_constant(X_train_2)

# Create a second fitted model
lr_2 = sm.OLS(y_train, X_train_2c).fit()
```

```
[222]: lr_2.params
```

```
[222]: const      0.0000
horsepower  0.4067
curbweight   0.5392
dtype: float64
```

```
[223]: print(lr_2.summary())
```

OLS Regression Results						
=====						
Dep. Variable:	price	R-squared:	0.786			
Model:	OLS	Adj. R-squared:	0.784			
Method:	Least Squares	F-statistic:	296.1			
Date:	Sun, 04 Sep 2022	Prob (F-statistic):	1.15e-54			
Time:	11:31:46	Log-Likelihood:	-106.19			
No. Observations:	164	AIC:	218.4			
Df Residuals:	161	BIC:	227.7			
Df Model:	2					
Covariance Type:	nonrobust					
=====						
	coef	std err	t	P> t	[0.025	0.975]
-----						
const	5.378e-17	0.036	1.48e-15	1.000	-0.072	0.072
horsepower	0.4067	0.055	7.345	0.000	0.297	0.516
curbweight	0.5392	0.055	9.738	0.000	0.430	0.648
=====						
Omnibus:	37.859	Durbin-Watson:	1.788			
Prob(Omnibus):	0.000	Jarque-Bera (JB):	87.458			
Skew:	0.987	Prob(JB):	1.02e-19			
Kurtosis:	5.983	Cond. No.	2.66			
=====						

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

```
[224]: X_train_3 = X_train[['horsepower', 'curbweight', 'enginesize']]
# Add a constant
X_train_3c = sm.add_constant(X_train_3)

# Create a third fitted model
lr_3 = sm.OLS(y_train, X_train_3c).fit()
lr_3.params
```

```
[224]: const      0.0000
horsepower  0.2721
curbweight   0.3485
enginesize   0.3379
dtype: float64
```

```
[225]: print(lr_3.summary())
```

### OLS Regression Results

```
=====
Dep. Variable:                  price      R-squared:                 0.807
Model:                          OLS        Adj. R-squared:            0.804
Method:                         Least Squares    F-statistic:             223.5
Date:                          Sun, 04 Sep 2022    Prob (F-statistic):       5.60e-57
Time:                           11:31:55        Log-Likelihood:          -97.678
No. Observations:                  164        AIC:                      203.4
Df Residuals:                     160        BIC:                      215.8
Df Model:                           3
Covariance Type:                nonrobust
=====
            coef    std err         t      P>|t|      [0.025      0.975]
-----
const     7.98e-17    0.035     2.3e-15    1.000     -0.069      0.069
horsepower  0.2721    0.062      4.406    0.000      0.150      0.394
curbweight   0.3485    0.070      4.999    0.000      0.211      0.486
enginesize   0.3379    0.081      4.183    0.000      0.178      0.497
=====
Omnibus:                   29.472    Durbin-Watson:           1.843
Prob(Omnibus):                  0.000    Jarque-Bera (JB):        65.487
Skew:                           0.778    Prob(JB):                  6.02e-15
Kurtosis:                        5.676    Cond. No.                   4.66
=====
```

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly

specified.

Élimination récursives des variables.

Utilisons l'élimination récursive des fonctionnalités car nous avons trop de variables indépendantes. Exécution de RFE avec le numéro de sortie de la variable égal à 15

```
[228]: from sklearn.linear_model import LinearRegression
from sklearn.feature_selection import RFE
lm = LinearRegression()
lm.fit(X_train, y_train)

rfe = RFE(lm, 15)
rfe = rfe.fit(X_train, y_train)
```

```
[229]: for i in range(0,X_train.columns.shape[0]):
    print(f'{X_train.columns[i]}:{rfe.support_[i]}:{rfe.ranking_[i]}')
```

```
wheelbase-----0---,9
curbweight-----1---,1
enginesize-----0---,12
boreratio-----0---,8
horsepower-----1---,1
citympg-----0---,5
highwaympg-----0---,10
carlength-----0---,13
carwidth-----1---,1
Cars_Category_Medium_Range---0---,2
Cars_Category_TopNotch_Cars---1---,1
fueltype_gas-----0---,14
aspiration_turbo-----0---,15
carbody_hardtop-----1---,1
carbody_hatchback-----1---,1
carbody_sedan-----1---,1
carbody_wagon-----1---,1
drivewheel_fwd-----0---,11
drivewheel_rwd-----0---,6
enginetype_dohcv-----1---,1
enginetype_l-----1---,1
enginetype_ohc-----1---,1
enginetype_ohcf-----1---,1
enginetype_ohcv-----1---,1
enginetype_rotor-----0---,17
cylindernumber_five-----1---,1
cylindernumber_four-----1---,1
cylindernumber_six-----0---,4
cylindernumber_three-----0---,7
cylindernumber_twelve-----0---,3
cylindernumber_two-----0---,16
```

### 6.4.2 Construction des modèles de prediction

```
[230]: col_sup = X_train.columns[rfe.support_]
col_sup
```

```
X_train_rfe = X_train[col_sup]
X_train_rfe.head()
```

```
[230]:    curbweight  horsepower  carwidth  Cars_Category_TopNotch_Cars \
3           -0.3433      0.0242   0.2059                      0
157        -0.7818     -0.7912  -0.6301                      0
81         -0.3606     -0.3325  -0.1657                      0
32         -1.3048     -1.0460  -0.8159                      0
99         -0.3683     -0.1032  -0.2586                      0

carbody_hardtop  carbody_hatchback  carbody_sedan  carbody_wagon \
3                  0                 0             1             0
157                0                 1             0             0
81                  0                 1             0             0
32                  0                 1             0             0
99                  0                 1             0             0

enginetype_dohcv  enginetype_l  enginetype_ohc  enginetype_ohcf \
3                  0                 0             1             0
157                0                 0             1             0
81                  0                 0             1             0
32                  0                 0             1             0
99                  0                 0             1             0

enginetype_ohcv  cylindernumber_five  cylindernumber_four
3                  0                 0             1
157                0                 0             1
81                  0                 0             1
32                  0                 0             1
99                  0                 0             1
```

Après avoir passé les colonnes sélectionnées arbitrairement par RFE, nous évaluerons manuellement la valeur p et la valeur VIF de chaque modèle. À moins que nous ne trouvions la plage acceptable pour les valeurs p et VIF, nous continuons à supprimer les variables une à la fois en fonction des critères ci-dessous.

- valeur-p élevée VIF élevé : Supprimer la variable
- VIF faible valeur-p élevée ou VIF élevé valeur-p faible : supprimez d'abord la variable à valeur-p élevée
- valeur-p faible VIF faible : accepter la variable

Le facteur d'inflation de la variance ou VIF donne une idée quantitative de base de la corrélation entre les variables de caractéristiques. C'est un paramètre extrêmement important pour tester notre modèle linéaire. La formule de calcul de VIF est la suivante :

$$VIF_i = \frac{1}{1 - R_i^2} \quad (6.1)$$

```
[231]: def train_model(X_train,y_train):
    X_trainc = sm.add_constant(X_train)
    lm_rfe = sm.OLS(y_train,X_trainc).fit()

    #Summary of linear model
    print(lm_rfe.summary())
```

```
[232]: def get_VIF(X):
    vif = pd.DataFrame()
    vif['Features'] = X.columns
    vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.
    ↪shape[1])]
    vif['VIF'] = round(vif['VIF'], 2)
    vif = vif.sort_values(by = "VIF", ascending = False)
    return vif
```

```
[233]: train_model(X_train_rfe,y_train)
```

### OLS Regression Results

```
=====
Dep. Variable:                  price      R-squared:                 0.938
Model:                          OLS        Adj. R-squared:            0.932
Method:                         Least Squares   F-statistic:                149.6
Date:                          Sun, 04 Sep 2022   Prob (F-statistic):       2.59e-81
Time:                           11:32:29      Log-Likelihood:           -4.5365
No. Observations:                  164      AIC:                      41.07
Df Residuals:                      148      BIC:                      90.67
Df Model:                           15
Covariance Type:                nonrobust
=====
```

		coef	std err	t	P> t
[0.025	0.975]				
-----					
const		0.8039	0.150	5.353	0.000
0.507	1.101				
curbweight		0.2256	0.062	3.646	0.000
0.103	0.348				
horsepower		0.2570	0.045	5.694	0.000
0.168	0.346				
carwidth		0.2197	0.048	4.586	0.000
0.125	0.314				
Cars_Category_TopNotch_Cars		1.0507	0.101	10.381	0.000
0.851	1.251				
carbody_hardtop		-0.6596	0.184	-3.581	0.000

```

-1.024      -0.296
carbody_hatchback          -0.8164    0.133   -6.150    0.000
-1.079      -0.554
carbody_sedan              -0.7020    0.132   -5.314    0.000
-0.963      -0.441
carbody_wagon               -0.8287    0.143   -5.796    0.000
-1.111      -0.546
enginetype_dohcv            -0.8684    0.327   -2.652    0.009
-1.515      -0.221
enginetype_l                 0.2664    0.132   2.015    0.046
0.005       0.528
enginetype_ohc                0.3051    0.107   2.860    0.005
0.094       0.516
enginetype_ohcf                0.2761    0.125   2.209    0.029
0.029       0.523
enginetype_ohcv                -0.1792    0.117   -1.533    0.127
-0.410       0.052
cylindernumber_five             -0.3776    0.135   -2.792    0.006
-0.645       -0.110
cylindernumber_four                -0.5267    0.100   -5.271    0.000
-0.724      -0.329
=====
Omnibus:                      35.523   Durbin-Watson:           2.154
Prob(Omnibus):                  0.000   Jarque-Bera (JB):        84.840
Skew:                           0.911   Prob(JB):                  3.78e-19
Kurtosis:                        6.016   Cond. No.                  28.6
=====
```

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[234]: `get_VIF(X_train_rfe)`

	Features	VIF
10	enginetype_ohc	19.5600
14	cylindernumber_four	14.5800
0	curbweight	9.1200
6	carbody_sedan	7.2000
2	carwidth	5.2500
5	carbody_hatchback	4.8800
1	horsepower	4.6100
7	carbody_wagon	3.1400
11	enginetype_ohcf	2.9500
3	Cars_Category_TopNotch_Cars	2.4900
13	cylindernumber_five	2.4300
9	enginetype_l	2.2800

```

12          enginetype_ohcv  1.6300
8          enginetype_dohcv  1.5600
4          carbody_hardtop  1.4500

```

```
[235]: X_train_rfe1 = X_train_rfe.drop(['enginetype_ohcv'], 1, )
train_model(X_train_rfe1,y_train)
get_VIF(X_train_rfe1)
```

OLS Regression Results

---

Dep. Variable:	price	R-squared:	0.937
Model:	OLS	Adj. R-squared:	0.931
Method:	Least Squares	F-statistic:	158.7
Date:	Sun, 04 Sep 2022	Prob (F-statistic):	6.42e-82
Time:	11:32:52	Log-Likelihood:	-5.8277
No. Observations:	164	AIC:	41.66
Df Residuals:	149	BIC:	88.15
Df Model:	14		
Covariance Type:	nonrobust		

---

		coef	std err	t	P> t
[0.025	0.975]				
-----	-----	-----	-----	-----	-----
const		0.7350	0.144	5.107	0.000
0.451	1.019				
curbweight		0.2292	0.062	3.691	0.000
0.107	0.352				
horsepower		0.2523	0.045	5.577	0.000
0.163	0.342				
carwidth		0.2100	0.048	4.402	0.000
0.116	0.304				
Cars_Category_TopNotch_Cars		1.0601	0.101	10.447	0.000
0.860	1.261				
carbody_hardtop		-0.6529	0.185	-3.530	0.001
-1.018	-0.287				
carbody_hatchback		-0.8131	0.133	-6.099	0.000
-1.077	-0.550				
carbody_sedan		-0.7010	0.133	-5.283	0.000
-0.963	-0.439				
carbody_wagon		-0.8310	0.144	-5.787	0.000
-1.115	-0.547				
enginetype_dohcv		-0.7660	0.322	-2.379	0.019
-1.402	-0.130				
enginetype_l		0.3194	0.128	2.492	0.014
0.066	0.573				

```

enginetype_ohc          0.3475    0.103    3.359    0.001
0.143      0.552
enginetype_ohcf          0.3228    0.122    2.651    0.009
0.082      0.563
cylindernumber_five     -0.3396    0.134   -2.543    0.012
-0.603      -0.076
cylindernumber_four     -0.5050    0.099   -5.083    0.000
-0.701      -0.309
=====
Omnibus:                38.750   Durbin-Watson:        2.106
Prob(Omnibus):           0.000    Jarque-Bera (JB):    99.305
Skew:                   0.967    Prob(JB):            2.73e-22
Kurtosis:                6.285   Cond. No.             28.1
=====
```

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[235]:

	Features	VIF
10	enginetype_ohc	18.2100
13	cylindernumber_four	14.5800
0	curbweight	9.1200
6	carbody_sedan	6.0300
2	carwidth	5.2200
1	horsepower	4.5000
5	carbody_hatchback	4.0700
11	enginetype_ohcf	2.7900
7	carbody_wagon	2.7700
3	Cars_Category_TopNotch_Cars	2.4900
12	cylindernumber_five	2.4000
9	enginetype_l	2.1400
8	enginetype_dohcv	1.4800
4	carbody_hardtop	1.4200

[237]: X\_train\_rfe2 = X\_train\_rfe1.drop('enginetype\_dohcv', 1,)  
train\_model(X\_train\_rfe2,y\_train)  
get\_VIF(X\_train\_rfe2)

#### OLS Regression Results

```

=====
Dep. Variable:                  price    R-squared:                 0.935
Model:                          OLS     Adj. R-squared:            0.929
Method: Least Squares          F-statistic:              165.3
Date: Sun, 04 Sep 2022          Prob (F-statistic):       7.74e-82
Time: 11:33:52                 Log-Likelihood:           -8.8843
No. Observations:               164     AIC:                      45.77
Df Residuals:                  150     BIC:                      89.17
```

Df Model:	13	Covariance Type:	nonrobust		
<hr/>					
<hr/>					
		coef	std err		
[0.025	0.975]			t	P> t
<hr/>					
const		0.7702	0.145	5.298	0.000
0.483	1.057				
curbweight		0.2900	0.057	5.044	0.000
0.176	0.404				
horsepower		0.2080	0.042	4.968	0.000
0.125	0.291				
carwidth		0.1772	0.046	3.821	0.000
0.086	0.269				
Cars_Category_TopNotch_Cars		1.0090	0.101	10.019	0.000
0.810	1.208				
carbody_hardtop		-0.6592	0.188	-3.510	0.001
-1.030	-0.288				
carbody_hatchback		-0.8409	0.135	-6.235	0.000
-1.107	-0.574				
carbody_sedan		-0.7264	0.134	-5.409	0.000
-0.992	-0.461				
carbody_wagon		-0.8819	0.144	-6.116	0.000
-1.167	-0.597				
enginetype_l		0.3126	0.130	2.403	0.017
0.056	0.570				
enginetype_ohc		0.3775	0.104	3.620	0.000
0.171	0.584				
enginetype_ohcf		0.3681	0.122	3.014	0.003
0.127	0.609				
cylindernumber_five		-0.3470	0.136	-2.559	0.011
-0.615	-0.079				
cylindernumber_four		-0.5446	0.099	-5.476	0.000
-0.741	-0.348				
<hr/>					
Omnibus:	38.756	Durbin-Watson:	2.059		
Prob(Omnibus):	0.000	Jarque-Bera (JB):	100.014		
Skew:	0.965	Prob(JB):	1.92e-22		
Kurtosis:	6.304	Cond. No.	25.4		
<hr/>					

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[237] :

	Features	VIF
9	enginetype_ohc	17.9600
12	cylindernumber_four	14.3000
0	curbweight	7.5100
6	carbody_sedan	6.0300
2	carwidth	4.7100
5	carbody_hatchback	4.0700
1	horsepower	3.7800
7	carbody_wagon	2.7300
10	enginetype_ohcf	2.7200
3	Cars_Category_TopNotch_Cars	2.4000
11	cylindernumber_five	2.4000
8	enginetype_l	2.1300
4	carbody_hardtop	1.4100

```
[238]: X_train_rfe3 = X_train_rfe2.drop('enginetype_l', 1, )
train_model(X_train_rfe3,y_train)
get_VIF(X_train_rfe3)
```

### OLS Regression Results

```
=====
Dep. Variable:                  price      R-squared:                 0.932
Model:                          OLS        Adj. R-squared:            0.927
Method:                         Least Squares   F-statistic:                173.1
Date:                          Sun, 04 Sep 2022   Prob (F-statistic):        9.64e-82
Time:                           11:34:25       Log-Likelihood:           -11.982
No. Observations:                  164        AIC:                      49.96
Df Residuals:                     151        BIC:                      90.26
Df Model:                          12
Covariance Type:                nonrobust
=====
```

		coef	std err	t	P> t
[0.025	0.975]				
-----					
const		0.7724	0.148	5.231	0.000
0.481	1.064				
curbweight		0.3091	0.058	5.343	0.000
0.195	0.423				
horsepower		0.1784	0.041	4.389	0.000
0.098	0.259				
carwidth		0.1800	0.047	3.823	0.000
0.087	0.273				
Cars_Category_TopNotch_Cars		1.0452	0.101	10.335	0.000
0.845	1.245				
carbody_hardtop		-0.5743	0.187	-3.065	0.003

```

-0.944      -0.204
carbody_hatchback          -0.7897      0.135     -5.838      0.000
-1.057      -0.522
carbody_sedan              -0.6700      0.134     -4.988      0.000
-0.935      -0.405
carbody_wagon               -0.8257      0.145     -5.714      0.000
-1.111      -0.540
enginetype_ohc              0.2283      0.085     2.682      0.008
0.060       0.396
enginetype_ohcf             0.2373      0.111     2.137      0.034
0.018       0.457
cylindernumber_five         -0.2767      0.134     -2.058      0.041
-0.542      -0.011
cylindernumber_four          -0.4518      0.093     -4.853      0.000
-0.636      -0.268
=====
Omnibus:                  43.320     Durbin-Watson:        2.063
Prob(Omnibus):            0.000     Jarque-Bera (JB):    119.139
Skew:                      1.058     Prob(JB):           1.35e-26
Kurtosis:                 6.600     Cond. No.           24.6
=====
```

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[238]:

	Features	VIF
11	cylindernumber_four	11.6500
8	enginetype_ohc	11.5800
0	curbweight	7.3700
6	carbody_sedan	5.4000
2	carwidth	4.7000
5	carbody_hatchback	3.6800
1	horsepower	3.4400
7	carbody_wagon	2.5500
3	Cars_Category_TopNotch_Cars	2.3400
10	cylindernumber_five	2.2800
9	enginetype_ohcf	2.1800
4	carbody_hardtop	1.3400

[239]:

```
X_train_rfe4 = X_train_rfe3.drop('cylindernumber_five', 1, )
train_model(X_train_rfe4,y_train)
get_VIF(X_train_rfe4)
```

OLS Regression Results

```

=====
Dep. Variable:      price    R-squared:     0.930
Model:              OLS      Adj. R-squared:  0.925
```

Method:	Least Squares	F-statistic:	184.5		
Date:	Sun, 04 Sep 2022	Prob (F-statistic):	5.53e-82		
Time:	11:34:41	Log-Likelihood:	-14.251		
No. Observations:	164	AIC:	52.50		
Df Residuals:	152	BIC:	89.70		
Df Model:	11				
Covariance Type:	nonrobust				
<hr/>					
<hr/>					
		coef	std err	t	P> t
[0.025	0.975]				
<hr/>					
const		0.6916	0.144	4.808	0.000
0.407	0.976				
curbweight		0.3057	0.058	5.231	0.000
0.190	0.421				
horsepower		0.1999	0.040	5.035	0.000
0.121	0.278				
carwidth		0.1454	0.044	3.271	0.001
0.058	0.233				
Cars_Category_TopNotch_Cars		1.1062	0.098	11.321	0.000
0.913	1.299				
carbody_hardtop		-0.5359	0.188	-2.845	0.005
-0.908	-0.164				
carbody_hatchback		-0.7482	0.135	-5.535	0.000
-1.015	-0.481				
carbody_sedan		-0.6288	0.134	-4.685	0.000
-0.894	-0.364				
carbody_wagon		-0.7920	0.145	-5.458	0.000
-1.079	-0.505				
enginetype_ohc		0.1377	0.074	1.871	0.063
-0.008	0.283				
enginetype_ohcf		0.1588	0.105	1.507	0.134
-0.049	0.367				
cylindernumber_four		-0.3341	0.074	-4.500	0.000
-0.481	-0.187				
<hr/>					
Omnibus:	50.728	Durbin-Watson:	2.063		
Prob(Omnibus):	0.000	Jarque-Bera (JB):	163.911		
Skew:	1.182	Prob(JB):	2.55e-36		
Kurtosis:	7.289	Cond. No.	24.0		
<hr/>					

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

		Features	VIF
8		enginetype_ohc	8.4100
10		cylindernumber_four	7.8100
0		curbweight	7.3400
6		carbody_sedan	5.2800
2		carwidth	4.2300
5		carbody_hatchback	3.5900
1		horsepower	3.3000
7		carbody_wagon	2.5000
3	Cars_Category_TopNotch_Cars		2.2000
9		enginetype_ohcf	1.9000
4		carbody_hardtop	1.3400

```
[240]: X_train_rfe5 = X_train_rfe4.drop('enginetype_ohcf', 1)
        train_model(X_train_rfe5,y_train)
        get_VIF(X_train_rfe5)
```

OLS Regression Results					
Dep. Variable:	price	R-squared:			0.929
Model:	OLS	Adj. R-squared:			0.925
Method:	Least Squares	F-statistic:			201.1
Date:	Sun, 04 Sep 2022	Prob (F-statistic):			1.17e-82
Time:	11:34:58	Log-Likelihood:			-15.466
No. Observations:	164	AIC:			52.93
Df Residuals:	153	BIC:			87.03
Df Model:	10				
Covariance Type:	nonrobust				
<hr/>					
		coef	std err	t	P> t
[0.025	0.975]				
<hr/>					
const		0.7157	0.144	4.986	0.000
0.432	0.999				
curbweight		0.2767	0.055	4.994	0.000
0.167	0.386				
horsepower		0.2043	0.040	5.138	0.000
0.126	0.283				
carwidth		0.1524	0.044	3.434	0.001
0.065	0.240				
Cars_Category_TopNotch_Cars		1.1492	0.094	12.244	0.000
0.964	1.335				
carbody_hardtop		-0.5005	0.188	-2.667	0.008
-0.871	-0.130				
carbody_hatchback		-0.7463	0.136	-5.498	0.000

-1.014	-0.478			
carbody_sedan		-0.6129	0.134	-4.562
-0.878	-0.348			0.000
carbody_wagon		-0.7579	0.144	-5.266
-1.042	-0.474			0.000
enginetype_ohc		0.0703	0.059	1.198
-0.046	0.186			0.233
cylindernumber_four		-0.3095	0.073	-4.256
-0.453	-0.166			0.000
<hr/>				
Omnibus:	53.714	Durbin-Watson:		2.077
Prob(Omnibus):	0.000	Jarque-Bera (JB):		189.601
Skew:	1.222	Prob(JB):		6.74e-42
Kurtosis:	7.666	Cond. No.		23.9
<hr/>				

## Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

		Features	VIF
9		cylindernumber_four	7.1600
0		curbweight	6.5900
8		enginetype_ohc	5.3500
6		carbody_sedan	4.7300
2		carwidth	4.1900
5		carbody_hatchback	3.4400
1		horsepower	3.2600
7		carbody_wagon	2.1300
3	Cars_Category_TopNotch_Cars		1.9800
4		carbody_hardtop	1.2700

```
[241]: X_train_rfe6 = X_train_rfe5.drop('enginetype_ohc', 1)
        train_model(X_train_rfe6,y_train)
        get_VIF(X_train_rfe6)
```

```
OLS Regression Results
=====
Dep. Variable:          price    R-squared:       0.929
Model:                  OLS      Adj. R-squared:  0.924
Method:                 Least Squares   F-statistic:     222.7
Date: Sun, 04 Sep 2022   Prob (F-statistic): 1.54e-83
Time:           11:35:12    Log-Likelihood:   -16.231
No. Observations:      164      AIC:             52.46
Df Residuals:          154      BIC:            83.46
Df Model:                   9
Covariance Type:        nonrobust
```

		coef	std err	t	P> t
[0.025	0.975]				
const		0.7238	0.144	5.041	0.000
0.440	1.007				
curbweight		0.2628	0.054	4.844	0.000
0.156	0.370				
horsepower		0.1946	0.039	4.992	0.000
0.118	0.272				
carwidth		0.1609	0.044	3.667	0.000
0.074	0.248				
Cars_Category_TopNotch_Cars		1.1790	0.091	13.011	0.000
1.000	1.358				
carbody_hardtop		-0.4667	0.186	-2.511	0.013
-0.834	-0.100				
carbody_hatchback		-0.7191	0.134	-5.366	0.000
-0.984	-0.454				
carbody_sedan		-0.5837	0.132	-4.412	0.000
-0.845	-0.322				
carbody_wagon		-0.7359	0.143	-5.148	0.000
-1.018	-0.454				
cylindernumber_four		-0.2932	0.072	-4.098	0.000
-0.435	-0.152				
<hr/>					
Omnibus:	57.601	Durbin-Watson:	2.073		
Prob(Omnibus):	0.000	Jarque-Bera (JB):	221.512		
Skew:	1.287	Prob(JB):	7.93e-49		
Kurtosis:	8.079	Cond. No.	22.6		
<hr/>					

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

	Features	VIF
8	cylindernumber_four	6.7900
0	curbweight	6.3100
2	carwidth	4.0900
6	carbody_sedan	3.8200
1	horsepower	3.1400
5	carbody_hatchback	2.8500
7	carbody_wagon	1.9700
3	Cars_Category_TopNotch_Cars	1.8200
4	carbody_hardtop	1.2100

```
[242]: X_train_rfe7 = X_train_rfe6.drop('carbody_hardtop', 1, )
train_model(X_train_rfe7,y_train)
get_VIF(X_train_rfe7)
```

OLS Regression Results					
Dep. Variable:	price	R-squared:	0.926		
Model:	OLS	Adj. R-squared:	0.922		
Method:	Least Squares	F-statistic:	241.4		
Date:	Sun, 04 Sep 2022	Prob (F-statistic):	2.09e-83		
Time:	11:35:32	Log-Likelihood:	-19.523		
No. Observations:	164	AIC:	57.05		
Df Residuals:	155	BIC:	84.94		
Df Model:	8				
Covariance Type:	nonrobust				
<hr/>					
		coef	std err	t	P> t
[0.025	0.975]				
<hr/>					
const		0.5159	0.119	4.324	0.000
0.280	0.752				
curbweight		0.2730	0.055	4.962	0.000
0.164	0.382				
horsepower		0.1894	0.040	4.785	0.000
0.111	0.268				
carwidth		0.1560	0.045	3.501	0.001
0.068	0.244				
Cars_Category_TopNotch_Cars		1.1865	0.092	12.882	0.000
1.005	1.368				
carbody_hatchback		-0.5073	0.106	-4.790	0.000
-0.717	-0.298				
carbody_sedan		-0.3758	0.105	-3.580	0.000
-0.583	-0.168				
carbody_wagon		-0.5306	0.119	-4.449	0.000
-0.766	-0.295				
cylindernumber_four		-0.2956	0.073	-4.063	0.000
-0.439	-0.152				
<hr/>					
Omnibus:	50.038	Durbin-Watson:	1.994		
Prob(Omnibus):	0.000	Jarque-Bera (JB):	190.097		
Skew:	1.099	Prob(JB):	5.26e-42		
Kurtosis:	7.795	Cond. No.	15.7		
<hr/>					

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[242] :

	Features	VIF
0	curbweight	6.3000
7	cylindernumber_four	5.9000
2	carwidth	4.0900
5	carbody_sedan	3.4700
1	horsepower	2.9900
4	carbody_hatchback	2.5700
6	carbody_wagon	1.8800
3	Cars_Category_TopNotch_Cars	1.7800

[246] : X\_train\_rfe8 = X\_train\_rfe7.drop('curbweight', 1,  
train\_model(X\_train\_rfe8,y\_train)  
get\_VIF(X\_train\_rfe8)

OLS Regression Results  
=====

Dep. Variable:	price	R-squared:	0.914
Model:	OLS	Adj. R-squared:	0.910
Method:	Least Squares	F-statistic:	236.6
Date:	Sun, 04 Sep 2022	Prob (F-statistic):	1.15e-79
Time:	11:37:38	Log-Likelihood:	-31.611
No. Observations:	164	AIC:	79.22
Df Residuals:	156	BIC:	104.0
Df Model:	7		
Covariance Type:	nonrobust		

=====

	coef	std err	t	P> t
[0.025 0.975]				
-----	-----	-----	-----	-----
const	0.5437	0.128	4.251	0.000
0.291 0.796				
horsepower	0.2738	0.038	7.138	0.000
0.198 0.350				
carwidth	0.3110	0.034	9.119	0.000
0.244 0.378				
Cars_Category_TopNotch_Cars	1.2593	0.098	12.908	0.000
1.067 1.452				
carbody_hatchback	-0.5757	0.113	-5.109	0.000
-0.798 -0.353				
carbody_sedan	-0.3861	0.113	-3.429	0.001
-0.609 -0.164				
carbody_wagon	-0.4388	0.126	-3.471	0.001
-0.688 -0.189				

```

cylindernumber_four      -0.3217      0.078     -4.132      0.000
-0.476     -0.168
=====
Omnibus:                  41.261   Durbin-Watson:          1.998
Prob(Omnibus):            0.000    Jarque-Bera (JB):       154.304
Skew:                      0.883    Prob(JB):                3.11e-34
Kurtosis:                 7.411   Cond. No.                 13.9
=====
```

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[246]:

	Features	VIF
6	cylindernumber_four	5.8800
4	carbody_sedan	3.4600
3	carbody_hatchback	2.5000
0	horsepower	2.3400
1	carwidth	2.0800
5	carbody_wagon	1.7300
2	Cars_Category_TopNotch_Cars	1.7200

[283]:

```
X_train_f = X_train_rfe8.drop('cylindernumber_four', 1, )
train_model(X_train_f,y_train)
get_VIF(X_train_f)
```

OLS Regression Results

```

=====
Dep. Variable:                  price      R-squared:           0.904
Model:                          OLS        Adj. R-squared:      0.901
Method:                         Least Squares      F-statistic:         247.8
Date:                          Sun, 04 Sep 2022      Prob (F-statistic):  2.25e-77
Time:                           11:47:51        Log-Likelihood:     -40.126
No. Observations:                164        AIC:                  94.25
Df Residuals:                   157        BIC:                  116.0
Df Model:                        6
Covariance Type:                nonrobust
=====
```

	coef	std err	t	P> t
[0.025 0.975]				
-----				
const	0.2552	0.112	2.269	0.025
0.033 0.477	0.3331	0.037	8.923	0.000
horsepower	0.3343	0.035	9.466	0.000
0.259 0.407				
carwidth				

```

0.265      0.404
Cars_Category_TopNotch_Cars    1.3581      0.099      13.674      0.000
1.162      1.554
carbody_hatchback            -0.5329      0.118      -4.524      0.000
-0.766      -0.300
carbody_sedan                -0.3552      0.118      -3.011      0.003
-0.588      -0.122
carbody_wagon                -0.4138      0.133      -3.121      0.002
-0.676      -0.152
=====
Omnibus:                      31.676      Durbin-Watson:          1.958
Prob(Omnibus):                0.000      Jarque-Bera (JB):       134.367
Skew:                          0.584      Prob(JB):              6.65e-30
Kurtosis:                     7.278      Cond. No.             12.1
=====
```

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[283]:

	Features	VIF
0	horsepower	2.2100
1	carwidth	1.9200
2	Cars_Category_TopNotch_Cars	1.7000
4	carbody_sedan	1.2100
3	carbody_hatchback	1.0600
5	carbody_wagon	1.0200

### 6.4.3 Evaluation des modeles

#### modele 1

[284]:

```
X_train_fc = sm.add_constant(X_train_f)
modellR = sm.OLS(y_train,X_train_fc).fit()

#Summary of linear model
print(modellR.summary())
```

OLS Regression Results

```

=====
Dep. Variable:                  price      R-squared:           0.904
Model:                          OLS        Adj. R-squared:      0.901
Method:                         Least Squares      F-statistic:         247.8
Date:                           Sun, 04 Sep 2022     Prob (F-statistic):   2.25e-77
Time:                           11:47:55          Log-Likelihood:      -40.126
No. Observations:                 164          AIC:                  94.25
Df Residuals:                   157          BIC:                  116.0
Df Model:                        6
```

```
Covariance Type: nonrobust
=====
=====

            coef      std err      t      P>|t|
[0.025    0.975]
-----
const          0.2552     0.112     2.269     0.025
0.033    0.477
horsepower     0.3331     0.037     8.923     0.000
0.259    0.407
carwidth       0.3343     0.035     9.466     0.000
0.265    0.404
Cars_Category_TopNotch_Cars   1.3581     0.099    13.674     0.000
1.162    1.554
carbody_hatchback   -0.5329     0.118    -4.524     0.000
-0.766   -0.300
carbody_sedan      -0.3552     0.118    -3.011     0.003
-0.588   -0.122
carbody_wagon      -0.4138     0.133    -3.121     0.002
-0.676   -0.152
=====
Omnibus:           31.676   Durbin-Watson:        1.958
Prob(Omnibus):    0.000   Jarque-Bera (JB):    134.367
Skew:              0.584   Prob(JB):            6.65e-30
Kurtosis:         7.278   Cond. No.          12.1
=====
```

Notes:

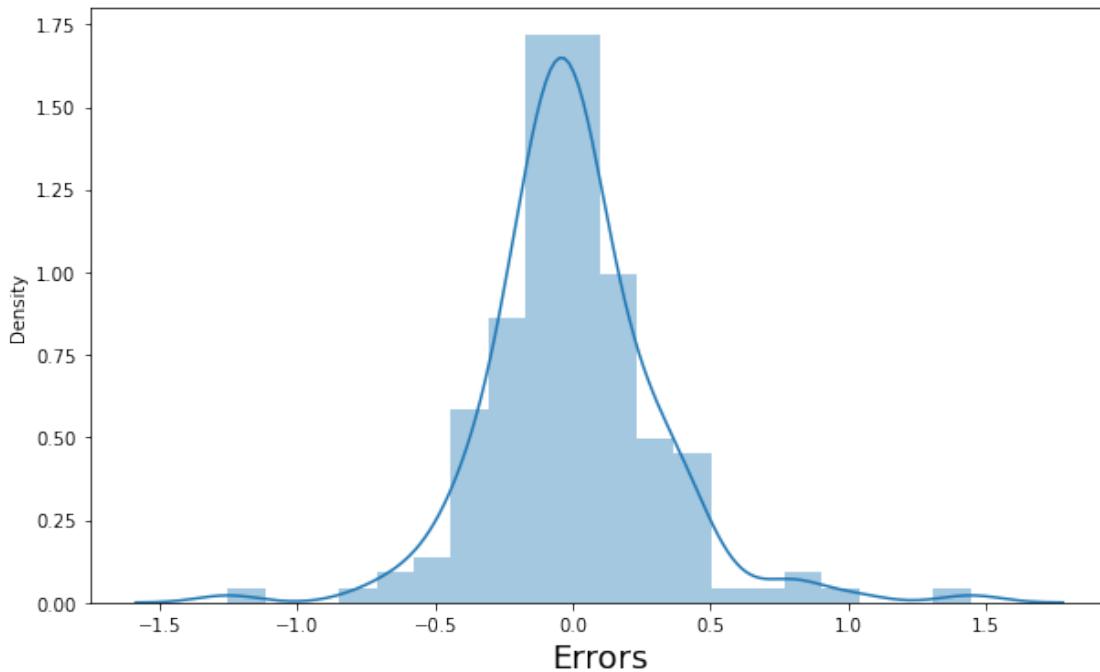
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

```
[285]: # Predicting the price of training set.
y_train_price = modellR.predict(X_train_fc)
```

```
[286]: # Plot the histogram of the error terms
fig = plt.figure(figsize=(10,6))
sns.distplot((y_train - y_train_price), bins = 20)
fig.suptitle('Error Terms Analysis', fontsize = 20)
plt.xlabel('Errors', fontsize = 18)
```

```
[286]: Text(0.5, 0, 'Errors')
```

### Error Terms Analysis



```
[258]: import warnings
warnings.filterwarnings("ignore")
```

```
[259]: df_test[sig_num_col] = scaler.transform(df_test[sig_num_col])
df_test.shape
```

```
[259]: (41, 32)
```

```
[260]: y_test = df_test.pop('price')
X_test = df_test
```

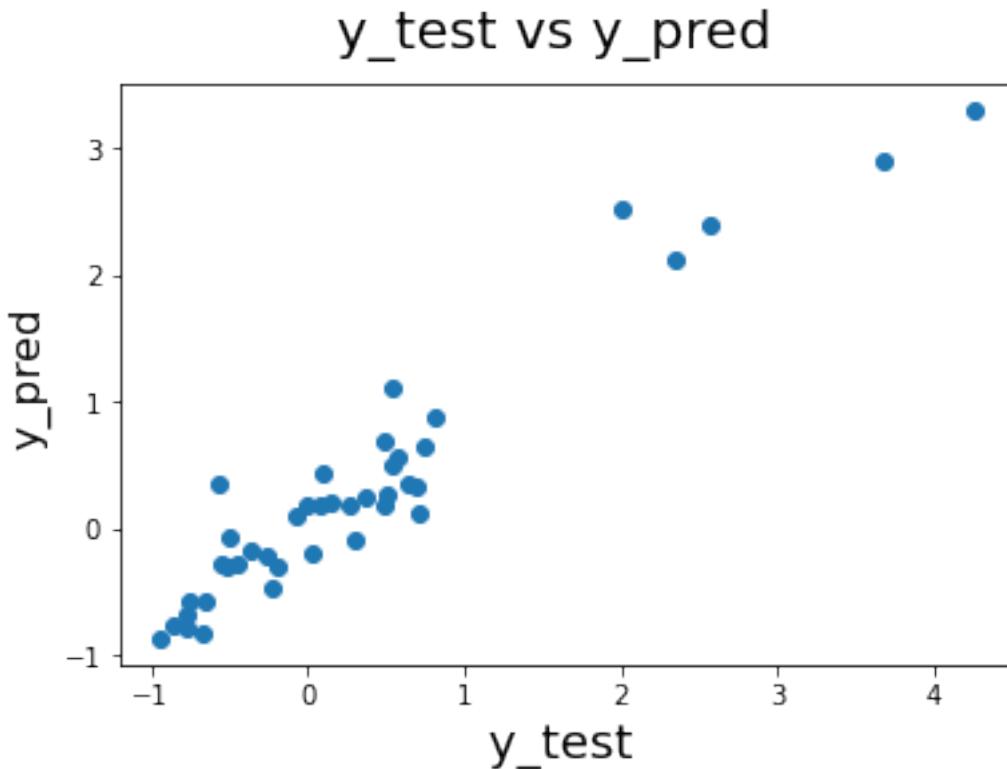
```
[287]: # Adding constant
X_test_1 = sm.add_constant(X_test)

X_test_new = X_test_1[X_train_fc.columns]
```

```
[288]: y_pred = modelLR.predict(X_test_new)
```

```
[289]: # Plotting y_test and y_pred to understand the spread.
fig = plt.figure()
plt.scatter(y_test,y_pred)
fig.suptitle('y_test vs y_pred', fontsize=20)
plt.xlabel('y_test ', fontsize=18)
plt.ylabel('y_pred', fontsize=16)
```

[289]: Text(0, 0.5, 'y\_pred')



[290]: `from sklearn.metrics import r2_score  
r2_score(y_pred,y_test)`

[290]: 0.8805215787691973

## Modele 2

[292]: `X_train_7fc = sm.add_constant(X_train_rfe7)  
modelLR7 = sm.OLS(y_train,X_train_7fc).fit()  
  
#Summary of linear model  
print(modelLR7.summary())`

### OLS Regression Results

```
=====
Dep. Variable:                  price      R-squared:                 0.926
Model:                          OLS        Adj. R-squared:            0.922
Method:                         Least Squares   F-statistic:             241.4
Date: Sun, 04 Sep 2022          Prob (F-statistic):       2.09e-83
Time: 11:50:57                  Log-Likelihood:           -19.523
No. Observations:                164        AIC:                      57.05
Df Residuals:                   155        BIC:                      84.94
Df Model:                       8
```

Covariance Type:	nonrobust				
		coef	std err	t	P> t
[0.025	0.975]				
-----	-----	-----	-----	-----	-----
const		0.5159	0.119	4.324	0.000
0.280	0.752				
curbweight		0.2730	0.055	4.962	0.000
0.164	0.382				
horsepower		0.1894	0.040	4.785	0.000
0.111	0.268				
carwidth		0.1560	0.045	3.501	0.001
0.068	0.244				
Cars_Category_TopNotch_Cars		1.1865	0.092	12.882	0.000
1.005	1.368				
carbody_hatchback		-0.5073	0.106	-4.790	0.000
-0.717	-0.298				
carbody_sedan		-0.3758	0.105	-3.580	0.000
-0.583	-0.168				
carbody_wagon		-0.5306	0.119	-4.449	0.000
-0.766	-0.295				
cylindernumber_four		-0.2956	0.073	-4.063	0.000
-0.439	-0.152				
=====	=====	=====	=====	=====	=====
Omnibus:	50.038	Durbin-Watson:		1.994	
Prob(Omnibus):	0.000	Jarque-Bera (JB):		190.097	
Skew:	1.099	Prob(JB):		5.26e-42	
Kurtosis:	7.795	Cond. No.		15.7	
=====	=====	=====	=====	=====	=====

Notes:

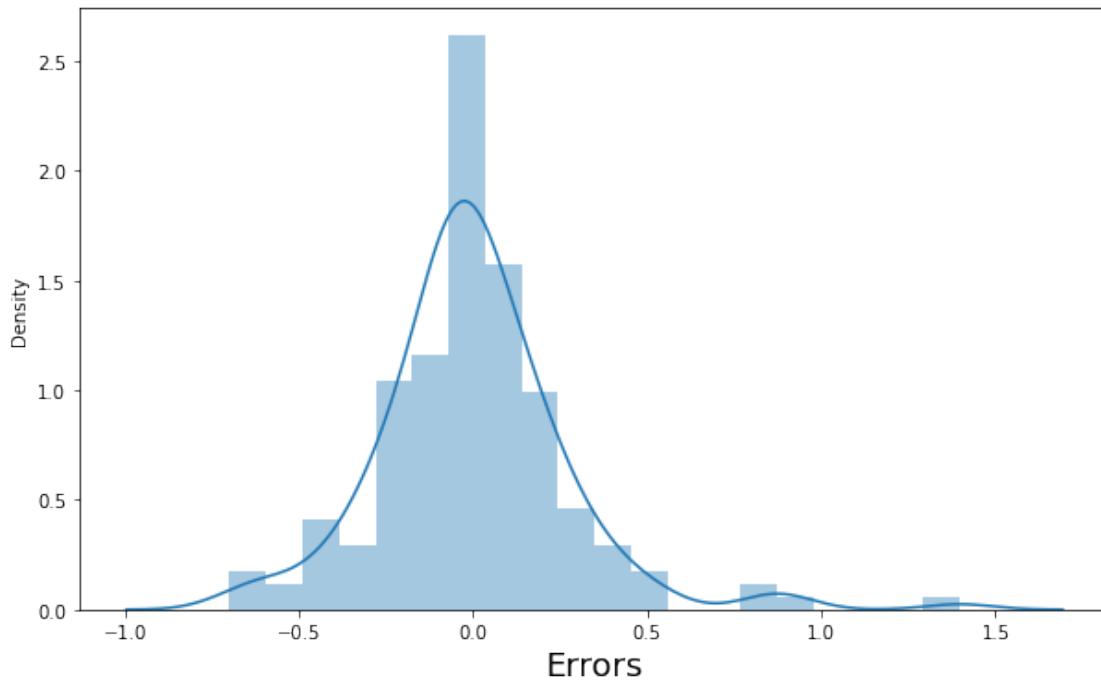
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

```
[297]: # Predicting the price of training set.
y_train_price = modellR7.predict(X_train_7fc)

# Plot the histogram of the error terms
fig = plt.figure(figsize=(10,6))
sns.distplot((y_train - y_train_price), bins = 20)
fig.suptitle('Error Terms Analysis', fontsize = 20)
plt.xlabel('Errors', fontsize = 18)
```

[297]: Text(0.5, 0, 'Errors')

## Error Terms Analysis

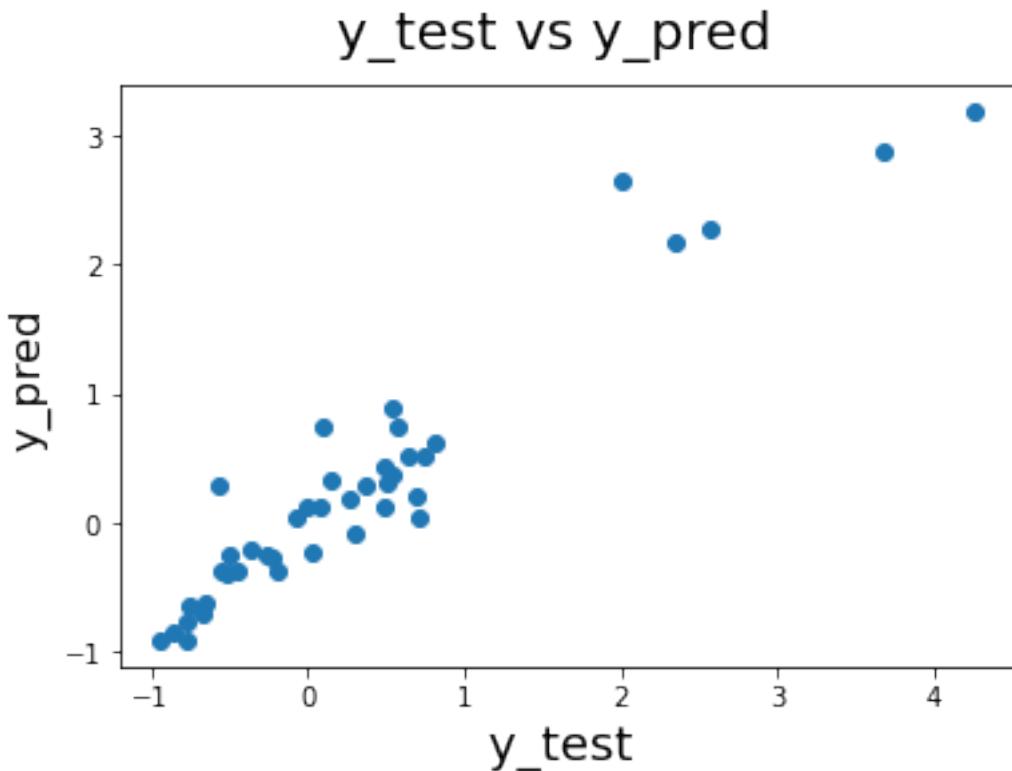


```
[300]: # Adding constant
X_test_1 = sm.add_constant(X_test)

X_test_new = X_test_1[X_train_7fc.columns]
y_pred = modelLR7.predict(X_test_new)
```

```
[301]: # Plotting y_test and y_pred to understand the spread.
fig = plt.figure()
plt.scatter(y_test,y_pred)
fig.suptitle('y_test vs y_pred', fontsize=20)
plt.xlabel('y_test ', fontsize=18)
plt.ylabel('y_pred', fontsize=16)
```

```
[301]: Text(0, 0.5, 'y_pred')
```



```
[302]: r2_score( y_pred,y_test)
```

```
[302]: 0.8744283636538179
```

## 6.5 Conclusion

### 1. modele 1

du modèle 1 :

- R-carré et R-carré ajusté 0,904 et 0,901 et 0,88 R-carré sur l'ensemble de test
- L'ajustement du modèle est significatif et explique que la variance de 88 % n'est pas due au hasard.
- valeur-p pour tous les coefficients semblent être inférieures au seuil de signification de 0,05. - ce qui signifie que tous les prédicteurs sont statistiquement significatifs.

### 2. modele 2

- R-carré et R-carré ajusté 0,926 et 0,922 - et 0,87 R-carré sur l'ensemble de test
- L'ajustement du modèle est significatif et explique que la variance de 87 % n'est tout simplement pas due au hasard.

- valeur-p pour tous les coefficients semblent être inférieures au seuil de signification de 0,05. ce qui signifie que tous les prédicteurs sont statistiquement significatifs.