

# 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 fonctionnalités similaires à celles de système de gestion de base de données (sélection, 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.

## 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.arange(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 [1]:

```
1 import pandas as pd
```

## Exemples de création de séries

In [2]:

```
1  # création d'une série vide
2  s=pd.Series()
3  print ('Une série pandas vide :')
4  print (s)
5
6  import numpy as np
7  # création d'une série par np.array
8  data = np.array(['a','b','c','d'])
9  s = pd.Series(data)
10 print ('\nUne série pandas par np.array sans index :')
11 print (s)
12
13 # création d'une série par np.array avec index
14 data = np.array(['a','b','c','d'])
15 s = pd.Series(data,index=[100,101,102,103])
16 print ('\nUne série pandas par np.array avec index :')
17 print (s)
18
19
20 # création d'une série par dictionnaire sans index
21 data = {'a' : 5.1, 'b' : 2., 'c' : 6.3}
22 s = pd.Series(data)
23 print ('\nUne série pandas par dictionnaire sans index :')
24 print (s)
25
26 # création d'une série par dictionnaire avec index
27 data = {'a' : 5.1, 'b' : 2., 'c' : 6.3}
28 s = pd.Series(data, index=['c','b','a'])
29 print ("\nremarque : l'index change l'ordre par rapport au précédent")
30 print ('\nUne série pandas par dictionnaire avec index :')
31 print (s)
32
33
34 # création d'une série par dictionnaire avec index
35 data = {'a' : 5.1, 'b' : 2., 'c' : 6.3}
```

```

36 s = pd.Series(data, index=['c','e','a','b'])
37 print ("\nQuand l'index est plus grand, la valeur prend NaN (Not a Number)")
38 print ('\nUne série pandas par dictionnaire avec index trop grand :')
39 print (s)
40
41
42 # création d'une série avec un scalaire et un index
43 s = pd.Series(10, index=[100,200,300])
44 print ('\nUne série pandas par un scalaire avec index :')
45 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

```

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 [3]:

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

In [4]:

```
1 print ('Le premier élément de la liste : '  
2 print (s[0])  
3  
4 print ('\n Le dernier élément de la liste : '  
5 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 [5]:

```
1  ▼  # Les quatre premiers éléments
2      print ('Les quatre premiers éléments')
3      print (s[:4])
4
5      # Les éléments entre 3 et 6
6      print ('\n Les éléments entre 4 et 6')
7      print (s[3:6])
8
9      # Les quatre derniers éléments
10     print ('\nLes quatre derniers éléments')
11     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 [6]:

```
1  ▼  # La première valeur de l'index
2      print ('Le premier élément de la liste : ')
3      print (s['a'])
4
5      # La dernière valeur de l'index
6      print ('Le dernier élément de la liste : ')
7      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 [7]:

```
1  # indexation de plusieurs valeurs
2  print ("Les valeurs pour les index a, h, j : ")
3  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.arange(n) s'il n'y a pas d'index passé

*columns* pour le nom des colonnes. Par défaut : np.arange(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 [8]:

```
1  import pandas as pd
```

## Exemple de création de dataframe

In [9]:

```
1  # création d'un dataframe vide
2  df=pd.DataFrame()
3  print ('Un dataframe pandas vide :')
4  print (df)
5
6
7  # création d'un dataframe à partir d'une liste
8  data = ['a', 'b', 'c', 'd']
9  df = pd.DataFrame(data)
10 print ("\nUn dataframe pandas à partir d'une liste :")
```

```

11 print (df)
12 # il est possible d'utiliser display (df)
13 display(df)
14 # création d'un dataframe à partir d'une liste
15 ▼ data = [['France', 'Paris'], ['Allemagne', 'Berlin'],
16          ['Italie', 'Rome']]
17 df = pd.DataFrame(data)
18 print ("\nUn dataframe pandas à partir d'une liste :")
19 print (df)
20
21
22 # création d'un dataframe à partir d'une liste
23 ▼ data = [['France', 'Paris'], ['Allemagne', 'Berlin'],
24          ['Italie', 'Rome']]
25 df = pd.DataFrame(data, )
26 print ("\nUn dataframe pandas à partir d'une liste :")
27 print (df)
28
29 # création d'un dataframe à partir d'une liste avec noms de colonnes
30 data = [['France', 'Paris'], ['Allemagne', 'Berlin'], ['Italie', 'Rome']]
31 df = pd.DataFrame(data, columns=['Pays', 'Capitale'])
32 print ("\nUn dataframe pandas à partir d'une liste avec noms de colonnes :")
33 print (df)
34
35
36 # création d'un dataframe à partir d'une liste avec noms de colonnes et types
37 ▼ data = [['France', 67186640], ['Allemagne', 82695000],
38          ['Italie', 59464644]]
39 ▼ df = pd.DataFrame(data,
40                     columns=['Pays', 'Habitants'],
41                     dtype=int)
42 print ("\nUn dataframe pandas à partir d'une liste avec noms de colonnes et types :")
43 print (df)
44 print ("Les types des colonnes sont :")
45 print (df.info())
46
47 # Création d'un dataframe à partir d'un dictionnaire
48 ▼ df = pd.DataFrame(
49 ▼     {'Nom': ['Pierre', 'Paul', 'Jean', 'Michel'],
50      'Age': [25, 32, 43, 60]})
51 print ("\nUn dataframe pandas à partir d'un dictionnaire :")
52 print(df)
53
54 # Création d'un dataframe à partir d'un dictionnaire en renommant les index
55 ▼ df = pd.DataFrame(
56 ▼     {'Nom': ['Pierre', 'Paul', 'Jean', 'Michel'],
57 ▼     'Age': [25, 32, 43, 60]},
58     index = ['i1', 'i2', 'i3', 'i4'])
59 print ("\nUn dataframe pandas à partir d'un dictionnaire en renommant les index :")
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 renommant"
76 print(df)
77
78 # Création d'un dataframe à partir d'une liste de dictionnaires et sélection de colonnes
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 sélection de colonnes"
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
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 type :

```
      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 :

```
   Age  Nom
0   25  Pierre
1   32   Paul
2   43   Jean
3   60  Michel
```

Un dataframe pandas à partir d'un dictionnaire en renommant les index :

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

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 sélectionnant 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)) ([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 [10]:

```
1  # A partir d'un fichier csv
2  #"https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data
3  url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris
4  names = ['sepal-length', 'sepal-width',
5           'petal-length', 'petal-width', 'class']
6
7  df = pd.read_csv(url, names=names)
8  # 5 premières lignes du fichier
9  df.head()
10
```

Out[10]:

	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 [11]:

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

In [12]:

```
1  ▼ # lecture du fichier en changeant de séparateur
2    df = pd.read_csv('exemple.csv',sep=';')
3    print ("Lecture du fichier exemple.csv avec un séparateur ;\n")
4    print (df)
5
6    # lecture du fichier sans lire la première ligne
7    df = pd.read_csv('exemple.csv',sep=';', skiprows=1)
8    print ("\nLecture du fichier exemple.csv en sautant")
9    print ("une ligne attention la première ligne devient la liste des attributs")
10   print (df)
11
12
13   # lecture du fichier en mettant des noms aux colonnes
14  ▼ df = pd.read_csv('exemple.csv',sep=';',skiprows=1,
15                    names=['Nom','Age','Note1','Note2'])
16   print ("\nLecture du fichier exemple.csv en sautant")
17   print ("une ligne et en mettant des noms aux attributs.")
18   print ("La première ligne commence au bon index. \n")
19   print (df)
20
```

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 [13]:

```
1  df = pd.DataFrame(  
2      {'Nom': ['Pierre', 'Paul', 'Jean', 'Michel'],  
3      'Age': [25, 32, 43, 60]},  
4      index = ['i1', 'i2', 'i3', 'i4'])  
5  
6  print('Le dataframe : ')  
7  print ("\n",df)  
8  
9  print ('\nLa colonne correspondant au Nom dans le dataframe : ')  
10 print ("\n",df['Nom'])  
11
```

Le dataframe :

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

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 [14]:

```
1 print ("La ligne correspondant à l'index i3 avec loc :")
2 print (df.loc[['i3']])
3
4 print ("\nLes trois premières lignes avec loc :")
5 # df.loc[inclusive:inclusive]
6 print (df.loc['i1':'i3'])
7
8
9 print ('\nLa première ligne du dataframe en utilisant la position : ')
10 print (df[:1])
11
12
13 print ('\nLa dernière ligne du dataframe en utilisant la position : ')
14 print (df[len(df)-1:])
15
16 print ('\nLes lignes 2 et 3 du dataframe en utilisant la position : ')
17 print (df[1:3])
18 # df.iloc[inclusive:exclusive]
19 # Note: .iloc est uniquement lié à
20 #la position et non pas au nom de l'index
21 print ("\nLes lignes 2 et 3 du dataframe avec iloc : ")
22 print (df.iloc[1:3])
```

La ligne correspondant à l'index i3 avec loc :

	Age	Nom
i3	43	Jean

Les trois premières lignes avec loc :

	Age	Nom
i1	25	Pierre
i2	32	Paul
i3	43	Jean

La première ligne du dataframe en utilisant la position :

	Age	Nom
i1	25	Pierre

La dernière ligne du dataframe en utilisant la position :

	Age	Nom
i4	60	Michel

Les lignes 2 et 3 du dataframe en utilisant la position :

	Age	Nom
i2	32	Paul
i3	43	Jean

Les lignes 2 et 3 du dataframe avec iloc :

	Age	Nom
i2	32	Paul
i3	43	Jean

Il est possible de spécifier les colonnes dans le résultat

In [15]:

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

Out[15]:

	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 [16]:

```
1 url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris"
2 names = ['SepalLengthCm', 'SepalWidthCm',
3          'PetalLengthCm', 'PetalWidthCm', 'Species']
4
5 df = pd.read_csv(url, names=names)
6
7 print ("Info \n")
8 print (df.info())
9 print ("\nLes deux premières lignes\n")
10 print (df.head(2))
11 print ("\nLes deux dernières lignes\n")
12 print (df.tail(2))
13 print ("\nCinq lignes au hasard\n")
14 print (df.sample(5))
15 print ('\nDimension du dataframe\n')
16 print (df.shape)
17 print ('\n\t Il y a :', df.shape[0], 'lignes et',
```

```
18         df.shape[1], 'colonnes\n')
19     print ('\nLe nombre de dimensions\n')
20     print (df.ndim)
21     print ('\nLes différents type du dataframe\n')
22     print (df.dtypes)
23
24     print ("\nNoms des colonnes\n")
25     print (df.columns)
26     print ('\nNom des index\n')
27     print (df.index)
28     print ('\nStatistiques élémentaires\n')
29     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	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa

Les deux dernières lignes

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

Cinq lignes au hasard

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
64	5.6	2.9	3.6	1.3	Iris-versicolor
40	5.0	3.5	1.3	0.3	Iris-setosa
17	5.1	3.5	1.4	0.3	Iris-setosa
34	4.9	3.1	1.5	0.1	Iris-setosa

16	5.4	3.9	1.3	0.4	Ir
----	-----	-----	-----	-----	----

is-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', 'PetalWidthCm',
      '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 [17]:

```
1  ▼  #Dataframe
2     d = [1,2,3,4,5]
3
4     df = pd.DataFrame(d)
5     print (df)
```

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

## Changement du nom de colonne

In [18]:

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

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

## Selection par valeur

In [19]:

```
1 print ("Pour une valeur :")
2 print (df.loc[df['Colonne']==1])
3
4 print ("\nEn prenant plusieurs valeurs avec isin :")
5 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 [20]:

```
1 print ('Tri par ordre décroissant :')
2 print(df.sort_values(by='Colonne',ascending=False))
3
4 print ('\nTri par ordre croissant (par défaut) : ')
5 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 [21]:

```
1 print ("Moyenne de la colonne : ")
2 print(df['Colonne'].mean())
3
4 print ('\nMaximum de la colonne :')
5 print(df['Colonne'].max())
6
7 print ('\nMinimul de la colonne :')
8 print(df['Colonne'].min())
9
10 print ("\nComptage des différentes valeurs de la colonne :")
11 print(df['Colonne'].value_counts())
12 print ("\nAjout d'une nouvelle ligne avec 5 pour vérifier : ")
13 df.loc[len(df)] = [5]
14 print (df)
15 print(df['Colonne'].value_counts())
16 df=df.drop(df.index[-1])
17
```

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 [22]:

```
1  # Ajout d'une colonne avec 1 comme valeur
2  df['Nouvelle Colonne'] = 1
3  print (df)
4
5  print ("\nSélection par valeur sur plusieurs colonnes avec un ET : ")
6  print (df.loc[(df['Colonne']==3) & (df['Nouvelle Colonne']==1)])
7
8  print ("\nSélection par valeur sur plusieurs colonnes avec un OU : ")
9  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 [23]:

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

Out[23]:

	Colonne	Nouvelle Colonne
0	1	6
1	2	6
2	3	6
3	4	6
4	5	6

Supression d'une colonne

In [24]:

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

Out[24]:

Colonne	
0	1
1	2
2	3
3	4
4	5

## Ajout d'une ligne

In [25]:

```
1  print ("Ajout d'une ligne à la fin : ")
2  df.loc[len(df)] = [6]
3  print (df)
4
5  print ("\nAjout d'une ligne au début attention il faut reorganiser les index")
6  df.loc[-1]=[7]
7  df.index = df.index + 1 # reorganiser les index
8  df = df.sort_index() # trier les index
9  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 [26]:

```
1 print (df)
2 print ("Modification de la valeur de la troisième ligne")
3 df.loc[3] = [10]
4 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 [27]:

```
1 print ("En utilisant une condition sur la colone : \n")
2 print ("Avant ",df)
3 df=df[df['Colonne']!=3]
4 print ("\nAprès ",df)
5
6 print ("\nEn utilisant les index (suppression de la dernière ligne) : ")
7 df=df.drop(df.index[-1])
8 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 [28]:

```
1 import numpy as np
2 print ("Ajout d'une ligne à la fin ne contenant rien (utilisant de numpy na
3 df.loc[len(df)] = [np.nan]
4 print (df)
5
6 print ("\nSuppression des lignes n'ayant pas de valeur : ")
7 df=df.dropna()
8 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 [29]:

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

Out[29]:

	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 [30]:

```
1  ▼ #il est possible de changer les noms ou valeurs des index
2    d = [1,2,3,4,5]
3
4    df = pd.DataFrame(d,columns = ['Colonne'])
5    print ("Dataframe initial :\n ",df)
6    i = [100,200,300,400,500]
7    df.index = i
8    print ("\nDataframe en changeant de valeur d'index : \n",df)
9
10   i = ['a','b','c','d','e']
11   df.index = i
12   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 [31]:

```
1  ▼ def multiplication (x):
2      return 100*x
3
4      print (df['Colonne'].apply(multiplication))
```

a	100
b	200
c	300
d	400
e	500

Name: Colonne, dtype: int64

## Boucler sur les colonnes

In [32]:

```
1  ▼ df = pd.DataFrame(  
2  ▼     {'Nom': ['Pierre', 'Paul', 'Jean', 'Michel'],  
3     'Age': [23, 22, 23, 20],  
4  ▼     'Note' : [15, 13, 14, 16]},  
5     index = ['i1', 'i2', 'i3', 'i4'])  
6  
7
```

In [33]:

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

```
int64  
object  
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 [34]:

```
1 print ("Dataframe initial : \n")
2 print (df)
3
4 print ("\nDataframe trié par Age : \n")
5 print (df.sort_values(by=['Age'], ascending=True))
6
7 print ("\nDataframe trié par Age et Note : \n")
8 print (df.sort_values(by=['Age', 'Note'],
9                       ascending=True))
```

Dataframe initial :

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

Dataframe trié par Age :

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

Dataframe trié par Age et Note :

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

## Groupby

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

In [35]:

```
1  ▼ # Definition du groupby sur la colonne note
2    g = df.groupby('Note')
3
4    # Il est possible de faire une boucle sur toutes les partitions
5  ▼ for groupe in g:
6      #groupe est un tuple
7      print(groupe[0]) # valeur du partitionnement
8      # Affichage des valeurs
9      print(groupe[1])
10     print ("\n")
11
```

```
13
      Age    Nom  Note
i2    22   Paul   13
```

```
14
      Age    Nom  Note
i3    23   Jean   14
```

```
15
      Age    Nom  Note
i1    23  Pierre   15
```

```
16
      Age    Nom  Note
i4    20  Michel   16
```

## Travailler avec plusieurs dataframes

### Concaténation

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

In [36]:

```
1  ▼ df = pd.DataFrame(
2  ▼     {'Nom': ['Pierre', 'Paul', 'Jean', 'Michel'],
3       'Age': [25, 32, 43, 60],
4       'Note': [14, 13, 14, 16],
5  ▼     'Sujet_id': [5, 3, 1, 4]},
6     index = ['i1', 'i2', 'i3', 'i4'])
7
8  ▼ df2 = pd.DataFrame(
9  ▼     { 'Sujet_id': [1, 2, 3, 4],
10       'Libelle': ['Math', 'Informatique', 'Physique', 'Chimie']})
11
12     print ('Dataframe 1 : \n',df)
13     print ('\nDataframe 1 : \n',df2)
```

```

14 print ( '\nDataframe 1 : \n', df1)
15 print ( '\nConcaténation de deux dataframes en ligne : ')
16 print (pd.concat([df, df2]))
17
18 print ( '\nConcaténation de deux dataframes en colonne : ')
19 print (pd.concat([df, df2], axis=1))
20

```

Dataframe 1 :

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

Dataframe 1 :

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

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 :

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

## Jointure

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

In [37]:

```
1 print ("\nJointure de deux dataframes en fonction de Sujet_id : ")
2 print (pd.merge(df, df2, on='Sujet_id', how='inner'))
3
4 print ("\nJointure externe (outer join) de deux dataframes en fonction de S
5 print (pd.merge(df, df2, on='Sujet_id', how='outer'))
6
7
8 print ("\nJointure externe droite (right outer join) : \n")
9 print (pd.merge(df, df2, on='Sujet_id', how='right'))
10
11 print ("\nJointure externe gauche (left outer join) : \n")
12 print (pd.merge(df, df2, on='Sujet_id', how='left'))
13
14
```

Jointure de deux dataframes en fonction de Sujet\_id :

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

Jointure externe (outer join) de deux dataframes en fonction de Sujet\_id :

	Age	Nom	Note	Sujet_id	Libelle
0	25.0	Pierre	14.0	5	NaN
1	32.0	Paul	13.0	3	Physique
2	43.0	Jean	14.0	1	Math
3	60.0	Michel	16.0	4	Chimie
4	NaN	NaN	NaN	2	Informatique

Jointure externe droite (right outer join) :

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

Jointure externe gauche (left outer join) :

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

## Sauvegarde des dataframes

Un dataframe peut être sauvegardé dans un fichier CSV.

In [38]:

```
1 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 défaut 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 [39]:

```
1 df = pd.DataFrame(  
2     {'Nom': ['Pierre', 'Paul', 'Jean', 'Michel'],  
3     'Age': [25, 32, 43, 60],  
4     'Note': [14, 13, 14, 16],  
5     'Sujet_id': [5, 3, 1, 4]},  
6     index = ['i1', 'i2', 'i3', 'i4'])  
7  
8 import sys  
9 print ('Affichage du fichier sauvegardé sur stdout \n')  
10 df.to_csv(sys.stdout)  
11  
12 print ('\nAffichage du fichier sauvegardé avec tabulation \n')  
13 df.to_csv(sys.stdout, sep='\t')  
14  
15 print ('\nAffichage du fichier sauvegardé avec tabulation sans header\n')  
16 df.to_csv(sys.stdout, sep='\t', header=False)  
17  
18  
19 print ('\nAffichage du fichier sauvegardé avec tabulation sans index \n')  
20 df.to_csv(sys.stdout, sep='\t', index=False)  
21  
22 print ('\nSauvegarde du fichier monfichier.csv \n')  
23 df.to_csv('monfichier.csv', sep='\t', index=False)  
24  
25 print ('\nLecture pour vérification \n')  
26 df = pd.read_csv('monfichier.csv', sep='\t')  
27 print (df)
```

Affichage du fichier sauvegardé sur stdout

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

Affichage du fichier sauvegardé avec tabulation

	Age	Nom	Note	Sujet_id
i1	25	Pierre	14	5

i2	32	Paul	13	3
i3	43	Jean	14	1
i4	60	Michel	16	4

Affichage du fichier sauvegardé avec tabulation sans header

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

Affichage du fichier sauvegardé avec tabulation sans index

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

Sauvegarde du fichier monfichier.csv

Lecture pour vérification

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