Python Pour la Science des données: Numpy, Pandas et Matplotlib

Said Gouanene

12/01/2022

1 Numpy

1.1 1- Introduction

- NumPy est une bibliothèque Python créé en 2005 par Travis Oliphant.
- C'est un projet open source et vous pouvez l'utiliser librement.
- NumPy est l'abréviation de "Python Numérique"
- NumPy est utilisé pour travailler avec:
  - des tableaux.
  - des matrices.
  - dans le domaine de l'algèbre linéaire,
  - de la transformée de Fourier et

1.1.1 1.1- Performances Numpy

- Les tableaux sont fréquemment utilisés en science des données, où la vitesse et les ressources sont très importantes.
- En Python, nous avons des listes qui servent à des tableaux, mais elles sont lentes à traiter.
- Contrairement aux listes, les tableaux NumPy sont stockés dans des **espace mémoire contiguës**
- NumPy fournit un objet tableau jusqu'à 50 fois plus rapide que les listes traditionnelles.

- Un objet tableau dans NumPy s'appelle **ndarray**,
- Il fournit de nombreuses fonctions de support qui facilitent le travail **ndarray**.

1.1.2 1.2- Installer et importer le module numpy

[1]:
```
! pip install numpy
import numpy as np
print(np.__version__)
```

```
Requirement already satisfied: numpy in /opt/homebrew/lib/python3.11/site-
packages (2.1.3)
1.26.4
```

1.2 2- Créer un ndarray

[4]:
```
arr = np.array([1, 2, 3, 4, 5])
print(arr)
print(arr.ndim)
type(arr)
```

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

[4]:
```
numpy.ndarray
```

1.2.1 2.1-Tableau de dimmension 0

[3]:
```
arr = np.array(2)
print(arr.ndim)
print(arr)
```

```
0
2
```

1.2.2 2.2-Tableau de dimmension 1

[4]:
```
arr = np.array([2,5,7])
print(arr.ndim)
print(arr)
```

```
1
[2 5 7]
```

1.2.3 2.3-Tableau de dimmension 2

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

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

1.2.4 2.4-Tableau 3D

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

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

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

1.2.5 2.5-Encore plus!!

[7]:
```
arr = np.array([1, 2, 3, 4], ndmin=5)
print(arr.ndim)
print(arr)
```

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

1.3 3-Indexation d'un ndarray

1.3.1 3.1-Indexation normale

Les index dans les tableaux NumPy commencent par 0,

[8]:
```
arr = np.array([1, 2, 3, 4])
print(arr[3])
```

```
4
```

[9]:
```
arr = np.array([[1, 2,7],[ 3, 4, 0]])
print(arr[1])
print(arr[1][0])
print(arr[0,2])
```

```
[3 4 0]
3
7
```

[10]:
```
arr = np.array([[1, 2,7],[ 3, 4, 0],[6, 7, 8]])
print(arr)
```

```
[[1 2 7]
 [3 4 0]
 [6 7 8]]
```

[11]:
```
sub=arr[1]
print(sub)
sub[2]=22
print(sub)
print(arr)
```

```
[3 4 0]
[ 3  4 22]
[[ 1  2  7]
 [ 3  4 22]
 [ 6  7  8]]
```

1.3.2 3.2-Indexation négative

Pour accéder à un tableau à partir de la fin, on utilisez l'indexation négative

[12]:
```
arr = np.array([[1, 2,7],[ 3, 4, 0]])
print(arr)
print(arr[0,-2])
```

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

1.4 4- Ndarray Slicing (Découpage)

Slicing signifie prendre des éléments d'un index donné à un autre index donné Pour cela, il faut indiquer des marges au lieu d'indices: - [start:end] - [start:end:step]

par defaut: - start : 0 - end : le dernier élément - step : 1

[7]:
```
arr = np.array([1, 2, 76, 4, 5, 6, 7, 8])
print(arr[2:5])
```

```
[76  4  5]
```

[11]:
```
print(arr[-2:10:1])
```

```
[7 8]
```

[14]:
```
print(arr[:5])
```

```
[ 1  2 76  4  5]
```

[16]:
```
print(arr[-2:])
```

```
[7]
```

[17]:
```
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8],[9, 10, 11, 12]])

print(arr[1:3, 1:3])
```

```
[[ 6  7]
 [10 11]]
```

[18]:
```
print(arr[::2, 1:3])
```

```
[[ 2  3]
 [10 11]]
```

[19]:
```
print(arr[1:3, 1:4])
```

```
[[ 6  7  8]
 [10 11 12]]
```

Découpage negatif

[20]:
```
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[-4:-1])
```

```
[4 5 6]
```

Exercice

[21]:
```
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8],[9, 10, 11, 12]])

print(arr[-1:-3:-1,-3:])
```

```
[[10 11 12]
 [ 6  7  8]]
```

1.5 5-Types de données dans NumPy

Dans python nous avons les type suivant: String, Integer, Float, Booléen et Complex. Numpy posséde des types de données supplémentaires:

```
i - integer
b - boolean
u - unsigned integer
f - float
c - complex float
m - timedelta
M - datetime
O - object
S - string
U - unicode string
V - fixed chunk of memory for other type ( void )
```

La propriété **dtype** renvoie le type de données d'un ndarray :

[22]:
```
arr = np.array([1, 2, 3, 4])
print(arr.dtype)
```

```
int64
```

[21]:
```
arr = np.array([1, 2, 3.5, 4])
print(arr.dtype)
arr
```

```
float64
```

[21]:
```
array([1. , 2. , 3.5, 4. ])
```

[24]:
```
arr = np.array(["un", "deux", "trois"])
print(arr.dtype)
```

```
<U5
```

1.5.1 5.1- Definir le type de donnée d'un ndarray

La fonction array() peut prendre un argument optionnel : **dtype**, il permet de définir le type de données d'un ndarray :

[25]:
```
arr = np.array([1, 2, 3, 4], dtype='i')
print(arr)
print(arr.dtype)
```

```
[1 2 3 4]
int32
```

[26]:
```
arr = np.array([1, 2, 3, 3], dtype='S')
print(arr)
print(arr.dtype)
```

```
[b'1' b'2' b'3' b'3']
|S1
```

[27]:
```
arr = np.array([1, 2, 3, 64], dtype='f')
print(arr)
print(arr.dtype)
```

```
[ 1.  2.  3. 64.]
float32
```

On peut aussi définir la taille Pour i, u, f, S et U

[28]:
```
arr = np.array([1, 2, 3, 4], dtype='i1')

print(arr)
print(arr.dtype)
```

```
[1 2 3 4]
int8
```

[29]:
```
arr = np.array(['4', 2, '3'], dtype='i')
print(arr)
print(arr.dtype)
```

```
[4 2 3]
int32
```

[30]:
```
arr = np.array(['y', 2, '3'], dtype='i')
print(arr)
print(arr.dtype)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [30], in <cell line: 1>()
----> 1 arr = np.array(['y', 2, '3'], dtype='i')
```

```
      2 print(arr)
      3 print(arr.dtype)

ValueError: invalid literal for int() with base 10: 'y'
```

1.5.2 5.2-Conversion du type de données

Pour changer le type de données d'un tableau existant on utilise la méthode **astype()**. La fonction **astype()** crée une copie du tableau et vous permet de spécifier le nouveau type de données.

[ ]:
```
arr = np.array([1.1, 2.1, 3.1])
newarr = arr.astype(str) # int, bool, float, 'i', 'S', 'f'
print(newarr)
print(arr)
print(newarr.dtype)
```

1.6 6- Copie et Vue d'un ndarray (Copy vs View)

- La copie est un nouveau tableau alors que la vue n'est qu'une vue du tableau d'origine.
- Toute modification apportée à la copie n'affectera pas le tableau d'origine, et toute modification apportée au tableau d'origine n'affectera pas la copie.
- Toute modification apportée à la vue affectera le tableau d'origine, et toute modification apportée au tableau d'origine affectera la vue. ### 6.1 Copie d'un ndarray Pour ceréer une copie d'un ndarray, on utilise la fonction **copy()**

[24]:
```
arr = np.array([1, 2, 3, 4, 5])
x = arr.copy()
arr[0] = 42
print(arr)
print(x)
```

```
[42  2  3  4  5]
[1 2 3 4 5]
```

1.6.1 6.2 Vue d'un ndarray

Pour ceréer une **vue** d'un ndarray, on utilise la fonction **view()**

[25]:
```
arr = np.array([1, 2, 3, 4, 5])
y = arr.view()
#y=arr[:]
arr[0] = 42
print(arr)
print(y)
```

```
[42  2  3  4  5]
[42  2  3  4  5]
```

pour vérifier si un ndarray posséde les donnée, on utilise la propriété **base**. si **base**== **None** alors le tableau posséde les données

[29]:
```
arr = np.array([1, 2, 3, 4, 5])
x = arr.copy()
y = arr.view()
z = arr[1:3]

print(arr.base)
print(x.base)
print(y.base)
print(z.base)
print(type(arr))
print(type(y))
```

```
None
None
[1 2 3 4 5]
[1 2 3 4 5]
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
```

1.7 7- La forme d'un ndarray (shape)

La forme d'un tableau est le nombre d'éléments dans chaque dimension. Un ndarray posséde un attribut appelé **shape** qui renvoie un tuple avec chaque index ayant le nombre d'éléments correspondants.

[30]:
```
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8],[9, 10, 11, 12]])
print(arr.shape)
```

```
(3, 4)
```

[33]:
```
arr = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]],[[9, 10], [11, 12]]])
print(arr.shape)
#2,3,2
a1=np.array([[[1,2],[3,4],[5,6]],[[7,8],[9,10],[11,12]]])
print(a1.shape)
#3,4
a2=np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
print(a2.shape)

a3=np.array([[[1],[2],[3],[4]],[[5],[6],[7],[8]],[[9],[10],[11],[12]]])
print(a3.shape)

a4=np.array([[[[1],[2],[3]]],[[[4],[5],[6]]],[[[1],[2],[3]]],[[[1],[2],[3]]]])
print(a4.shape)
r=arr.reshape((-1,2,2,1))
print(r)
```

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

  [[ 3]
   [ 4]]]


 [[[ 5]
   [ 6]]

  [[ 7]
   [ 8]]]


 [[[ 9]
   [10]]

  [[11]
   [12]]]]
```

[ ]:
```
arr = np.array([1, 2, 3, 4], ndmin=3)
print(arr.shape)
```

1.8 7.1 Changer la forme d'un ndarray

[ ]:
```
arr = np.array([[[[1, 2, 3, 4, 5, 6], [7, 8, 9, 10, 11, 12]]]]) # 12 éléments
print(arr.shape)
arr1 = arr.reshape((6,2)) # 4x3 éléments
print(arr1)
```

[ ]:
```
arr2 = arr.reshape(2, 3, 2) # 2x3x2 éléments
print(arr2)
```

[ ]:
```
arr3 = arr.reshape(2, -1, 2) #calcul la deuxième dimenssion
print(arr3.shape)
arr3[0][0][0]=111
print(arr)
print(arr3)
```

[ ]:
```
print(arr3.base)
```

1.9 8- Parcourir les éléments d'un ndarray

[34]:
```
arr = np.array([1, 2, 3])
for x in arr:
  print(x)
```

```
1
2
3
```

[38]:
```
arr = np.array([[1, 2, 3], [4, 5, 6]])
T=arr>2
print(T.shape)
print(T)
for x in arr:
  print(x)
```

```
(2, 3)
[[False False  True]
 [ True  True  True]]
[1 2 3]
[4 5 6]
```

[47]:
```
arr = np.array([[1, 2, 3], [4, 5, 6]])
sum=0
for x in arr:
  for y in x:
    sum+=y
print(sum)
```

```
21
```

[48]:
```
v=arr.reshape(6)
sum=0
print(v)
for i in v:
    sum+=i
print(sum)
```

```
[1 2 3 4 5 6]
21
```

[49]:
```
s=0
for x in np.nditer(arr):
    s+=x
print(s)
```

```
21
```

[44]:
```
arr = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
for x in np.nditer(arr):
  print(x)
```

[44]:
```
range(0, 20)
```

[54]:
```
print(arr)
print(arr.sum())
```

```
[[1 2 3]
 [4 5 6]]
21
```

[ ]:
```
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
for x in np.nditer(arr[1:2, :]):
  print(x)
```

[ ]:
```
for idx, x in np.ndenumerate(arr):
  print(idx, x)
```

1.10 9- Jointure des ndarray

Dans numpy, la jointure des ndarray se fait suivant un axe. Si l'axe n'est pas spécifier, alors la jointure se fait suivant l'axe 0. Pour joindre deux ndarray, on utilise la fonction **concatinate()**

[62]:
```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.concatenate((arr1, arr2))
print(arr)
```

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

[69]:
```
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6], [7, 8]])
arr = np.concatenate((arr1, arr2),axis=1)
print(arr)
```

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

[ ]:
```
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6], [7, 8]])
arr = np.concatenate((arr1, arr2),axis=1)
print(arr)
```

1.10.1 La fonction stack

- L'empilement est identique à la concaténation, la seule différence est que l'empilement se fait le long d'un **nouvel axe**.

- Nous pouvons concaténer deux tableaux 1-D le long du deuxième axe, ce qui aurait pour effet de les placer l'un sur l'autre, c'est-à-dire. empilement.
- Nous passons une séquence de tableaux que nous voulons joindre à la méthode **stack()** avec l'axe. Si l'axe n'est pas explicitement passé, il est considéré comme 0.

[70]:
```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.stack((arr1, arr2))
print(arr)
```

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

[73]:
```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.stack((arr1, arr2),axis=1)
print(arr)
```

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

1.10.2 Concat Vs Stack

[74]:
```
u=np.array(np.arange(1,9))
v=np.array(np.arange(11,19))
print(np.concatenate((u,v)))
```

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

[75]:
```
print(np.stack((u,v)))
```

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

[76]:
```
print(np.stack((u,v),axis=1))
```

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

[79]:
```
print(u)
print(v)
print(np.hstack((u,v)))
```

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

[ ]:
```
u=u.reshape(2,-1)
v=v.reshape(2,-1)
print(u)
print(v)
```

[ ]:
```
print(np.concatenate((u,v)))
```

[ ]:
```
print(np.concatenate((u,v),axis=1))
```

[ ]:
```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.hstack((arr1, arr2))
print(arr)
```

[ ]:
```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.vstack((arr1, arr2))
print(arr)
```

[ ]:
```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.dstack((arr1, arr2))
print(arr)
```

1.11 10- Fractionner un ndarray

[123]:
```
arr = np.array([1, 2, 3, 4, 5, 6])
newarr = np.array_split(arr, 5) # tester avec 4
print(newarr[0])
```

```
[1 2]
```

[127]:
```
arr = np.array([[1, 2, 6],
                [3, 4, 5],
                [5, 6, 6],
                [7, 8, 6],
                [9, 10, 6],
                [11, 12, 6]])
print(arr.shape)
newarr = np.array_split(arr, 3,axis=0) #axis=1
print(newarr)
#hsplit()
```

```
(6, 3)
[array([[1, 2, 6],
```

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

1.12 11- Recherche dans un ndarray

Pour rechercher une certaine valeur dans un ndarray et renvoyer les index qui correspondent, on utilise la méthode **where()**

[129]:
```
arr = np.array([[1, 2, -3], [4, -5, 8]])
x = np.where(arr >= 0)
print(x)
print(arr[x])
```

```
(array([0, 0, 1, 1]), array([0, 1, 0, 2]))
[1 2 4 8]
```

[88]:
```
arr = np.array([[1, 2, 3], [4, 5, 4]])
x = np.where(arr == 4)
print(x)
```

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

1.12.1 Recherche triée

On suppose que le tableau est trié La méthode **searchsorted()** effectue une recherche binaire dans le tableau et renvoie l'index où la valeur spécifiée serait insérée pour maintenir l'ordre.

[130]:
```
arr = np.array([6, 7, 8, 9])
x = np.searchsorted(arr, 7)
print(x)
```

```
1
```

[90]:
```
arr = np.array([6, 7, 8, 19])
x = np.searchsorted(arr, 18)
print(x)
```

```
3
```

[91]:
```
arr = np.array([1, 3, 5, 7])
x = np.searchsorted(arr, [2, 4, 6])
print(x)
```

```
[1 2 3]
```

1.13 12- Trier les ndarray

L'objet ndarray a une fonction appelée sort(), qui triera un tableau spécifié.

[92]:
```
arr = np.array([3, 2, 0, 1])
v=np.sort(arr)
print(v.base)
print(arr)
```

```
None
[3 2 0 1]
```

[93]:
```
arr = np.array(['Un', 'Deux', 'Trois'])
print(np.sort(arr))
```

```
['Deux' 'Trois' 'Un']
```

[132]:
```
arr = np.array([[3, 2, 4], [5, 0, 1], [-5, 10, 10]])
print(arr)
print(np.sort(arr))# axis=0
```

```
[[ 3  2  4]
 [ 5  0  1]
 [-5 10 10]]
[[-5  0  1]
 [ 3  2  4]
 [ 5 10 10]]
```

1.14 Filtrer les éléments d'un ndarray

Extraire certains éléments d'un tableau existant et créer un nouveau tableau à partir d'eux s'appelle filtering .

Dans NumPy, vous filtrez un tableau à l'aide d'une liste d'index booléens .

Si la valeur à un index est True, alors cet élément est contenu dans le tableau filtré, sinon cet élément est exclu du tableau filtré.

[147]:
```
arr = np.array([1, -2, 1, 4, 2])
x = [False, True, False, False, True]
x1=np.where(arr==2)
newarr1=arr[x1]
newarr = arr[x]
print(newarr1,newarr)
print(newarr1.base,newarr.base)
```

```
[2] [-2  2]
None None
```

[148]:
```
print(arr[arr%2==0])
```

```
[-2  4  2]
```

[149]:
```
filter_arr = []
for element in arr:
```

```
  if element%2==0:
    filter_arr.append(True)
  else:
    filter_arr.append(False)
newarr = arr[filter_arr]
print(filter_arr)
print(newarr)
```

```
[False, True, False, True, True]
[-2  4  2]
```

[ ]:
```
arr = np.array([41, 42, 43, 44])
filter_arr = arr%2==0
newarr = arr[filter_arr]
print(filter_arr)
print(newarr)
```

2 Ufuncs

2.1 Introduction

- ufuncs signifie "Fonctions universelles", des fonctions NumPy qui opèrent sur les obojet ndarray.
- Elles sont utilisées pour implémenter la vectorisation dans NumPy, ce qui est bien plus rapide que l'itération sur les éléments.
- La vectorisation est la conversion d'instructions itératives en une opération vectorielle.

[ ]:
```
x = [1, 2, 3, 4]
y = [4, 5, 6, 7]
z = [] #z=x+y
```

2.2 Créer une ufunc

Pour créer une ufunc, il faut définir une fonction normales en Python, puis l'ajoutez à la ufunc NumPy avec la méthode **frompyfunc()**.

La méthode **frompyfunc(function, inputs, outputs)** prend les arguments suivants :

- function- le nom de la fonction.
- inputs- le nombre d'arguments d'entrée (tableaux).
- outputs- le nombre de tableaux de sortie.

[98]:
```
def add(x, y):
  return x+y
addArr = np.frompyfunc(add, 2, 1)
print(addArr([1, 2, 3, 4], [5, 6, 7, 8]))
print(type(add))
print(type(addArr))
```

```
[6 8 10 12]
<class 'function'>
<class 'numpy.ufunc'>
```

[158]:
```
u = np.array([1, 11, 12, 13, 14, 15])
v = np.array([2, 5, 1, 6, 2, 1])
w = np.absolute(u)
print(w)
#subtract, multiply, divide, power, mod, remainder, divmod, absolute
u%v
```

```
[ 1 11 12 13 14 15]
```

[158]:
```
array([1, 1, 0, 1, 0, 0])
```

2.3 Arrondir les décimales

Il existe principalement 5 façons d'arrondir les décimales dans NumPy : - truncation : Supprimez les décimales et renvoyez le nombre flottant le plus proche de zéro - fix : Supprimez les décimales et renvoyez le nombre flottant le plus proche de zéro - rounding : incrémente le chiffre ou la décimale qui précède de 1 si >=5 sinon ne fait rien - floor : arrondit la décimale à l'entier inférieur le plus proche - ceil : arrondit la décimale à l'entier supérieur le plus proche

[149]:
```
arr = np.array([-3.1645, 3.6667])
a=np.trunc(arr)
print(a)
a=np.fix(arr)
print(a)
a=np.around(arr,2)
print(a)
a=np.floor(arr)
print(a)
a=np.ceil(arr)
print(a)
```

```
[-3.  3.]
[-3.  3.]
[-3.16  3.67]
[-4.  3.]
[-3.  4.]
```

2.3.1 Autres ufunc

log, log2, log10, sum, sin, arcsin, sinh ...

[159]:
```
arr1 = np.array([1, 2, 3])
arr2 = np.array([1, 12, 3])

newarr = np.sum([arr1, arr2])
```

```
print(np.log(arr1))
```

```
[0.         0.69314718 1.09861229]
```

[172]:
```
arr1 = np.array([10, 2,7])
arr2 = np.array([1, 2, 3])
mat=np.vstack((arr1, arr2))
newarr = np.sum(mat,axis=1)
print(mat)
print(newarr)
```

```
[[10  2  7]
 [ 1  2  3]]
[19  6]
```

[173]:
```
arr1 = np.array([1, 2, 3])
arr2 = np.array([1, 2, 3])
newarr = np.sum([arr1, arr2],axis=1)
print(newarr)
```

```
[6 6]
```

[47]:
```
arr = np.array([1, 2, 3])
newarr = np.cumsum(arr)
print(newarr)
```

```
[1 3 6]
```

[175]:
```
arr = np.array([[1, 2, 3],[2,4,5]])
print(arr)
newarr = np.cumsum(arr)
print(newarr)
```

```
[[1 2 3]
 [2 4 5]]
[ 1  3  6  8 12 17]
```

[ ]:
```
arr1 = np.array([1, 2, 3, 4])
arr2 = np.array([5, 6, 7, 8])
newarr = np.prod([arr1, arr2], axis=0) #cumprod
print(newarr)
```

[180]:
```
arr = np.array([[2, 0, 4],[1,1,0]])
newarr = np.diff(arr,axis=0)
print(newarr)
```

```
[[-1  1 -4]]
```

2.4 ufuncs sur les ensembles

[182]:
```
arr = np.array([1, 1, 1, 2, 3, 4, 5,2, 5, 6, 7])
u = np.unique(arr)
print(u)
```

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

[191]:
```
arr1 = np.array([[1, 2,8],[3,3, 4]])
arr2 = np.array([[3, 4,3], [5, 6,8]])
u = np.union1d(arr1, arr2)
print(u)
```

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

[208]:
```
arr1 = np.array([1, 2, 3, 5,5,5,5,4])
arr2 = np.array([3,4, 5, 6])# assume_unique=True
u = np.intersect1d(arr1, arr2)
print(u)
```

```
[3 4 5]
```

[160]:
```
arr1 = np.array([1, 2, 3, 4])
arr2 = np.array([3, 4, 5, 6,9])
u = np.setdiff1d(arr2, arr1)# assume_unique=True
print(u)
```

```
[5 6 9]
```

[161]:
```
arr1 = np.array([1, 2, 3, 4])
arr2 = np.array([3, 4, 2, 6])
u = np.setxor1d(arr1, arr2)#assume_unique=True
print(u)
```

```
[1 6]
```

[162]:
```
arr= np.arange(0, 21, 2)
print(arr)
```

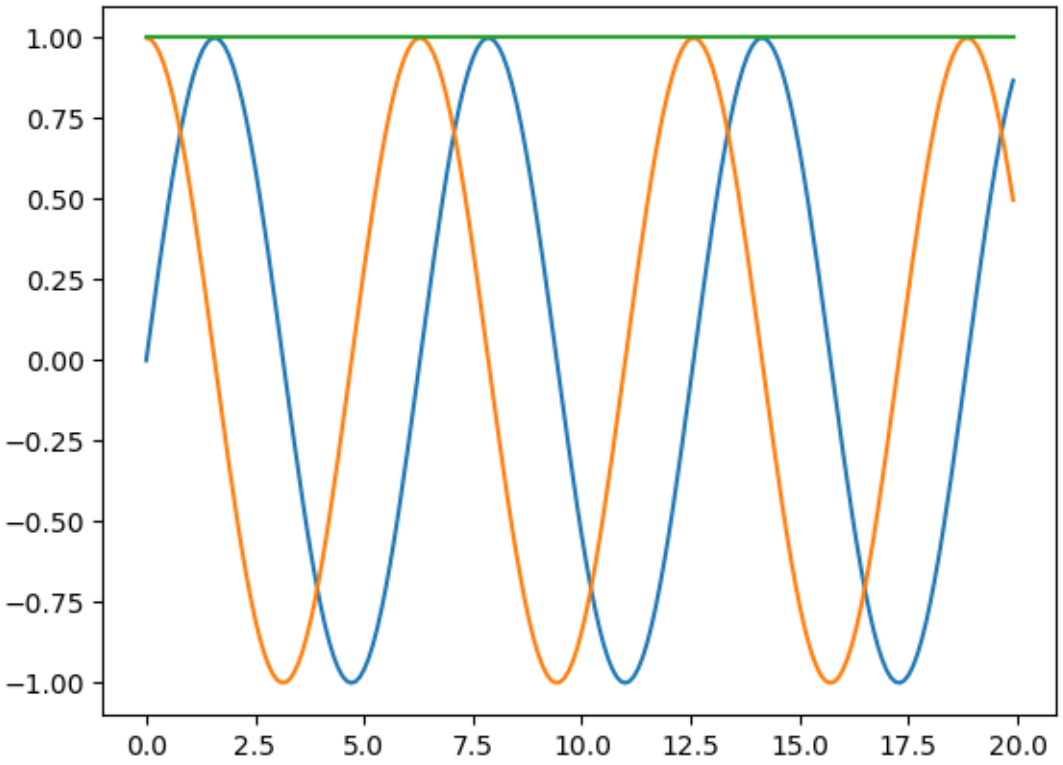
```
[ 0  2  4  6  8 10 12 14 16 18 20]
```

[167]:
```
# Example 4: Create a sequence of 5 values in range 0 to 3
x= np.arange(0, 20, 0.1)

sin=np.sin(x)
cos=np.cos(x)

import matplotlib.pyplot as plt
plt.plot(x,sin)
plt.plot(x,cos)
plt.plot(x,np.add(np.power(sin,2),np.power(cos,2)))
```

```
plt.show()
```



[164]:
```
a=np.arange(0,20,0.1)
a
```

[164]:
```
array([ 0. ,  0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8,  0.9,  1. ,
        1.1,  1.2,  1.3,  1.4,  1.5,  1.6,  1.7,  1.8,  1.9,  2. ,  2.1,
        2.2,  2.3,  2.4,  2.5,  2.6,  2.7,  2.8,  2.9,  3. ,  3.1,  3.2,
        3.3,  3.4,  3.5,  3.6,  3.7,  3.8,  3.9,  4. ,  4.1,  4.2,  4.3,
        4.4,  4.5,  4.6,  4.7,  4.8,  4.9,  5. ,  5.1,  5.2,  5.3,  5.4,
        5.5,  5.6,  5.7,  5.8,  5.9,  6. ,  6.1,  6.2,  6.3,  6.4,  6.5,
        6.6,  6.7,  6.8,  6.9,  7. ,  7.1,  7.2,  7.3,  7.4,  7.5,  7.6,
        7.7,  7.8,  7.9,  8. ,  8.1,  8.2,  8.3,  8.4,  8.5,  8.6,  8.7,
        8.8,  8.9,  9. ,  9.1,  9.2,  9.3,  9.4,  9.5,  9.6,  9.7,  9.8,
        9.9, 10. , 10.1, 10.2, 10.3, 10.4, 10.5, 10.6, 10.7, 10.8, 10.9,
       11. , 11.1, 11.2, 11.3, 11.4, 11.5, 11.6, 11.7, 11.8, 11.9, 12. ,
       12.1, 12.2, 12.3, 12.4, 12.5, 12.6, 12.7, 12.8, 12.9, 13. , 13.1,
       13.2, 13.3, 13.4, 13.5, 13.6, 13.7, 13.8, 13.9, 14. , 14.1, 14.2,
       14.3, 14.4, 14.5, 14.6, 14.7, 14.8, 14.9, 15. , 15.1, 15.2, 15.3,
       15.4, 15.5, 15.6, 15.7, 15.8, 15.9, 16. , 16.1, 16.2, 16.3, 16.4,
       16.5, 16.6, 16.7, 16.8, 16.9, 17. , 17.1, 17.2, 17.3, 17.4, 17.5,
       17.6, 17.7, 17.8, 17.9, 18. , 18.1, 18.2, 18.3, 18.4, 18.5, 18.6,
```

```
       18.7, 18.8, 18.9, 19. , 19.1, 19.2, 19.3, 19.4, 19.5, 19.6, 19.7,
       19.8, 19.9])
```

[166]:
```
b=np.linspace(0,20,100)
b
```

[166]:
```
array([ 0.        ,  0.2020202 ,  0.4040404 ,  0.60606061,  0.80808081,
        1.01010101,  1.21212121,  1.41414141,  1.61616162,  1.81818182,
        2.02020202,  2.22222222,  2.42424242,  2.62626263,  2.82828283,
        3.03030303,  3.23232323,  3.43434343,  3.63636364,  3.83838384,
        4.04040404,  4.24242424,  4.44444444,  4.64646465,  4.84848485,
        5.05050505,  5.25252525,  5.45454545,  5.65656566,  5.85858586,
        6.06060606,  6.26262626,  6.46464646,  6.66666667,  6.86868687,
        7.07070707,  7.27272727,  7.47474747,  7.67676768,  7.87878788,
        8.08080808,  8.28282828,  8.48484848,  8.68686869,  8.88888889,
        9.09090909,  9.29292929,  9.49494949,  9.6969697 ,  9.8989899 ,
       10.1010101 , 10.3030303 , 10.50505051, 10.70707071, 10.90909091,
       11.11111111, 11.31313131, 11.51515152, 11.71717172, 11.91919192,
       12.12121212, 12.32323232, 12.52525253, 12.72727273, 12.92929293,
       13.13131313, 13.33333333, 13.53535354, 13.73737374, 13.93939394,
       14.14141414, 14.34343434, 14.54545455, 14.74747475, 14.94949495,
       15.15151515, 15.35353535, 15.55555556, 15.75757576, 15.95959596,
       16.16161616, 16.36363636, 16.56565657, 16.76767677, 16.96969697,
       17.17171717, 17.37373737, 17.57575758, 17.77777778, 17.97979798,
       18.18181818, 18.38383838, 18.58585859, 18.78787879, 18.98989899,
       19.19191919, 19.39393939, 19.5959596 , 19.7979798 , 20.        ])
```

[101]:
```
# Example 5: Use asarray() convert array
array = np.asarray([20,40,60,80])
array
```

[101]:
```
array([20, 40, 60, 80])
```

[102]:
```
# Example 6: Use empty() create array 3 rows and 4 columns
rr1 = np.empty((3, 4,2))
rr1
```

[102]:
```
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.]]])
```

[168]:
```
# Example 7:Use zero() create array
arr = np.zeros((3,4,2))
arr
```

[168]:
```
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.]]])
```

[94]:
```
# Example 8: Use ones() create array
arr = np.ones((2,4))
arr
```

[94]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

[105]:
```
from numpy import random
```

[173]:
```
#int aleatoire entre 0 et 100
x = random.randint(100)
print(x)
```

```
89
```

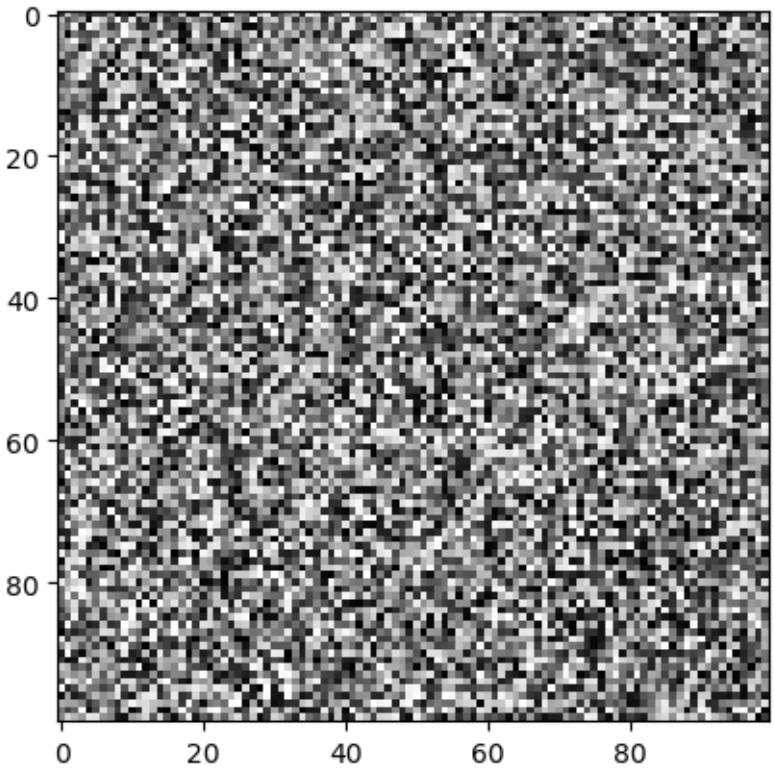
[174]:
```
#Tableau de 5 entiers aleatoire entre 0 et 100
x=random.randint(100, size=(3,3))
print(x)
```

```
[[35 32 23]
 [96 53 87]
 [17 67 15]]
```

[110]:
```
x = random.randint(100, size=(3, 5))
print(x)
from matplotlib import image
from matplotlib import pyplot

x = random.randint(255, size=(100, 100))
pyplot.imshow(x,cmap=pyplot.get_cmap('gray'))
pyplot.show()
```

```
[[11 13 51 93 43]
 [47 55 40 82 17]
 [40 59 19 44 90]]
```



[111]:
```
#float aleatoire entre 0 et 1
x = random.rand()*100
print(x)
```

```
51.74424532046825
```

[248]:
```
x = random.rand(5)
print(x)
```

```
[0.99314376 0.59620004 0.48011138 0.32949584 0.45980546]
```

[104]:
```
x = random.rand(5,3)
print(x)
```

```
[[0.06574461 0.80286608 0.76351973]
 [0.88308438 0.61953663 0.29663565]
 [0.96006712 0.42282145 0.32720951]
 [0.70611839 0.38868795 0.67293233]
 [0.1939297  0.79354486 0.57959309]]
```

2.5 Générer des nombre aléatoires à partir d'un tableau

[179]:
```
x = random.choice([1,2,3,4,5,6])
print(x)
```

```
1
```

[109]:
```
x = random.choice([3, 5, 7, 9], size=(3, 5))
print(x)
```

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

[251]:
```
x = random.choice([3, 5, 7, 9], p=[0.1, 0.2, 0.7, 0], size=(3, 5))
print(x)
```

```
[[5 7 7 5 7]
 [7 3 7 7 7]
 [7 7 7 7 7]]
```

2.6 Distribution normale

- La distribution normale, appelée aussi distribution gaussienne.
- Il correspond à la distribution de probabilité de nombreux événements, par ex. Scores de QI, rythme cardiaque, etc.
- Utilisez la méthode **random.normal()** pour obtenir une distribution normale des données.
- Il a trois paramètres :
  - loc- (Mean) où le pic de la cloche existe.
  - scale- (Écart type) à quel point la distribution du graphique doit être plate.
  - size- La forme du tableau renvoyé.

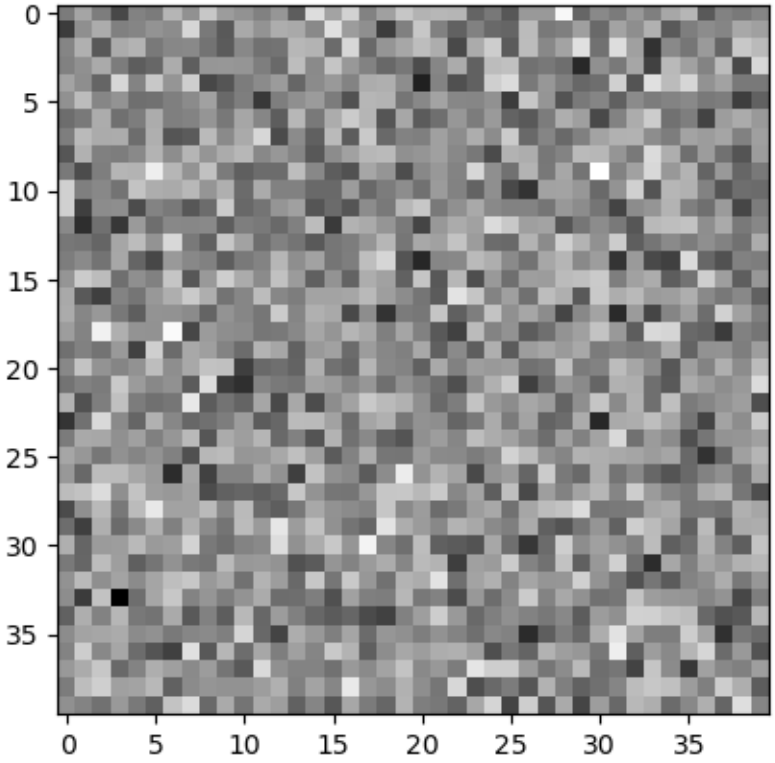
[192]:
```
x = random.normal(size=(10))

print(x)
```

```
[ 0.88175924 -0.1293344   2.31758639 -1.46015159  0.86383055 -0.64787888
 -0.09018273  0.0450445  -0.56120637  0.13089048]
```

[113]:
```
import matplotlib.pyplot as plt
import seaborn as sns
x=random.normal(0,1,size=(40,40))
pyplot.imshow(x,cmap=pyplot.get_cmap('gray'))
pyplot.show()
#sns.displot(x,kde=True)
#sns.displot(random.normal(size=1000))

#plt.show()
```



2.7 Distribution binomiale

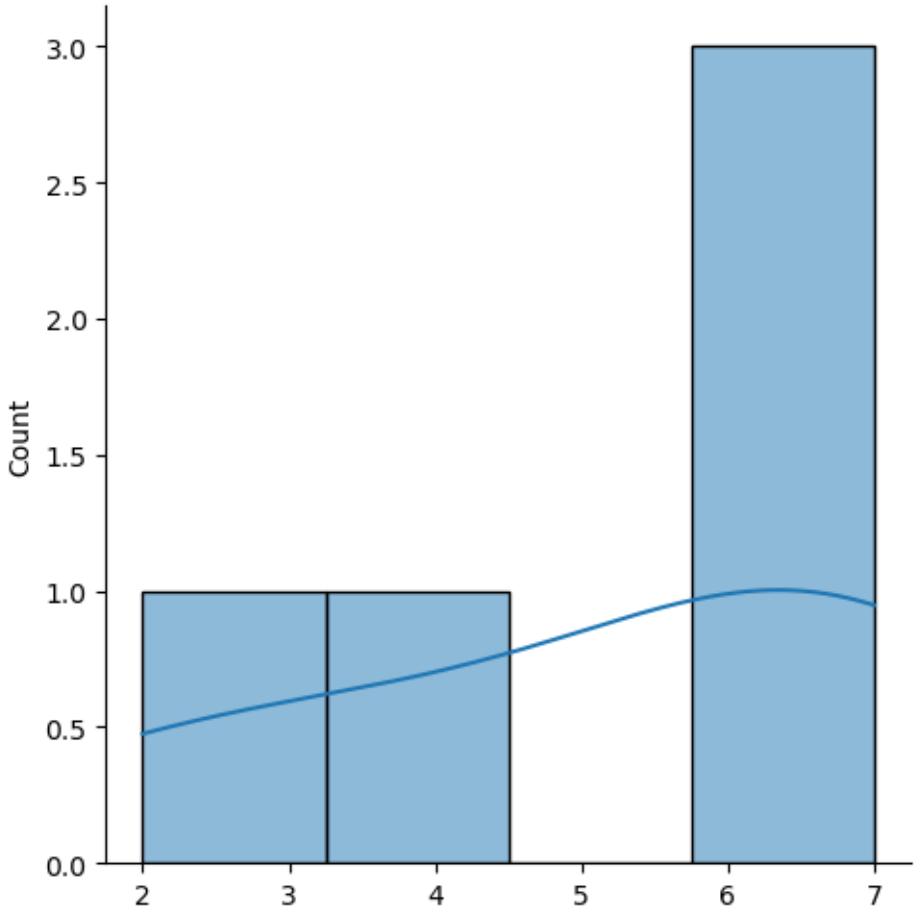
- La distribution binomiale est une distribution discrète .
- Il décrit le résultat de scénarios binaires, par exemple un tirage au sort, ce sera pile ou face.
- Il a trois paramètres :
  - n- nombre d'essais.
  - p- probabilité d'occurrence de chaque essai (par exemple pour un tirage au sort 0,5 chacun).
  - size- La forme du tableau renvoyé.

[193]:
```
x = random.binomial(n=10, p=0.5, size=5)

print(x)
sns.displot(x, kde=True)

plt.show()
```

```
[6 7 2 4 7]
```



2.8 Distribution uniforme

- Utilisé pour décrire la probabilité où chaque événement a des chances égales de se produire.
- Il a trois paramètres :
  - a- limite inférieure - par défaut 0 .0.
  - b- limite supérieure - valeur par défaut 1.0.
  - size- La forme du tableau renvoyé.

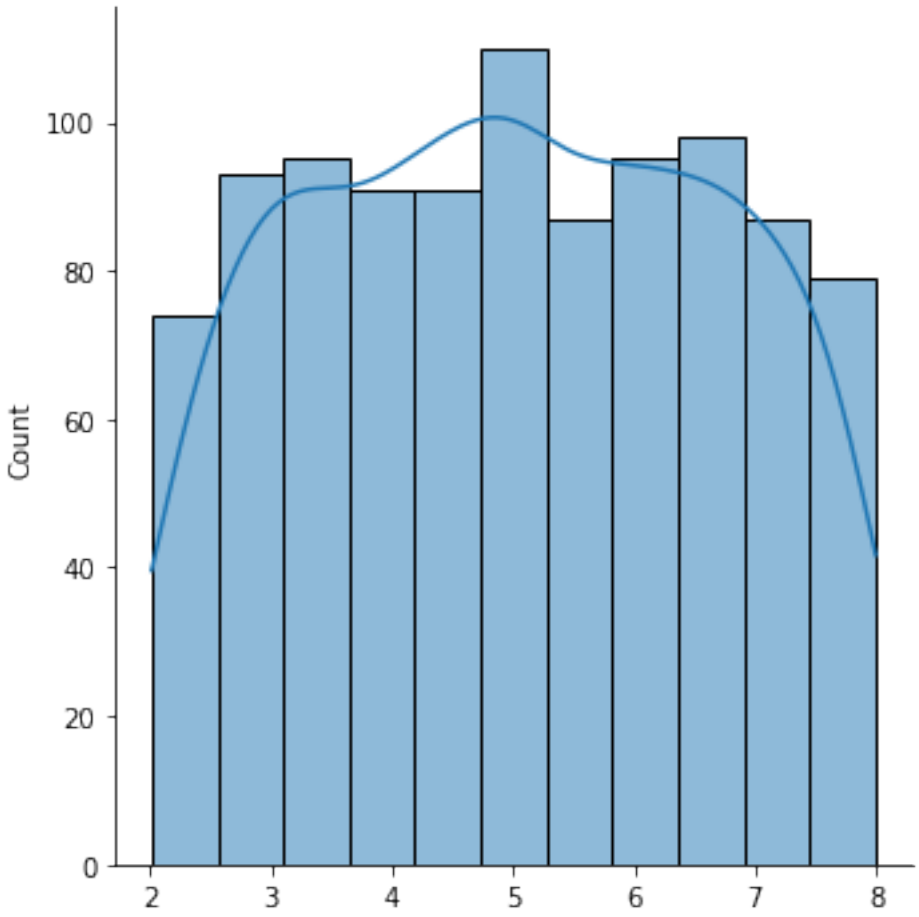
[118]:
```
x = random.uniform(4,12,size=(2, 3))

print(x)
```

```
[[ 5.79352203 11.65171176  7.83091824]
 [10.51964938  9.63851358  9.05551568]]
```

[271]:
```
sns.displot(random.uniform(2,8,size=1000), kde=True)

plt.show()
```



[194]:
```
X=np.array([1,2,3])
Y=np.array([3,4,5])
s=0
for i in range(3):
    s+=X[i]*Y[i]
s
```

[194]: 26

[195]:
```
s=X.dot(Y)
s
```

[195]: 26

[198]:
```
X=np.array([[1,2,3],[5,1,3]])
Y=np.array([[3,4],[1,2],[1,3]])
p=X.dot(Y)
p
```

[198]:
```
array([[ 8, 17],
       [19, 31]])
```

3 Introduction à Pandas

- Pandas est une bibliothèque Python, créé par Wes McKinney en 2008, utilisée pour travailler avec des ensembles de données.
- Il a des fonctions d'analyse, de nettoyage, d'exploration et de manipulation des données:
  - Corrélation entre deux colonnes ou plus ?
  - Qu'est-ce que la valeur moyenne ?
  - Valeur max?
  - Valeur mini ?
  - ...
- Le nom "Pandas" fait référence à "Panel Data" et à "Python Data Analysis".
- Pandas est open source https://github.com/pandas-dev/pandas

[1]:
```
#Installation de pandas
! pip install pandas
```

```
Requirement already satisfied: pandas in
/Users/gounane/opt/anaconda3/lib/python3.9/site-packages (1.4.2)
Requirement already satisfied: python-dateutil>=2.8.1 in
/Users/gounane/opt/anaconda3/lib/python3.9/site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/Users/gounane/opt/anaconda3/lib/python3.9/site-packages (from pandas) (2021.3)
Requirement already satisfied: numpy>=1.18.5 in
/Users/gounane/opt/anaconda3/lib/python3.9/site-packages (from pandas) (1.21.5)
Requirement already satisfied: six>=1.5 in
/Users/gounane/opt/anaconda3/lib/python3.9/site-packages (from python-
dateutil>=2.8.1->pandas) (1.16.0)
```

3.1 1. Exemple preliminaire

[1]:
```
#importation du module pandas
import pandas as pd
#version de pandas
print(pd.__version__)
```

```
2.2.2
```

[2]:
```
data = {
    'id': [1,2,3,"4e"],
    'nom': ["yahyaoui", "Moutoul", "Ait el Haj","Moufakkir"],
    'prenom':['Reda','Saad','Amina','Sara'],
    'moy': [13.25, 17.0, 12.5, 8.5]
}

df = pd.DataFrame(data)

print(df)
```

```
   id         nom prenom    moy
0   1    yahyaoui   Reda  13.25
1   2     Moutoul   Saad  17.00
2   3  Ait el Haj  Amina  12.50
3  4e   Moufakkir   Sara   8.50
```

3.2 2. Serie

Comme une colonne dans un tableau. C'est un tableau unidimensionnel contenant des données de tout type.

[4]:
```
#Création d'une serie sans spécifier les étiquettes
a = ['14', "Ahmed", 2]
sr = pd.Series(a)
print(sr)
#Label
print(sr[1])
```

```
0       14
1    Ahmed
2        2
dtype: object
Ahmed
```

3.3 2. Label (Étiquette)

Peut être utilisée pour accéder à une valeur spécifiée. Si rien d'autre n'est spécifié, les valeurs sont étiquetées avec leur numéro d'index. La première valeur a l'indice 0, la deuxième valeur a l'indice 1, etc.

[3]:
```
#Création d'une serie avec des étiquettes
a = ['14', "Ahmed", 2]
sr = pd.Series(a, index = ["x", "y", "z"])
print(sr)
#Label
print(sr["y"])
```

```
x       14
y    Ahmed
z        2
dtype: object
Ahmed
```

3.4 3. DataFrame

Une dataframe est structure de données à 2 dimensions, comme un tableau à 2 dimensions ou un tableau avec des lignes et des colonnes (tableau excel par exemple).

3.4.1 3.1. Créer un DataFrame

[11]:
```
data = {
    'id': [1,2,3,4],
    'nom': ["Yahyaoui", "Moutoul", "Ait el Haj","Moufakkir"],
    'prenom':['Reda','Saad','Amina','Sara'],
    'moy': [13.25, 17.0, 12.5, 8.5]
}
df = pd.DataFrame(data)
print(df)

df1 = pd.DataFrame(data, index=["Etudiant1","Etudiant2","Etudiant3","Etudiant4"])
print(df1)
```

```
   id         nom prenom    moy
0   1    Yahyaoui   Reda  13.25
1   2     Moutoul   Saad  17.00
2   3  Ait el Haj  Amina  12.50
3   4   Moufakkir   Sara   8.50
           id         nom prenom    moy
Etudiant1   1    Yahyaoui   Reda  13.25
Etudiant2   2     Moutoul   Saad  17.00
Etudiant3   3  Ait el Haj  Amina  12.50
Etudiant4   4   Moufakkir   Sara   8.50
```

3.4.2 3.2. Localiser une ligne

Pandas utilisent l'attribut **loc** pour renvoyer une lignes d'un DataFrame sous la forme d'une **Serie**

[16]:
```
# Une seule ligne sans index
l=df.loc[2]
```

```
print(type(l))
print(l)
# Une seule ligne avec index
l=df1.loc["Etudiant3"]
print(type(l))
print(l)
```

```
<class 'pandas.core.series.Series'>
id                 3
nom       Ait el Haj
prenom         Amina
moy             12.5
Name: 2, dtype: object
<class 'pandas.core.series.Series'>
id                 3
nom       Ait el Haj
prenom         Amina
moy             12.5
Name: Etudiant3, dtype: object
```

3.4.3 3.2. Localiser plusieurs lignes

Pandas utilisent l'attribut **loc** pour renvoyer plusieurs lignes d'un DataFrame sous la forme d'un **DataFrame**

[21]:
```
#Plusieurs Lignes sans index
l=df.loc[[0,3]]
print(type(l))
print(l)
#Plusieurs Lignes avec index
l=df1.loc[["Etudiant1","Etudiant3"]]
print(type(l))
print(l)

ll=df.loc[range(1,3)]
print(ll)
```

```
<class 'pandas.core.frame.DataFrame'>
   id        nom prenom    moy
0   1   Yahyaoui   Reda  13.25
3   4  Moufakkir   Sara   8.50
<class 'pandas.core.frame.DataFrame'>
           id         nom prenom    moy
Etudiant1   1    Yahyaoui   Reda  13.25
Etudiant3   3  Ait el Haj  Amina  12.50
   id         nom prenom   moy
1   2     Moutoul   Saad  17.0
2   3  Ait el Haj  Amina  12.5
```

[5]:
```
#Plusieurs Lignes
l=df.loc[[0,2]]
print(type(l))
print(l)
```

```
<class 'pandas.core.frame.DataFrame'>
  id         nom prenom    moy
0  1    yahyaoui   Reda  13.25
2  3  Ait el Haj  Amina  12.50
```

[7]:
```
l=df.loc[:,["nom"]]
print(l)
s=df.loc[0]
print(type(s))
print(s.index[0:2])
print(df.loc[:,df.columns[1:]])
```

```
          nom
0    yahyaoui
1     Moutoul
2  Ait el Haj
3   Moufakkir
<class 'pandas.core.series.Series'>
Index(['id', 'nom'], dtype='object')
          nom prenom    moy
0    yahyaoui   Reda  13.25
1     Moutoul   Saad  17.00
2  Ait el Haj  Amina  12.50
3   Moufakkir   Sara   8.50
```

3.4.4 3.4. DataFrame et Fichiers

3.4.1. Enregistrer un dataframe dans un fichier

[62]:
```
df.to_csv('myDataset.csv')
```

3.4.2. Charger un fichier dans un dataframe

[39]:
```
df=pd.read_csv("home_loan.csv")
```

3.5 4. Analyser un Dataframe

3.5.1 4.1 head

La méthode head() renvoie les en-têtes et un nombre spécifié de lignes, en commençant par le haut.

[11]:
```
df=pd.read_csv("archive/data.csv")
df.head()
```

[11]:
```
                  date      price  bedrooms  bathrooms  sqft_living  sqft_lot  \
0  2014-05-02 00:00:00   313000.0       3.0       1.50         1340      7912
```

```
1  2014-05-02 00:00:00  2384000.0       5.0       2.50         3650      9050
2  2014-05-02 00:00:00   342000.0       3.0       2.00         1930     11947
3  2014-05-02 00:00:00   420000.0       3.0       2.25         2000      8030
4  2014-05-02 00:00:00   550000.0       4.0       2.50         1940     10500

   floors  waterfront  view  condition  sqft_above  sqft_basement  yr_built  \
0     1.5           0     0          3        1340              0      1955
1     2.0           0     4          5        3370            280      1921
2     1.0           0     0          4        1930              0      1966
3     1.0           0     0          4        1000           1000      1963
4     1.0           0     0          4        1140            800      1976

   yr_renovated                    street       city  statezip country
0          2005      18810 Densmore Ave N  Shoreline  WA 98133     USA
1             0           709 W Blaine St    Seattle  WA 98119     USA
2             0  26206-26214 143rd Ave SE       Kent  WA 98042     USA
3             0           857 170th Pl NE   Bellevue  WA 98008     USA
4          1992         9105 170th Ave NE    Redmond  WA 98052     USA
```

[47]:
```
df.head(3)
```

[47]:
```
    Loan_ID Gender Married Dependents Education Self_Employed  \
0  LP001002   Male      No          4  Graduate            No
1  LP001003   Male     Yes          1  Graduate            No
2  LP001005   Male     Yes          0  Graduate           Yes

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             5849                0.0         NaN             360.0
1             4583             1508.0       128.0             360.0
2             3000                0.0        66.0             360.0

   Credit_History Property_Area Loan_Status
0             1.0         Urban           Y
1             1.0         Rural           N
2             1.0         Urban           Y
```

3.5.2 4.2 tail

La méthode tail() renvoie les en-têtes et un nombre spécifié de lignes, en commençant par le bas.

[74]:
```
df.tail()
```

[74]:
```
     PassengerId  Survived  Pclass                          Name     Sex  \
413         1305         0       3            Spector, Mr. Woolf    male
414         1306         1       1  Oliva y Ocana, Dona. Fermina  female
415         1307         0       3  Saether, Mr. Simon Sivertsen    male
416         1308         0       3           Ware, Mr. Frederick    male
417         1309         0       3      Peter, Master. Michael J    male
```

```
      Age  SibSp  Parch             Ticket      Fare Cabin Embarked
413   NaN      0      0          A.5. 3236    8.0500   NaN        S
414  39.0      0      0           PC 17758  108.9000  C105        C
415  38.5      0      0  SOTON/O.Q. 3101262    7.2500   NaN        S
416   NaN      0      0             359309    8.0500   NaN        S
417   NaN      1      1               2668   22.3583   NaN        C
```

[48]:
```
df.tail(4)
```

[48]:
```
      Loan_ID  Gender Married Dependents Education Self_Employed  \
610  LP002979    Male     Yes         3+  Graduate            No
611  LP002983    Male     Yes          1  Graduate            No
612  LP002984    Male     Yes          2  Graduate            No
613  LP002990  Female      No          0  Graduate           Yes

     ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
610             4106                0.0        40.0             180.0
611             8072              240.0       253.0             360.0
612             7583                0.0       187.0             360.0
613             4583                0.0       133.0             360.0

     Credit_History Property_Area Loan_Status
610             1.0         Rural           Y
611             1.0         Urban           Y
612             1.0         Urban           Y
613             0.0     Semiurban           N
```

3.5.3 4.2 info

La méthode info() donne plus d'informations sur l'ensemble de données.

[65]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            614 non-null    object
 1   Gender             601 non-null    object
 2   Married            611 non-null    object
 3   Dependents         599 non-null    object
 4   Education          614 non-null    object
 5   Self_Employed      582 non-null    object
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
```

```
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object
 12  Loan_Status        614 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

3.5.4 4.4 describe

Cette méthode renvoie quelques statistiques

[66]:
```
df.describe()
```

[66]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	\
count	614.000000	614.000000	592.000000	600.00000	
mean	5403.459283	1621.245798	146.412162	342.00000	
std	6109.041673	2926.248369	85.587325	65.12041	
min	150.000000	0.000000	9.000000	12.00000	
25%	2877.500000	0.000000	100.000000	360.00000	
50%	3812.500000	1188.500000	128.000000	360.00000	
75%	5795.000000	2297.250000	168.000000	360.00000	
max	81000.000000	41667.000000	700.000000	480.00000	

	Credit_History
count	564.000000
mean	0.842199
std	0.364878
min	0.000000
25%	1.000000
50%	1.000000
75%	1.000000
max	1.000000

3.6 5. Nettoyage des données

Le nettoyage des données consiste à réparer les mauvaises dans le dataset: 1. Cellules vides (null) 2. Données au mauvais format 3. Données erronées 4. Doublons

3.6.1 5.1 Nettoyage des cellules vides (null)

Les cellules vides peuvent donner un résultat erroné lors de l'analysez des données. On peut traiter ces cellules par: 1. suppression des lignes contenant des cellules vides. 2. remplacer les valeurs vides.

5.1.1 Nettoyage par suppression Généralement, on peut supprimer les lignes contenant des cellules vides, car les datasets peuvent être très volumineux et la suppression de quelques lignes n'aura pas un grand impact sur le résultat. Pour supprimer ces lignes, on utilise la méthode **dropna()** du DataFrame

[14]:
```
df=pd.read_csv("archive/data.csv")
df.info()
print()
new_df=df.dropna(inplace=False)
#Pour modifier directement le dataframe
#df.dropna(inplace=True)
new_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4600 entries, 0 to 4599
Data columns (total 18 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   date           4600 non-null   object
 1   price          4600 non-null   float64
 2   bedrooms       4600 non-null   float64
 3   bathrooms      4600 non-null   float64
 4   sqft_living    4600 non-null   int64
 5   sqft_lot       4600 non-null   int64
 6   floors         4600 non-null   float64
 7   waterfront     4600 non-null   int64
 8   view           4600 non-null   int64
 9   condition      4600 non-null   int64
 10  sqft_above     4600 non-null   int64
 11  sqft_basement  4600 non-null   int64
 12  yr_built       4600 non-null   int64
 13  yr_renovated   4600 non-null   int64
 14  street         4600 non-null   object
 15  city           4600 non-null   object
 16  statezip       4600 non-null   object
 17  country        4600 non-null   object
dtypes: float64(4), int64(9), object(5)
memory usage: 647.0+ KB

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4600 entries, 0 to 4599
Data columns (total 18 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   date           4600 non-null   object
 1   price          4600 non-null   float64
 2   bedrooms       4600 non-null   float64
 3   bathrooms      4600 non-null   float64
 4   sqft_living    4600 non-null   int64
 5   sqft_lot       4600 non-null   int64
 6   floors         4600 non-null   float64
 7   waterfront     4600 non-null   int64
 8   view           4600 non-null   int64
```

```
 9   condition      4600 non-null   int64
 10  sqft_above     4600 non-null   int64
 11  sqft_basement  4600 non-null   int64
 12  yr_built       4600 non-null   int64
 13  yr_renovated   4600 non-null   int64
 14  street         4600 non-null   object
 15  city           4600 non-null   object
 16  statezip       4600 non-null   object
 17  country        4600 non-null   object
dtypes: float64(4), int64(9), object(5)
memory usage: 647.0+ KB
```

[60]:
```
df.head()
```

[60]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	\
1	LP001003	Male	Yes	1	Graduate	No	
2	LP001005	Male	Yes	0	Graduate	Yes	
3	LP001006	Male	Yes	0	Not Graduate	No	
4	LP001008	Male	No	0	Graduate	No	
5	LP001011	Male	Yes	2	Graduate	Yes	

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	\
1	4583	1508.0	128.0	360.0	
2	3000	0.0	66.0	360.0	
3	2583	2358.0	120.0	360.0	
4	6000	0.0	141.0	360.0	
5	5417	4196.0	267.0	360.0	

	Credit_History	Property_Area	Loan_Status
1	1.0	Rural	N
2	1.0	Urban	Y
3	1.0	Urban	Y
4	1.0	Urban	Y
5	1.0	Urban	Y

3.6.2 5.1.2 Netoyage par remplacement des valeurs vides.

Au lieu de supprimer des ligne entière simplement à cause des cellules vides. La méthode **fillna()** permet de remplacer les cellules vides par: 1. une valeur 2. la valeur moyenne de la colonne 3. la médiane de la colonne 4. la valeur la plus frequente (mode) de la colonne #### 5.1.2.1 remplacement par une valeur

[67]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
```

```
 0   Loan_ID            614 non-null    object
 1   Gender             601 non-null    object
 2   Married            611 non-null    object
 3   Dependents         599 non-null    object
 4   Education          614 non-null    object
 5   Self_Employed      582 non-null    object
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object
 12  Loan_Status        614 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

[78]:
```
la=df.loc[:,"LoanAmount"]
type(la)
lan=la[la.isnull()]
df.loc[lan.index,"LoanAmount"]=0
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            614 non-null    object
 1   Gender             601 non-null    object
 2   Married            611 non-null    object
 3   Dependents         599 non-null    object
 4   Education          614 non-null    object
 5   Self_Employed      582 non-null    object
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         614 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object
 12  Loan_Status        614 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

[84]:
```
df=pd.read_csv("home_loan.csv")
df["LoanAmount"].info()
print()
df["LoanAmount"].fillna(20, inplace=True)
df["LoanAmount"].info()
```

```
type(df["LoanAmount"])
print(pd.__version__)
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 614 entries, 0 to 613
Series name: LoanAmount
Non-Null Count  Dtype
--------------  -----
592 non-null    float64
dtypes: float64(1)
memory usage: 4.9 KB

<class 'pandas.core.series.Series'>
RangeIndex: 614 entries, 0 to 613
Series name: LoanAmount
Non-Null Count  Dtype
--------------  -----
614 non-null    float64
dtypes: float64(1)
memory usage: 4.9 KB
1.4.2
```

[93]:
```
import numpy as np
df=pd.read_csv("home_loan.csv")
lam=df["LoanAmount"]
lam.info()
print()
moy=np.round(lam.mean(),2)
print(moy)
df["LoanAmount"].fillna(moy,inplace=True)
df.info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 614 entries, 0 to 613
Series name: LoanAmount
Non-Null Count  Dtype
--------------  -----
592 non-null    float64
dtypes: float64(1)
memory usage: 4.9 KB

146.41
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            614 non-null    object
```

```
 1   Gender             601 non-null    object
 2   Married            611 non-null    object
 3   Dependents         599 non-null    object
 4   Education          614 non-null    object
 5   Self_Employed      582 non-null    object
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         614 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object
 12  Loan_Status        614 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

[122]:
```
##Cherche la mediane de LoanAmount

ndf=df.dropna()
ndf.sort_values("LoanAmount",inplace=True)
n=ndf["LoanAmount"].count()
print(ndf.loc[(int)(n+1)/2,"LoanAmount"])
#print(ndf.loc[df["LoanAmount"].count()/2-1:ndf["LoanAmount"].count()/
 ↪2,"LoanAmount"].mean())
ndf["LoanAmount"].median()
```

```
201.0

/var/folders/sf/961f5b2128ncfymk_8mq35v40000gn/T/ipykernel_64180/380842513.py:4:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  ndf.sort_values("LoanAmount",inplace=True)
```

[122]:
```
130.0
```

5.1.2.2 remplacement par la valeur moyenne de la colonne

[98]:
```
df.info()
m = df["l"].mean()
print(m)
#m = df["LoanAmount"].median()
new_df=df["LoanAmount"].fillna(m)
new_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
```

```
---  ------       --------------  -----
 0   PassengerId  418 non-null    int64
 1   Survived     418 non-null    int64
 2   Pclass       418 non-null    int64
 3   Name         418 non-null    object
 4   Sex          418 non-null    object
 5   Age          418 non-null    float64
 6   SibSp        418 non-null    int64
 7   Parch        418 non-null    int64
 8   Ticket       418 non-null    object
 9   Fare         417 non-null    float64
 10  Cabin        418 non-null    object
 11  Embarked     418 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 39.3+ KB
35.6271884892086
<class 'pandas.core.series.Series'>
RangeIndex: 418 entries, 0 to 417
Series name: Fare
Non-Null Count  Dtype
--------------  -----
418 non-null    float64
dtypes: float64(1)
memory usage: 3.4 KB
```

5.1.2.3 remplacement par la mediane de la colonne

[132]:
```
df.median()["LoanAmount"]
```

```
/var/folders/sf/961f5b2128ncfymk_8mq35v40000gn/T/ipykernel_64180/2520599487.py:1
: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with
'numeric_only=None') is deprecated; in a future version this will raise
TypeError.  Select only valid columns before calling the reduction.
  df.median()["LoanAmount"]
```

[132]:
```
129.0
```

[127]:
```
df.describe()
m = df["LoanAmount"].median()# numeric_only=True
print(m)
new_df=df["LoanAmount"].fillna(m)
new_df.info()
```

```
129.0
<class 'pandas.core.series.Series'>
Int64Index: 614 entries, 568 to 171
Series name: LoanAmount
Non-Null Count  Dtype
--------------  -----
```

```
614 non-null    float64
dtypes: float64(1)
memory usage: 9.6 KB
```

5.1.2.3 remplacement par le mode de la colonne La méthode **mode()** renvoie les valeurs frequentes dans un DataFrame/Serie sous la forme d'un DataFrame/Serie

[153]:
```
df=pd.read_csv("home_loan.csv")
m = df["LoanAmount"].mode()
print(type(m))
print(m)
new_df=df["LoanAmount"].fillna(m[0])
print(m[0])
```

```
<class 'pandas.core.series.Series'>
0    120.0
Name: LoanAmount, dtype: float64
120.0
```

3.7 5.2 Nétoyage des données au mauvais format

Les cellules contenant des données au mauvais format peuvent rendre difficile, l'analyse des données.il faut alors soit: 1. supprimer les lignes 2. convertir toutes les cellules des colonnes dans le même format.

Pour mieux adapter au contenu, on utilise la méthode **convert_dtypes()** :

[154]:
```
df=pd.read_csv("home_loan.csv")
df.info()
df.convert_dtypes()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            614 non-null    object
 1   Gender             601 non-null    object
 2   Married            611 non-null    object
 3   Dependents         599 non-null    object
 4   Education          614 non-null    object
 5   Self_Employed      582 non-null    object
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object
```

```
 12  Loan_Status        614 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            614 non-null    object
 1   Gender             601 non-null    object
 2   Married            611 non-null    object
 3   Dependents         599 non-null    object
 4   Education          614 non-null    object
 5   Self_Employed      582 non-null    object
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object
 12  Loan_Status        614 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

[16]:
```
data = {
    'id': [1,2,3,9],
    'nom': ["yahyaoui", "Moutoul", "Ait el Haj","Moufakkir"],
    'prenom':['Reda','Saad','Amina','Sara'],
    'moy': [13.25, 17.0, 12.5, 8.5]
}

df = pd.DataFrame(data)
df.info()
df.convert_dtypes()
df.info()
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   id      4 non-null      int64
 1   nom     4 non-null      object
 2   prenom  4 non-null      object
 3   moy     4 non-null      float64
dtypes: float64(1), int64(1), object(2)
memory usage: 260.0+ bytes
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   id      4 non-null      int64
 1   nom     4 non-null      object
 2   prenom  4 non-null      object
 3   moy     4 non-null      float64
dtypes: float64(1), int64(1), object(2)
memory usage: 260.0+ bytes
```

[16]:

	id	nom	prenom	moy
0	1	yahyaoui	Reda	13.25
1	2	Moutoul	Saad	17.00
2	3	Ait el Haj	Amina	12.50
3	9	Moufakkir	Sara	8.50

On peut aussi caster les types de données vers un autre type de notre choix avec la méthode **astype()**

[17]:
```
d = {'col1': [1, 2], 'col2': [3, 4]}
df = pd.DataFrame(data=d)
df.dtypes
```

[17]:
```
col1    int64
col2    int64
dtype: object
```

[18]:
```
#Series.astype(dtype, copy=True, errors='raise') #errors='ignore'
df.astype('int32').dtypes
#df.astype({'col1': 'int32'}).dtypes
```

[18]:
```
col1    int32
col2    int32
dtype: object
```

Après un changement de type, si une donnée ne peut pas être converti au type souhaité, elle prend la valeur NaT, qui peut être traitée comme une valeur NULL, et nous pouvons supprimer la ligne en utilisant la dropna()méthode

[25]:
```
d = {'col1': [1, 2, 6], 'col2': [3, 4, 4]}
df = pd.DataFrame(data=d)
print(df.dtypes)
df=df.astype('int32',errors='ignore')
print(df.dtypes)
df.index
```

```
col1    int64
col2    int64
dtype: object
```

```
col1    int32
col2    int32
dtype: object
```

[25]:
```
RangeIndex(start=0, stop=3, step=1)
```

3.8 5.3 Nétoyage des données erronées

[169]:
```
df=pd.read_csv("data.csv")
for i in df[df["Duration"]>200].index:
    df.loc[i,"Duration"]=200
df.describe()
```

[169]:

	Duration	Pulse	Maxpulse	Calories
count	169.000000	169.000000	169.000000	164.000000
mean	62.721893	107.461538	134.047337	375.790244
std	37.284933	14.510259	16.450434	266.379919
min	15.000000	80.000000	100.000000	50.300000
25%	45.000000	100.000000	124.000000	250.925000
50%	60.000000	105.000000	131.000000	318.600000
75%	60.000000	111.000000	141.000000	387.600000
max	200.000000	159.000000	184.000000	1860.400000

[170]:
```
df=pd.read_csv("data.csv")
for i in df[df["Duration"]>200].index:
    df.drop(i,inplace=True)

df[df["Duration"]>200]
```

[170]:
```
Empty DataFrame
Columns: [Duration, Pulse, Maxpulse, Calories]
Index: []
```

3.9 5.4 Suppression des doublons

[160]:
```
df=pd.read_csv("data.csv")

print(~df.duplicated())
```

```
0      True
1      True
2      True
3      True
4      True
       ...
164    True
165    True
166    True
```

```
167    True
168    True
Length: 169, dtype: bool
```

[161]:
```
print(df[~df.duplicated()])
```

```
     Duration  Pulse  Maxpulse  Calories
0          60    110       130     409.1
1          60    117       145     479.0
2          60    103       135     340.0
3          45    109       175     282.4
4          45    117       148     406.0
..        ...    ...       ...       ...
164        60    105       140     290.8
165        60    110       145     300.0
166        60    115       145     310.2
167        75    120       150     320.4
168        75    125       150     330.4

[162 rows x 4 columns]
```

[178]:
```
df.drop_duplicates(inplace = True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 162 entries, 0 to 168
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Duration  162 non-null    int64
 1   Pulse     162 non-null    int64
 2   Maxpulse  162 non-null    int64
 3   Calories  157 non-null    float64
dtypes: float64(1), int64(3)
memory usage: 6.3 KB
```

3.10 6. La correlation

La correlation consiste à déterminer l'absence ou la présence d'une relation linéaire significative entre les variables.

[179]:
```
df=pd.read_csv("data.csv")
df.corr()
```

[179]:

	Duration	Pulse	Maxpulse	Calories
Duration	1.000000	-0.155408	0.009403	0.922717
Pulse	-0.155408	1.000000	0.786535	0.025121
Maxpulse	0.009403	0.786535	1.000000	0.203813
Calories	0.922717	0.025121	0.203813	1.000000

[ ]: