# Introduction à Pandas

Pandas est une librairie python qui permet de manipuler facilement des données que l'on souhaite analyser. Elle considère trois types de structures :

- les séries : un tableau à une dimension où les données sont de même type
- les dataframes : un tableau à deux dimensions où les données peuvent être de types différents
- les panels : un tableau à trois dimensions où les données peuvent être de types différents

Le **dataframe** est le plus utilisé dans pandas car il permet de pouvoir manipuler des tableaux avec les noms des colonnes ou des lignes, offre de nombreuses fonctionalités similaires à celles de système de gestion de base de données (séléction, group-by, etc), offre des facilités pour pouvoir sauvegarder ou afficher des résultats.

Les différents types pandas :

Etant donné qu'il y a trois structure de données manipulables avec Pandas il va exister différentes manières de les indexer.

Les séries ont un seul dimension appelé index (axis ==0)
Les dataframes ont deux axes l'axe *index* (axis == 0), and l'axe des *colonnes* (axis == 1). Ils peuvent être vus comme des dictionnaires Python où la clé correspond aux noms des colonnes et la valeurs aux séries des colonnes.
Les panels peuvent être vus comme des dictionnaires Python de dataframes. Ils ont donc des *items* ou *index* (axis == 0), des *axes majeurs* (axis == 1) et des *axes mineurs*(axis == 2).

| dtype Pandas | type Python | type NumPy | Utilisation |
|---|---|---|---|
| object | str | string_, unicode_ | Texte |
| int64 | int | int_, int8, int16, int32, int64, uint8, uint16, uint32, uint64 | Integer |
| float64 | float | float_, float16, float32, float64 | Float |
| bool | bool | bool_ | True/False |
| datetime64 | NA | datetime64 | Date et Heure |
| timedelta | NA | NA | Différence entre deux datetime |
| category | NA | NA | Liste finie de valeurs textuelles |

Remarque : un dataframe peut être affiché par print ou par display.

# Petit rappel sur les tableaux en python

Un tableau peut être créé à l'aide de la fonction *np.array()*

In [1]:

```
import numpy as np
print ("Création d'un tableau à 1 dimension ")
tableau_une_dimension = np.array([5, 4, 3, 2, 1])
print (tableau_une_dimension)

print ("\nCréation d'un tableau à 1 dimension dont le type est float")
tableau_une_dimension_float = np.array([5, 4, 3, 2, 1],dtype='float')
print (tableau_une_dimension_float)

print ("\nCréation d'un tableau à 2 dimensions (2 lignes, 5 colonnes)")
tableau_deux_dimensions = np.array([[5, 4, 3, 2, 1],
                                    [1, 2, 3, 4, 5]])

print (tableau_deux_dimensions)


```

```
Création d'un tableau à 1 dimension
[5 4 3 2 1]

Création d'un tableau à 1 dimension dont le type est float
[5. 4. 3. 2. 1.]

Création d'un tableau à 2 dimensions (2 lignes, 5 colonnes)
[[5 4 3 2 1]
 [1 2 3 4 5]]
```

### Autres types d'initialisation

In [2]:

```
print ("\nCréation d'un tableau à 2 dimensions initialisé à 0 et de type floa
tableau_deux_dimensions_zero_float = np.zeros((2, 3), dtype='f')
print (tableau_deux_dimensions_zero_float)

print ("\nCréation d'un tableau à 2 dimensions initialisé à 1 et de type inte
tableau_deux_dimensions_un_entier = np.ones((3, 5), dtype='i')
print (tableau_deux_dimensions_un_entier)

print ("\nCréation d'un tableau à 1 dimension initialisé avec des valeurs rég
tableau_une_dimension_arrange= np.arange(10)
print (tableau_une_dimension_arrange)

print ("\nCréation d'un tableau à 1 dimension initialisé de 1 à 3 avec un esp
tableau_une_dimension_arrange2= np.arange(1,3,0.5)
print (tableau_une_dimension_arrange2)

print ("\nCréation d'un tableau à 2 dimensions initialisé avec arange ")
line_0=np.arange(1,3,0.5)
line_1=np.arange(3,1,-0.5)
tableau_deux_dimensions_arrange = np.array([line_0,
                                    line_1])
print (tableau_deux_dimensions_arrange)

print ("\nCréation d'un tableau à 2 dimensions initialisé aléatoirement suiva
tableau_deux_dimensions_aleatoire = np.random.normal(0, 1, (4, 4))
print (tableau_deux_dimensions_arrange)
```

```
Création d'un tableau à 2 dimensions initialisé à 0 et de type float
[[0. 0. 0.]
 [0. 0. 0.]]

Création d'un tableau à 2 dimensions initialisé à 1 et de type integer
[[1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]]

Création d'un tableau à 1 dimension initialisé avec des valeurs réguli
èrement espacées
[0 1 2 3 4 5 6 7 8 9]

Création d'un tableau à 1 dimension initialisé de 1 à 3 avec un espace
de 0.5
[1.  1.5 2.  2.5]

Création d'un tableau à 2 dimensions initialisé avec arange
[[1.  1.5 2.  2.5]
 [3.  2.5 2.  1.5]]

Création d'un tableau à 2 dimensions initialisé aléatoirement suivant
une loi normale centrée sur 0 avec une dispersion de 1
[[1.  1.5 2.  2.5]
 [3.  2.5 2.  1.5]]
```

### Copie d'un tableau

In [3]:

```
tableau_une_dimension = np.array([5, 4, 3, 2, 1])
print ("Tableau initial\n",tableau_une_dimension)
copie_directe=tableau_une_dimension
print ("\nCopie directe\n",copie_directe)
print ("\nAttention une copie directe est un pointeur vers le tableau initial
print ("\nModification d'une valeur de copie directe - copie_directe[0]=1")
copie_directe[0]=1
print ("\nTableau initial\n",tableau_une_dimension)

print ("\nUtilisation de la fonction .copy")
copie_copy=tableau_une_dimension.copy()
print ("\nCopie par .copy\n",copie_copy)
print ("\nModification d'une valeur de copie par copy - copie_copy[0]=5")
copie_copy[0]=5
print ("\nIl n'y a pas de modification dans tableau initial\n",tableau_une_di
```

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

Copie directe
 [5 4 3 2 1]

Attention une copie directe est un pointeur vers le tableau initial

Modification d'une valeur de copie directe - copie_directe[0]=1

Tableau initial
 [1 4 3 2 1]

Utilisation de la fonction .copy

Copie par .copy
 [1 4 3 2 1]

Modification d'une valeur de copie par copy - copie_copy[0]=5

Il n'y a pas de modification dans tableau initial
 [1 4 3 2 1]
```

### Shape, size, len et accès aux valeurs

*shape* permet de connaître les dimensions d'un tableau. Il est possibe d'accéder aux valeurs en utilisant [nb]. Attention les index des tableaux commencent à 0.

In [4]:

```
1
2  tableau_une_dimension = np.array([5, 4, 3, 2, 1])
3  print ("Tableau à 1 dimension \n",tableau_une_dimension)
4  print ("\nLes dimensions du tableau : ",tableau_une_dimension.shape)
5  print ("\nAccès au 3 ième élément du tableau : ", tableau_une_dimension[2])
6  print ("\nAccès au dernier élément du tableau avec -1 : ", tableau_une_dimens
7
8  tableau_deux_dimensions = np.array([[5, 4, 3, 2, 1],
9                                      [1, 2, 3, 4, 5]])
10
11 print ("\nTableau à 2 dimensions \n",tableau_deux_dimensions)
12 print ("\nLes dimensions du tableau ",tableau_deux_dimensions.shape)
13 print ("\nshape[0] correspond au nombre de lignes", tableau_deux_dimensions.s
14 print ("\nshape[1] correspond au nombre de colonnes", tableau_deux_dimensions
15
16 print ("\nValeur pour la 2 ligne et colonne 3 : ",tableau_deux_dimensions[1,2
17 print ("\nValeur de la seconde colonne, laisser la partie après la virgule vi
18 print ("\nAttention size et différent de len pour un tableau 2D")
19 print ("\nsize retourne le nombre d'éléments du tableau : ",np.size(tableau_d
20 print ("\nlen retourne combien il y a de lignes dans le tableau : ",len(table
```

```
Tableau à 1 dimension
 [5 4 3 2 1]

Les dimensions du tableau :  (5,)

Accès au 3 ième élément du tableau :  3

Accès au dernier élément du tableau avec -1 :  1

Tableau à 2 dimensions
 [[5 4 3 2 1]
 [1 2 3 4 5]]

Les dimensions du tableau  (2, 5)

shape[0] correspond au nombre de lignes 2

shape[1] correspond au nombre de colonnes 5

Valeur pour la 2 ligne et colonne 3 :  3

Valeur de la seconde colonne, laisser la partie après la virgule vide
:  [1 2 3 4 5]

Attention size et différent de len pour un tableau 2D

size retourne le nombre d'éléments du tableau :  10

len retourne combien il y a de lignes dans le tableau :  2
```

### Slicing

Il est possible d'utiliser des techniques de "tranchage" (*slicing*) pour obtenir des parties de tableaux.
Elle consiste à indiquer entre crochets des indices pour définir le début et la fin de la tranche (non comprise) et le pas éventuel et à les séparer par deux-points ':'. Syntaxe : [début:fin:pas].

Le principe est le même pour un tableau à deux dimensions, il suffit de séparer par des , les parties.

In [5]:

```
tableau_une_dimension = np.array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
print ("Tableau à 1 dimension\n",tableau_une_dimension)
print ("\nObtention des trois premiers éléments du tableau :",tableau_une_dim
print ("\nObtention des trois premiers éléments du tableau sans spécifier à g
print ("\nObtention des derniers éléments après le troisième sans spécifier à
print ("\nObtention des éléments du premier au dernier non compris avec un pa

print ()
tableau_deux_dimensions = np.array([[9, 8, 7, 6, 5, 4, 3, 2, 1, 0],
                                    [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
                                    [9, 8, 7, 6, 5, 4, 3, 2, 1, 0],
                                    [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]])
print ("Tableau à 2 dimensions\n",tableau_deux_dimensions)
print ("\nValeur pour la 2 ligne et colonne 3 :",tableau_deux_dimensions[1,2]
print ("\nTableau qui va de la ligne 1 à 3 et des colonnes 0 à 5 :\n",tableau
print ("\nTableau avec un saut de 2 dans les valeurs :\n",tableau_deux_dimens
```

```
Tableau à 1 dimension
 [9 8 7 6 5 4 3 2 1 0]

Obtention des trois premiers éléments du tableau : [9 8 7]

Obtention des trois premiers éléments du tableau sans spécifier à gauc
he des ':' : [9 8 7]

Obtention des derniers éléments après le troisième sans spécifier à dr
oite des ':' : [6 5 4 3 2 1 0]

Obtention des éléments du premier au dernier non compris avec un pas d
e 3 : [8 5 2]

Tableau à 2 dimensions
 [[9 8 7 6 5 4 3 2 1 0]
 [0 1 2 3 4 5 6 7 8 9]
 [9 8 7 6 5 4 3 2 1 0]
 [0 1 2 3 4 5 6 7 8 9]]

Valeur pour la 2 ligne et colonne 3 : 2

Tableau qui va de la ligne 1 à 3 et des colonnes 0 à 5 :
 [[0 1 2 3 4]
 [9 8 7 6 5]
 [0 1 2 3 4]]

Tableau avec un saut de 2 dans les valeurs :
 [[9 7 5 3 1]
 [9 7 5 3 1]]
```

### Redimensionnement d'un tableau

Il est parfois très utile de pouvoir redimensionner un tableau (certaines fonctions scikit learn nécessite d'avoir une vision en deux dimensions pour un tableau en une dimension). Il faut utiliser la fonction *reshape*. Syntaxe : namearray.reshape((un tuple de valeurs))

In [6]:

```
tableau_une_dimension = np.array([5, 4, 3, 2, 1, 0])
print ("Transformation 1 dimension en 2 dimensions")
print ("\nTableau initial ",tableau_une_dimension)
print ("\nShape du tableau", tableau_une_dimension.shape)
tableau_reshape_2D=tableau_une_dimension.reshape((tableau_une_dimension.shape
print ("\nTableau transformé en 2 dimensions \n",tableau_reshape_2D)
print ("\nShape du tableau", tableau_reshape_2D.shape)

print ("\nTransformation 2 dimensions en 3 dimensions")
tableau_deux_dimensions = np.array([[5, 4, 3, 2, 1],
                                    [1, 2, 3, 4, 5]])
print ("\nTableau initial \n",tableau_deux_dimensions)
print ("\nShape du tableau", tableau_deux_dimensions.shape)
tableau_reshape_3D=tableau_deux_dimensions.reshape((tableau_deux_dimensions.s
print ("\nTableau transformé en 3 dimensions \n",tableau_reshape_3D)
print ("\nShape du tableau", tableau_reshape_3D.shape)
```

```
Transformation 1 dimension en 2 dimensions

Tableau initial  [5 4 3 2 1 0]

Shape du tableau (6,)

Tableau transformé en 2 dimensions
 [[5]
 [4]
 [3]
 [2]
 [1]
 [0]]

Shape du tableau (6, 1)

Transformation 2 dimensions en 3 dimensions

Tableau initial
 [[5 4 3 2 1]
 [1 2 3 4 5]]

Shape du tableau (2, 5)

Tableau transformé en 3 dimensions
 [[[5]
  [4]
  [3]
  [2]
  [1]]

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

Shape du tableau (2, 5, 1)
```

### Transposition

Il est possible de faire une transposition en utilisation la fonction *.transpose()* ou *.T*

In [7]:

```
tableau_deux_dimensions = np.array([[5, 4, 3, 2, 1],
                                    [1, 2, 3, 4, 5]])
print ("\nTableau initial \n",tableau_deux_dimensions)
print ("\nTransposition avec .transpose()",tableau_deux_dimensions.transpose(
print ("\nTransposition avec .T",tableau_deux_dimensions.T)
```

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

Transposition avec .transpose() [[5 1]
 [4 2]
 [3 3]
 [2 4]
 [1 5]]

Transposition avec .T [[5 1]
 [4 2]
 [3 3]
 [2 4]
 [1 5]]
```

**Produit scalaire**

Le produit scalaire de deux tableaux (matrices) peut se faire via la fonction *.dot* appliquée à une matrice ou par la fonction *np.dot(matrice1,matrice2)*

In [8]:

```
tableau_deux_dimensions = np.array([[5, 4, 3, 2, 1],
                                    [1, 2, 3, 4, 5]])
tableau_deux_dimensions2 = np.array([[5, 4, 3, 2, 1],
                                    [1, 2, 3, 4, 5]])
print ("Produit scalaire de la matrice \n",tableau_deux_dimensions)
print ("\navec \n",tableau_deux_dimensions2)
print ("\nDans un premier temps il faut faire un transpose de la seconde matr
tableau2_transpose=tableau_deux_dimensions2.transpose()
print (tableau2_transpose)
print ("\nProduit scalaire des 2 matrices avec .dot sur la première matrice\n
print ("\nProduit scalaire des 2 matrices avec np.dot\n",np.dot(tableau_deux_
```

```
Produit scalaire de la matrice
 [[5 4 3 2 1]
 [1 2 3 4 5]]

avec
 [[5 4 3 2 1]
 [1 2 3 4 5]]

Dans un premier temps il faut faire un transpose de la seconde matrice

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

Produit scalaire des 2 matrices avec .dot sur la première matrice
 [[55 35]
 [35 55]]

Produit scalaire des 2 matrices avec np.dot
 [[55 35]
 [35 55]]
```

### Quelques fonctions utiles

In [9]:

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

print ("Nombre d'éléments du tableau\n",len(tableau_une_dimension),' ',tablea
print ("\nMinimum du tableau", tableau_une_dimension.min())
print ("\nMaximum du tableau", tableau_une_dimension.max())
print ("\nMoyenne du tableau", tableau_une_dimension.mean())
print ("\nSomme des éléments du tableau", tableau_une_dimension.sum())
print ("\nTri du tableau ",np.sort(tableau_une_dimension))
```

```
Nombre d'éléments du tableau
 10   10

Minimum du tableau 0

Maximum du tableau 9

Moyenne du tableau 4.5

Somme des éléments du tableau 45

Tri du tableau  [0 1 2 3 4 5 6 7 8 9]
```

# Les séries

Une série pandas peut être créée à partir du constructeur :

```
pandas.Series( data, index, dtype, copy)
```

où
*data* peut être un ndarray, une liste, des constantes
*index* doit être unique est hachable. Par défaut : np.arrange(n) s'il n'y a pas d'index passé
*dtype* type de données. Déterminé automatiquement s'il n'est pas indiqué
*copy* copie des données. Par défaut : false

Il est nécessaire d'importer la librairie :

In [10]:

```
import pandas as pd
```

## Exemples de création de séries

In [11]:

```
# création d'une série vide
s=pd.Series()
print ('Une série pandas vide :')
print (s)

import numpy as np
# création d'une série par np.array
data = np.array(['a','b','c','d'])
s = pd.Series(data)
print ('\nUne série pandas par np.array sans index :')
print (s)
print ('\nForme de la série :')
print (s.shape)

# création d'une série par np.array avec index
data = np.array(['a','b','c','d'])
s = pd.Series(data,index=[100,101,102,103])
print ('\nUne série pandas par np.array avec index :')
print (s)


# création d'une série par dictionnaire sans index
data = {'a' : 5.1, 'b' : 2., 'c' : 6.3}
s = pd.Series(data)
print ('\nUne série pandas par dictionnaire sans index :')
print (s)

# création d'une série par dictionnaire avec index
data = {'a' : 5.1, 'b' : 2., 'c' : 6.3}
s = pd.Series(data, index=['c','b','a'])
print ("\nremarque : l'index change l'ordre par rapport au précédent")
print ('\nUne série pandas par dictionnaire avec index :')
print (s)


# création d'une série par dictionnaire avec index
data = {'a' : 5.1, 'b' : 2., 'c' : 6.3}
s = pd.Series(data, index=['c','e','a','b'])
print ("\nQuand l'index est plus grand, la valeur prend NaN (Not a Number)")
print ('\nUne série pandas par dictionnaire avec index trop grand :')
print (s)


# création d'une série avec un scalaire et un index
s = pd.Series(10, index=[100,200,300])
print ('\nUne série pandas par un scalaire avec index :')
print (s)
```

```
Une série pandas vide :
Series([], dtype: float64)

Une série pandas par np.array sans index :
0    a
1    b
2    c
3    d
dtype: object

Forme de la série :
```

```
(4,)

Une série pandas par np.array avec index :
100    a
101    b
102    c
103    d
dtype: object

Une série pandas par dictionnaire sans index :
a    5.1
b    2.0
c    6.3
dtype: float64

remarque : l'index change l'ordre par rapport au précédent

Une série pandas par dictionnaire avec index :
c    6.3
b    2.0
a    5.1
dtype: float64

Quand l'index est plus grand, la valeur prend NaN (Not a Number)

Une série pandas par dictionnaire avec index trop grand :
c    6.3
e    NaN
a    5.1
b    2.0
dtype: float64

Une série pandas par un scalaire avec index :
100    10
200    10
300    10
dtype: int64
```

## Accès aux éléments d'une série

Les éléments peuvent être accédés par leur position de la même manière qu'un ndarray (début position 0).

Retrouver un élément d'une liste

In [12]:

```
s = pd.Series([1,2,3,4,5,6,7,8,9,10],
              index = ['a','b','c','d','e','f','g','h','i','j'])
```

In [13]:

```
print ('Le premier élément de la liste : ')
print (s[0])

print ('\n Le dernier élément de la liste : ')
print (s[len(s)-1])
```

```
Le premier élément de la liste :
1

 Le dernier élément de la liste :
10
```

Retrouver les quatre premiers éléments de la liste. Si la valeur est précédée par : alors toutes les valeurs le précédent seront renvoyées

In [14]:

```
# Les quatre premiers éléments
print ('Les quatre premiers éléments')
print (s[:4])

# Les éléments entre 3 et 6
print ('\n Les éléments entre 4 et 6')
print (s[3:6])

# Les quatre derniers éléments
print ('\nLes quatre derniers éléments')
print (s[-4:])
```

```
Les quatre premiers éléments
a    1
b    2
c    3
d    4
dtype: int64

 Les éléments entre 4 et 6
d    4
e    5
f    6
dtype: int64

Les quatre derniers éléments
g     7
h     8
i     9
j    10
dtype: int64
```

Utilisation du nom de l'index

In [15]:

```
# La première valeur de l'index
print ('Le premier élément de la liste : ')
print (s['a'])

# La dernière valeur de l'index
print ('Le dernier élément de la liste : ')
print (s['j'])
```

```
Le premier élément de la liste :
1
Le dernier élément de la liste :
10
```

Il est possible d'indexer plusieurs valeurs

In [16]:

```
# indexation de plusieurs valeurs
print ("Les valeurs pour les index a, h, j : ")
print (s[['a','h','j']])
```

```
Les valeurs pour les index a, h, j :
a     1
h     8
j    10
dtype: int64
```

# Les dataframes

Un dataframe pandas peut être créé à partir du constructeur :

```
pandas.DataFrame( data, index, columns, dtype, copy)
```

où
*data* peut être un ndarray, des séries, des map, des listes, des constantes ou un autre dataframe
*index* doit être unique est hachable. Par défaut : np.arrange(n) s'il n'y a pas d'index passé
*columns* pour le nom des colonnes. Par défaut : np.arrange(n)
*dtype* le type de données de chaque colonne.
*copy* copie des données. Par défaut : false

Un dataframe peut être créé directement, importé d'un fichier CSV, importé d'une page HTML, de SQL, etc. Ici nous ne considérons que la création directe ou celle à partir d'un CSV. Pour de plus amples information ne pas hésiter à ce reporter à la page officielle de pandas (https://pandas.pydata.org) (https://pandas.pydata.org)).

Il est nécessaire d'importer la librairie :

In [17]:

```
1 import pandas as pd
```

## Exemple de création de dataframe

In [18]:

```
# création d'un dataframe vide
df=pd.DataFrame()
print ('Un dataframe pandas vide :')
print (df)


# création d'un dataframe à partir d'une liste
data = ['a','b','c','d']
df = pd.DataFrame(data)
print ("\nUn dataframe pandas à partir d'une liste :")
print (df)
# il est possible d'utiliser display (df)
display(df)
# création d'un dataframe à partir d'une liste
data = [['France','Paris'],['Allemagne','Berlin'],
        ['Italie','Rome']]
df = pd.DataFrame(data)
print ("\nUn dataframe pandas à partir d'une liste :")
print (df)


# création d'un dataframe à partir d'une liste
data = [['France','Paris'],['Allemagne','Berlin'],
        ['Italie','Rome']]
df = pd.DataFrame(data, )
print ("\nUn dataframe pandas à partir d'une liste :")
print (df)

# création d'un dataframe à partir d'une liste avec noms de colonnes
data = [['France','Paris'],['Allemagne','Berlin'],['Italie','Rome']]
df = pd.DataFrame(data, columns=['Pays','Capitale'])
print ("\nUn dataframe pandas à partir d'une liste avec noms de colonnes :")
print (df)


# création d'un dataframe à partir d'une liste avec noms de colonnes et typag
data = [['France',67186640],['Allemagne',82695000],
        ['Italie',59464644]]
df = pd.DataFrame(data,
                  columns=['Pays','Habitants'],
                  dtype=int)
print ("\nUn dataframe pandas à partir d'une liste avec noms de colonnes et t
print (df)
print ("Les types des colonnes sont :")
print (df.info())

# Création d'un dataframe à partir d'un dictionnaire
df = pd.DataFrame(
    {'Nom': ['Pierre', 'Paul', 'Jean','Michel'],
     'Age': [25, 32, 43,60]})
print ("\nUn dataframe pandas à partir d'un dictionnaire :")
print(df)

# Création d'un dataframe à partir d'un dictionnaire en renommant les index
df = pd.DataFrame(
    {'Nom': ['Pierre', 'Paul', 'Jean','Michel'],
     'Age': [25, 32, 43,60]},
      index = ['i1', 'i2', 'i3','i4'])
print ("\nUn dataframe pandas à partir d'un dictionnaire en renommant les ind
```

```
60  print(df)
61
62  # Création d'un dataframe à partir d'une liste de dictionnaires
63  df = pd.DataFrame(
64      [{'a':10, 'b':15,'c':30,'d':40 },
65       {'a':25, 'b':32, 'd':60}])
66  print ("\nUn dataframe pandas à partir d'une liste de dictionnaires :")
67  print(df)
68
69  # Création d'un dataframe à partir d'une liste de dictionnaires en renommant
70  df = pd.DataFrame(
71      [{'a':10, 'b':15,'c':30,'d':40 },
72       {'a':25, 'b':32, 'd':60}],
73      index=['premier', 'second'])
74  print ("\nNoter le NaN (Not a Numeric number) quand il n'y a pas de valeur")
75  print ("\nUn dataframe pandas à partir d'une liste de dictionnaires en renomm
76  print(df)
77
78  # Création d'un dataframe à partir d'une liste de dictionnaires et sélection
79  data=[{'a':10, 'b':15,'c':30,'d':40 },
80       {'a':25, 'b':32, 'd':60}]
81  df = pd.DataFrame(data,
82                    index=['premier', 'second'],
83                    columns=['a','d'])
84  print ("\nNoter le NaN (Not a Numeric number quand il n'y a pas de valeur")
85  print ("\nUn dataframe pandas à partir d'une liste de dictionnaires en sélect
86  print(df)
87
88
```

```
Un dataframe pandas vide :
Empty DataFrame
Columns: []
Index: []

Un dataframe pandas à partir d'une liste :
   0
0  a
1  b
2  c
3  d
```

| | 0 |
|---|---|
| **0** | a |
| **1** | b |
| **2** | c |
| **3** | d |

```
Un dataframe pandas à partir d'une liste :
           0       1
0     France   Paris
1  Allemagne  Berlin
2     Italie    Rome

Un dataframe pandas à partir d'une liste :
           0       1
0     France   Paris
1  Allemagne  Berlin
```

```
1  Allemagne  Berlin
2     Italie    Rome

Un dataframe pandas à partir d'une liste avec noms de colonnes :
        Pays Capitale
0     France    Paris
1  Allemagne   Berlin
2     Italie     Rome

Un dataframe pandas à partir d'une liste avec noms de colonnes et typa
ge :
        Pays  Habitants
0     France   67186640
1  Allemagne   82695000
2     Italie   59464644
Les types des colonnes sont :
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 2 columns):
Pays         3 non-null object
Habitants    3 non-null int64
dtypes: int64(1), object(1)
memory usage: 128.0+ bytes
None

Un dataframe pandas à partir d'un dictionnaire :
      Nom  Age
0  Pierre   25
1    Paul   32
2    Jean   43
3  Michel   60

Un dataframe pandas à partir d'un dictionnaire en renommant les index
:
       Nom  Age
i1  Pierre   25
i2    Paul   32
i3    Jean   43
i4  Michel   60

Un dataframe pandas à partir d'une liste de dictionnaires :
    a   b     c   d
0  10  15  30.0  40
1  25  32   NaN  60

Noter le NaN (Not a Numeric number) quand il n'y a pas de valeur

Un dataframe pandas à partir d'une liste de dictionnaires en renommant
les index :
          a   b     c   d
premier  10  15  30.0  40
second   25  32   NaN  60

Noter le NaN (Not a Numeric number quand il n'y a pas de valeur

Un dataframe pandas à partir d'une liste de dictionnaires en sélection
nant des colonnes :
          a   d
premier  10  40
second   25  60
```

## Création de dataframe à partir d'un fichier CSV

Il est possible de créer un data frame à partir d'un fichier csv :

```
df = pandas.read_csv('myFile.csv')
```

Il existe de très nombreuses options (voir https://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html) (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html)).

Par défaut suppose qu'il y a un header (header = 0) et qu'il n'y a pas de noms de colonnes.
encoding='latin1' indique que le contenu doit être converti. Par défaut 'UTF-8'. sep = '\t' indique que le séparateur est une tabulation plutôt qu'une virgule.
Pour donner un nom aux colonnes : names = ['col1','col2',...,'coln']. Pour préciser les types de certaines colonnes, dtype = {'col1': str, 'col2': int, ...'col4': float}.
Pour sauter des lignes au début du fichier : skiprows = nombre de lignes à sauter. Attention la première ligne sera considérée comme celle des attributs
Pour lire un nombre limité de lignes : nrows = nombre de lignes à lire

In [19]:

```
# A partir d'un fichier csv
#"https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.da
names = ['sepal-length', 'sepal-width',
         'petal-length', 'petal-width', 'class']

df = pd.read_csv(url, names=names)
# 5 premières lignes du fichier
df.head()

```

Out[19]:

| | sepal-length | sepal-width | petal-length | petal-width | class |
|---|---|---|---|---|---|
| **0** | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| **1** | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| **4** | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |

Considérer le fichier exemple.csv suivant :

In [20]:

```
#Création d'un fichier exemple
fichier = open("exemple.csv", "w")
fichier.write("A;B;C;D\n")
fichier.write("Pierre;10;18.5;14.5\n")
fichier.write("Paul;12;18.7;15.5\n")
fichier.write("Jacques;11;15.3;15.5\n")
fichier.close()
```

In [21]:

```
# lecture du fichier en changeant de séparateur
df = pd.read_csv('exemple.csv',sep=';')
print ("Lecture du fichier exemple.csv avec un séparateur ;\n")
print (df)

# lecture du fichier sans lire la première ligne
df = pd.read_csv('exemple.csv',sep=';', skiprows=1)
print ("\nLecture du fichier exemple.csv en sautant")
print ("une ligne attention la première ligne devient la liste des attributs.
print (df)


# lecture du fichier en mettant des noms aux colonnes
df = pd.read_csv('exemple.csv',sep=';',skiprows=1,
                 names=['Nom','Age','Note1','Note2'])
print ("\nLecture du fichier exemple.csv en sautant")
print ("une ligne et en mettant des noms aux attributs.")
print ("La première ligne commence au bon index. \n")
print (df)

```

```
Lecture du fichier exemple.csv avec un séparateur ;

         A   B     C     D
0   Pierre  10  18.5  14.5
1     Paul  12  18.7  15.5
2  Jacques  11  15.3  15.5

Lecture du fichier exemple.csv en sautant
une ligne attention la première ligne devient la liste des attributs.

    Pierre  10  18.5  14.5
0     Paul  12  18.7  15.5
1  Jacques  11  15.3  15.5

Lecture du fichier exemple.csv en sautant
une ligne et en mettant des noms aux attributs.
La première ligne commence au bon index.

       Nom  Age  Note1  Note2
0   Pierre   10   18.5   14.5
1     Paul   12   18.7   15.5
2  Jacques   11   15.3   15.5
```

## Accès aux éléments d'un dataframe

Comme les séries les dataframes peuvent être accédés par leur position de la même manière qu'un ndarray (début position 0), par les index ou par le nom de la colonne. L'intérêt des dataframe est justement de pouvoir utiliser le nom des colonnes pour les accès.

In [22]:

```
df = pd.DataFrame(
    {'Nom': ['Pierre', 'Paul', 'Jean','Michel'],
     'Age': [25, 32, 43,60]},
      index = ['i1', 'i2', 'i3','i4'])

print('Le dataframe : ')
print ("\n",df)

print ('\nLa colonne correspondant au Nom dans le dataframe :  ')
print ("\n",df['Nom'])

```

```
Le dataframe :

        Nom  Age
i1  Pierre   25
i2    Paul   32
i3    Jean   43
i4  Michel   60

La colonne correspondant au Nom dans le dataframe :

 i1    Pierre
i2      Paul
i3      Jean
i4    Michel
Name: Nom, dtype: object
```

**Accès aux lignes d'un dataframe**

Il est possible d'accéder aux lignes d'un dataframe par leur nom ou bien en précisant l'intervalle.

In [23]:

```
print ("La ligne corrspondant à l'index i3 avec loc :")
print (df.loc[['i3']])

print ("\nLes trois premières lignes avec loc :")
# df.loc[inclusive:inclusive]
print (df.loc['i1':'i3'])


print ('\nLa première ligne du dataframe en utilisant la position : ')
print (df[:1])


print ('\nLa dernière ligne du dataframe en utilisant la position :  ')
print (df[len(df)-1:])

print ('\nLes lignes 2 et 3 du dataframe en utilisant la position :  ')
print (df[1:3])
# df.iloc[inclusive:exclusive]
# Note: .iloc est uniquement lié à
#la position et non pas au nom de l'index
print ("\nLes lignes 2 et 3 du dataframe avec iloc : ")
print (df.iloc[1:3])
```

```
La ligne corrspondant à l'index i3 avec loc :
     Nom  Age
i3  Jean   43

Les trois premières lignes avec loc :
       Nom  Age
i1  Pierre   25
i2    Paul   32
i3    Jean   43

La première ligne du dataframe en utilisant la position :
       Nom  Age
i1  Pierre   25

La dernière ligne du dataframe en utilisant la position :
       Nom  Age
i4  Michel   60

Les lignes 2 et 3 du dataframe en utilisant la position :
     Nom  Age
i2  Paul   32
i3  Jean   43

Les lignes 2 et 3 du dataframe avec iloc :
     Nom  Age
i2  Paul   32
i3  Jean   43
```

Il est possible de spécifier les colonnes dans le résultat

In [24]:

```
1  df.loc[['i3'],['Age']]
```

Out[24]:

| | Age |
|---|---|
| **i3** | 43 |

# Manipulation des dataframes

## Information sur les dataframes

pandas propose de nombreuses fonctions pour connaître les informations des dataframes.

*df.info()* : donne des infos sur le dataframe
*df.head()* : retourne les 5 premières lignes
*df.tail()* : retourne les 5 dernières lignes
*df.sample()* : retourne un ensemble aléatoire de données *df.head(10)* (*df.tail(10)*) : retourne les 10 premières lignes (resp. les 10 dernières) *df.shape* : renvoie la taille du dataframe avec nombre de lignes, nombre de colonnes *df.ndim* : retourne le nombre de dimensions
*df.columns* : retourne les noms des colonnes
*df.columns.values* : le nom des colonnes sous forme d'array numpy
*df.dtypes* : retourne les différents types du dataframe
*df.index* : les noms des lignes (individus)
*df.index.values* : le nom des lignes sous forme d'array numpy
*df.values* : pour récupérer le dataframe sous forme d'array numpy 2d
*df.describe()* : renvoie un dataframe donnant des statistiques, pour les colonnes numériques, sur les valeurs (nombres de valeurs, moyenne, écart-type, ...)

In [25]:

```
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.da
names = ['SepalLengthCm', 'SepalWidthCm',
         'PetalLengthCm', 'PetalWidthCm', 'Species']

df = pd.read_csv(url, names=names)

print ("Info \n")
print (df.info())
print ("\nLes deux premières lignes\n")
print (df.head(2))
print ("\nLes deux dernières lignes\n")
print (df.tail(2))
print ("\nCinq lignes au hasard\n")
print (df.sample(5))
print ('\nDimension du dataframe\n')
print (df.shape)
print ('\n\t Il y a :',df.shape[0],'lignes et',
       df.shape[1], 'colonnes\n')
print ('\nLe nombre de dimensions\n')
print (df.ndim)
print ('\nLes différents type du dataframe\n')
print (df.dtypes)

print ("\nNoms des colonnes\n")
print (df.columns)
print ('\nNom des index\n')
print (df.index)
print ('\nStatistiques élémentaires\n')
print (df.describe())
```

```
Info

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
SepalLengthCm    150 non-null float64
SepalWidthCm     150 non-null float64
PetalLengthCm    150 non-null float64
PetalWidthCm     150 non-null float64
Species          150 non-null object
dtypes: float64(4), object(1)
memory usage: 5.9+ KB
None

Les deux premières lignes

   SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Speci
es
0            5.1           3.5            1.4           0.2  Iris-seto
sa
1            4.9           3.0            1.4           0.2  Iris-seto
sa

Les deux dernières lignes

     SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
Species
148            6.2           3.4            5.4           2.3  Iris-vi
rginica
```

```
149            5.9           3.0            5.1           1.8  Iris-vi
rginica
```

```
Cinq lignes au hasard
```

```
    SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
Species
41            4.5           2.3            1.3           0.3      Iris
-setosa
89            5.5           2.5            4.0           1.3  Iris-ver
sicolor
13            4.3           3.0            1.1           0.1      Iris
-setosa
77            6.7           3.0            5.0           1.7  Iris-ver
sicolor
37            4.9           3.1            1.5           0.1      Iris
-setosa
```

```
Dimension du dataframe
```

```
(150, 5)
```

```
         Il y a : 150 lignes et 5 colonnes
```

```
Le nombre de dimensions
```

```
2
```

```
Les différents type du dataframe
```

```
SepalLengthCm    float64
SepalWidthCm     float64
PetalLengthCm    float64
PetalWidthCm     float64
Species           object
dtype: object
```

```
Noms des colonnes
```

```
Index(['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthC
m',
       'Species'],
      dtype='object')
```

```
Nom des index
```

```
RangeIndex(start=0, stop=150, step=1)
```

```
Statistiques élémentaires
```

```
       SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
count     150.000000    150.000000     150.000000    150.000000
mean        5.843333      3.054000       3.758667      1.198667
std         0.828066      0.433594       1.764420      0.763161
min         4.300000      2.000000       1.000000      0.100000
25%         5.100000      2.800000       1.600000      0.300000
50%         5.800000      3.000000       4.350000      1.300000
75%         6.400000      3.300000       5.100000      1.800000
max         7.900000      4.400000       6.900000      2.500000
```

# Faire une copie d'un dataframe

Il est parfois utile de faire une copie d'un dataframe. Il existe deux manières différentes.

df2=df

**Attention** toute modification faite sur df2 sera aussi reportée sur df

df2=df.copy()

Deux versions indépendantes sont créées.

## Manipulation des colonnes et des lignes

In [26]:

```
#Dataframe
d = [1,2,3,4,5]

df = pd.DataFrame(d)
print (df)
```

```
   0
0  1
1  2
2  3
3  4
4  5
```

### Changement du nom de colonne

In [27]:

```
# Changement du nom de colonne
df.columns = ['Colonne']
print (df)
```

```
   Colonne
0        1
1        2
2        3
3        4
4        5
```

### Selection par valeur

In [28]:

```
print ("Pour une valeur :")
print (df.loc[df['Colonne']==1])

print ("\nEn prenant plusieurs valeurs avec isin :")
print(df.loc[df['Colonne'].isin([1,2])])
```

```
Pour une valeur :
   Colonne
0        1

En prenant plusieurs valeurs avec isin :
   Colonne
0        1
1        2
```

### Trier les valeurs d'une colonne

In [29]:

```
print ('Tri par ordre décroissant :')
print(df.sort_values(by='Colonne',ascending=False))

print ('\nTri par ordre croissant (par défaut) : ')
print(df.sort_values(by='Colonne'))
```

```
Tri par ordre décroissant :
   Colonne
4        5
3        4
2        3
1        2
0        1

Tri par ordre croissant (par défaut) :
   Colonne
0        1
1        2
2        3
3        4
4        5
```

### Statistiques sur une colonne

In [30]:

```
print ("Moyenne de la colonne : ")
print(df['Colonne'].mean())

print ('\nMaximum de la colonne :')
print(df['Colonne'].max())

print ('\nMinimul de la colonne :')
print(df['Colonne'].min())

print ("\nComptage des différentes valeurs de la colonne :")
print(df['Colonne'].value_counts())
print ("\nAjout d'une nouvelle ligne avec 5 pour vérifier : ")
df.loc[len(df)] = [5]
print (df)
print(df['Colonne'].value_counts())
df=df.drop(df.index[-1])

```

```
Moyenne de la colonne :
3.0

Maximum de la colonne :
5

Minimul de la colonne :
1

Comptage des différentes valeurs de la colonne :
5    1
4    1
3    1
2    1
1    1
Name: Colonne, dtype: int64

Ajout d'une nouvelle ligne avec 5 pour vérifier :
   Colonne
0        1
1        2
2        3
3        4
4        5
5        5
5    2
4    1
3    1
2    1
1    1
Name: Colonne, dtype: int64
```

### Ajout d'une colonne

In [31]:

```
# Ajout d'une colonne avec 1 comme valeur
df['Nouvelle Colonne'] = 1
print (df)

print ("\nSélection par valeur sur plusieurs colonnes avec un ET : ")
print (df.loc[(df['Colonne']==3) & (df['Nouvelle Colonne']==1)])

print ("\nSélection par valeur sur plusieurs colonnes avec un OU : ")
print (df.loc[(df['Colonne']==3) | (df['Nouvelle Colonne']==1)])
```

```
   Colonne  Nouvelle Colonne
0        1                 1
1        2                 1
2        3                 1
3        4                 1
4        5                 1

Sélection par valeur sur plusieurs colonnes avec un ET :
   Colonne  Nouvelle Colonne
2        3                 1

Sélection par valeur sur plusieurs colonnes avec un OU :
   Colonne  Nouvelle Colonne
0        1                 1
1        2                 1
2        3                 1
3        4                 1
4        5                 1
```

### Modification d'une colonne

In [32]:

```
# Modification de la colonne en ajoutant un nombre aléaloire
import random
nb=random.randint(1, 6)
df['Nouvelle Colonne'] = df['Nouvelle Colonne'] + nb
df
```

Out[32]:

| | Colonne | Nouvelle Colonne |
|---|---|---|
| **0** | 1 | 5 |
| **1** | 2 | 5 |
| **2** | 3 | 5 |
| **3** | 4 | 5 |
| **4** | 5 | 5 |

### Supression d'une colonne

In [33]:

```
# Supression d'une colonne
del df['Nouvelle Colonne']
df
```

Out[33]:

| | Colonne |
|---|---|
| **0** | 1 |
| **1** | 2 |
| **2** | 3 |
| **3** | 4 |
| **4** | 5 |

### Ajout d'une ligne

In [34]:

```
print ("Ajout d'une ligne à la fin : ")
df.loc[len(df)] = [6]
print (df)

print ("\nAjout d'une ligne au début attention il faut reorganiser les index
df.loc[-1]=[7]
df.index = df.index + 1  # reorganiser les index
df = df.sort_index()  # trier les index
print (df)
```

```
Ajout d'une ligne à la fin :
   Colonne
0        1
1        2
2        3
3        4
4        5
5        6

Ajout d'une ligne au début attention il faut reorganiser les index :
   Colonne
0        7
1        1
2        2
3        3
4        4
5        5
6        6
```

### Modification d'une ligne

In [35]:

```
print (df)
print ("Modification de la valeur de la troisième ligne")
df.loc[3] = [10]
print (df)
```

```
   Colonne
0        7
1        1
2        2
3        3
4        4
5        5
6        6
Modification de la valeur de la troisième ligne
   Colonne
0        7
1        1
2        2
3       10
4        4
5        5
6        6
```

### Suppression d'une ligne

In [36]:

```
print ("En utilisant une condition sur la colone : \n")
print ("Avant ",df)
df=df[df['Colonne']!=3]
print ("\nAprès ",df)

print ("\nEn utilisant les index (suppression de la dernière ligne) : ")
df=df.drop(df.index[-1])
print (df)
```

```
En utilisant une condition sur la colone :

Avant     Colonne
0         7
1         1
2         2
3        10
4         4
5         5
6         6

Après     Colonne
0         7
1         1
2         2
3        10
4         4
5         5
6         6

En utilisant les index (suppression de la dernière ligne) :
   Colonne
0         7
1         1
2         2
3        10
4         4
5         5
```

**Suppression d'une ligne dont la valeur est NaN**

In [37]:

```
import numpy as np
print ("Ajout d'une ligne à la fin ne contenant rien (utilisant de numpy nan)
df.loc[len(df)] = [np.nan]
print (df)

print ("\nSuppression des lignes n'ayant pas de valeur : ")
df=df.dropna()
print (df)
```

```
Ajout d'une ligne à la fin ne contenant rien (utilisant de numpy nan)
:
   Colonne
0      7.0
1      1.0
2      2.0
3     10.0
4      4.0
5      5.0
6      NaN

Suppression des lignes n'ayant pas de valeur :
   Colonne
0      7.0
1      1.0
2      2.0
3     10.0
4      4.0
5      5.0
```

### re-indexer un index

In [38]:

```
df = df.reset_index(drop=True)
df
```

Out[38]:

| | Colonne |
|---|---|
| **0** | 7.0 |
| **1** | 1.0 |
| **2** | 2.0 |
| **3** | 10.0 |
| **4** | 4.0 |
| **5** | 5.0 |

### Changement du nom des index

In [39]:

```
#il est possible de changer les noms ou valeurs des index
d = [1,2,3,4,5]

df = pd.DataFrame(d,columns = ['Colonne'])
print ("Dataframe initial :\n ",df)
i = [100,200,300,400,500]
df.index = i
print ("\nDataframe en changeant de valeur d'index : \n",df)

i = ['a','b','c','d','e']
df.index = i
print ("\nDataframe en changeant de valeur d'index avec des lettres : \n",df)
```

```
Dataframe initial :
     Colonne
0        1
1        2
2        3
3        4
4        5

Dataframe en changeant de valeur d'index :
      Colonne
100        1
200        2
300        3
400        4
500        5

Dataframe en changeant de valeur d'index avec des lettres :
    Colonne
a        1
b        2
c        3
d        4
e        5
```

### Application d'une fonction à un dataframe

In [40]:

```
def multiplication (x):
    return 100*x

print (df['Colonne'].apply(multiplication))
```

```
a    100
b    200
c    300
d    400
e    500
Name: Colonne, dtype: int64
```

### Boucler sur les colonnes

In [41]:

```
df = pd.DataFrame(
    {'Nom': ['Pierre', 'Paul', 'Jean','Michel'],
     'Age': [23, 22, 23,20],
     'Note' : [15, 13, 14, 16]},
      index = ['i1', 'i2', 'i3','i4'])


```

In [42]:

```
# il est possible de boucler sur les colonnes
for col in df.columns:
              print(df[col].dtype)
```

```
object
int64
int64
```

### Trier les colonnes

Il est possible de trier l'ensemble du dataframe par en fonction de valeur de colonnes à l'aide de la fonction sort_values.

In [43]:

```
print ("Dataframe initial : \n")
print (df)

print ("\nDataframe trié par Age : \n")
print (df.sort_values(by=['Age'], ascending=True))

print ("\nDataframe trié par Age et Note : \n")
print (df.sort_values(by=['Age','Note'],
                      ascending=True))
```

```
Dataframe initial :

       Nom  Age  Note
i1  Pierre   23    15
i2    Paul   22    13
i3    Jean   23    14
i4  Michel   20    16

Dataframe trié par Age :

       Nom  Age  Note
i4  Michel   20    16
i2    Paul   22    13
i1  Pierre   23    15
i3    Jean   23    14

Dataframe trié par Age et Note :

       Nom  Age  Note
i4  Michel   20    16
i2    Paul   22    13
i3    Jean   23    14
i1  Pierre   23    15
```

### Groupby

Il est possible de faire des group by comme en SQL :

In [44]:

```
# Definition du groupby sur la colonne note
g = df.groupby('Note')

# Il est possible de faire une boucle sur toutes les partitions
for groupe in g:
    #groupe est un tuple
    print(groupe[0]) # valeur du partitionnement
    # Affichage des valeurs
    print(groupe[1])
    print ("\n")

```

```
13
     Nom  Age  Note
i2  Paul   22    13


14
     Nom  Age  Note
i3  Jean   23    14


15
       Nom  Age  Note
i1  Pierre   23    15


16
       Nom  Age  Note
i4  Michel   20    16
```

## Travailler avec plusieurs dataframes

### Concaténation

Il est possible de concaténer des dataframes à l'aide de la fonction *concat*

In [45]:

```
df = pd.DataFrame(
    {'Nom': ['Pierre', 'Paul', 'Jean','Michel'],
     'Age': [25, 32, 43,60],
     'Note' : [14, 13, 14, 16],
     'Sujet_id' : [5,3,1,4]},
      index = ['i1', 'i2', 'i3','i4'])

df2 = pd.DataFrame(
    { 'Sujet_id' : [1, 2, 3, 4],
      'Libelle' : ['Math','Informatique','Physique','Chimie']})

print ('Dataframe 1 : \n',df)
print ('\nDataframe 1 : \n',df2)

print ('\nConcaténation de deux dataframes en ligne : ')
print (pd.concat([df, df2]))

print ('\nConcaténation de deux dataframes en colonne : ')
print (pd.concat([df, df2], axis=1))

```

```
Dataframe 1 :
        Nom  Age  Note  Sujet_id
i1  Pierre   25    14         5
i2    Paul   32    13         3
i3    Jean   43    14         1
i4  Michel   60    16         4

Dataframe 1 :
    Sujet_id       Libelle
0         1          Math
1         2  Informatique
2         3      Physique
3         4        Chimie

Concaténation de deux dataframes en ligne :
     Age       Libelle     Nom  Note  Sujet_id
i1  25.0           NaN  Pierre  14.0         5
i2  32.0           NaN    Paul  13.0         3
i3  43.0           NaN    Jean  14.0         1
i4  60.0           NaN  Michel  16.0         4
0    NaN          Math     NaN   NaN         1
1    NaN  Informatique     NaN   NaN         2
2    NaN      Physique     NaN   NaN         3
3    NaN        Chimie     NaN   NaN         4

Concaténation de deux dataframes en colonne :
        Nom   Age  Note  Sujet_id  Sujet_id       Libelle
i1  Pierre  25.0  14.0       5.0       NaN           NaN
i2    Paul  32.0  13.0       3.0       NaN           NaN
i3    Jean  43.0  14.0       1.0       NaN           NaN
i4  Michel  60.0  16.0       4.0       NaN           NaN
0      NaN   NaN   NaN       NaN       1.0          Math
1      NaN   NaN   NaN       NaN       2.0  Informatique
2      NaN   NaN   NaN       NaN       3.0      Physique
3      NaN   NaN   NaN       NaN       4.0        Chimie

/Users/pascalponcelet/Desktop/Sicki-learn/Tools/tools/lib/python3.6/si
te-packages/ipykernel_launcher.py:16: FutureWarning: Sorting because n
```

```
on-concatenation axis is not aligned. A future version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=Tru
e'.

  app.launch_new_instance()
```

**Jointure**

Il est possible d'exprimer différentes jointures (inner, outer, left, right) à l'aide de *merge*

In [46]:

```
print ("\nJointure de deux dataframes en fonction de Sujet_id : ")
print (pd.merge(df, df2, on='Sujet_id', how='inner'))

print ("\nJointure externe (outer join) de deux dataframes en fonction de Suj
print (pd.merge(df, df2, on='Sujet_id', how='outer'))


print ("\nJointure externe droite (right outer join) : \n")
print (pd.merge(df, df2, on='Sujet_id', how='right'))

print ("\nJointure externe gauche (left outer join) : \n")
print (pd.merge(df, df2, on='Sujet_id', how='left'))


```

```
Jointure de deux dataframes en fonction de Sujet_id :
      Nom  Age  Note  Sujet_id   Libelle
0    Paul   32    13         3  Physique
1    Jean   43    14         1      Math
2  Michel   60    16         4    Chimie

Jointure externe (outer join) de deux dataframes en fonction de Sujet_
id :
      Nom   Age  Note  Sujet_id       Libelle
0  Pierre  25.0  14.0         5           NaN
1    Paul  32.0  13.0         3      Physique
2    Jean  43.0  14.0         1          Math
3  Michel  60.0  16.0         4        Chimie
4     NaN   NaN   NaN         2  Informatique

Jointure externe droite (right outer join) :

      Nom   Age  Note  Sujet_id       Libelle
0    Paul  32.0  13.0         3      Physique
1    Jean  43.0  14.0         1          Math
2  Michel  60.0  16.0         4        Chimie
3     NaN   NaN   NaN         2  Informatique

Jointure externe gauche (left outer join) :

      Nom  Age  Note  Sujet_id   Libelle
0  Pierre   25    14         5       NaN
1    Paul   32    13         3  Physique
2    Jean   43    14         1      Math
3  Michel   60    16         4    Chimie
```

## Sauvegarde des dataframes

Un dataframe peut être sauvegardé dans un fichier CSV.

In [47]:

```
df.to_csv('myFile.csv')
```

*Separateur*. Par défaut le séparateur est une virgule. df.to_csv('myFile.csv', sep = '\t') utilise une tabulation

comme séparateur
*Header* Par defaut le header est sauvegardé. df.to_csv('myFile.csv', header=false) pour ne pas sauver l'entête
*Index* Par défaut le nom des lignes est sauvegardé. df.to_csv('myFile.csv', index=false) pour ne pas les sauvegarder
*NaN* Par défaut les NaN sont considérées comme des chaînes vides. Il est possible de remplacer le caractère. df.to_csv('myFile.csv', na_rep='-') remplace les valeurs manquantes par des -.

In [48]:

```
df = pd.DataFrame(
    {'Nom': ['Pierre', 'Paul', 'Jean','Michel'],
     'Age': [25, 32, 43,60],
     'Note' : [14, 13, 14, 16],
     'Sujet_id' : [5,3,1,4]},
      index = ['i1', 'i2', 'i3','i4'])

import sys
print ('Affichage du fichier sauvegardé sur stdout \n')
df.to_csv(sys.stdout)

print ('\nAffichage du fichier sauvegardé avec tabulation \n')
df.to_csv(sys.stdout,sep='\t')

print ('\nAffichage du fichier sauvegardé avec tabulation sans header\n')
df.to_csv(sys.stdout,sep='\t', header=False)


print ('\nAffichage du fichier sauvegardé avec tabulation sans index \n')
df.to_csv(sys.stdout,sep='\t', index=False)

print ('\nSauvegarde du fichier monfichier.csv \n')
df.to_csv('monfichier.csv',sep='\t', index=False)

print ('\nLecture pour vérification \n')
df = pd.read_csv('monfichier.csv',sep='\t')
print (df)
```

```
Affichage du fichier sauvegardé sur stdout

,Nom,Age,Note,Sujet_id
i1,Pierre,25,14,5
i2,Paul,32,13,3
i3,Jean,43,14,1
i4,Michel,60,16,4

Affichage du fichier sauvegardé avec tabulation

        Nom     Age     Note    Sujet_id
i1      Pierre  25      14      5
i2      Paul    32      13      3
i3      Jean    43      14      1
i4      Michel  60      16      4

Affichage du fichier sauvegardé avec tabulation sans header

i1      Pierre  25      14      5
i2      Paul    32      13      3
i3      Jean    43      14      1
i4      Michel  60      16      4

Affichage du fichier sauvegardé avec tabulation sans index

Nom     Age     Note    Sujet_id
Pierre  25      14      5
Paul    32      13      3
Jean    43      14      1
Michel  60      16      4
```

```
Sauvegarde du fichier monfichier.csv
```

```
Lecture pour vérification
```

| | Nom | Age | Note | Sujet_id |
|---|---|---|---|---|
| 0 | Pierre | 25 | 14 | 5 |
| 1 | Paul | 32 | 13 | 3 |
| 2 | Jean | 43 | 14 | 1 |
| 3 | Michel | 60 | 16 | 4 |