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.

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 [1]:

```
import pandas as pd
```

Exemples de création de séries

In [2]:

```
# 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)

# 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}
```

```
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]:

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

In [4]:

```
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 [5]:

```
# 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 [6]:

```
# 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 [7]:

```
# 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 [8]:

```
import pandas as pd
```

Exemple de création de dataframe

In [9]:

```
# 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 typ
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
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 inde
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
print(df)

# Création d'un dataframe à partir d'une liste de dictionnaires
df = pd.DataFrame(
    [{'a':10, 'b':15,'c':30,'d':40 },
     {'a':25, 'b':32, 'd':60}])
print ("\nUn dataframe pandas à partir d'une liste de dictionnaires :")
print(df)

# Création d'un dataframe à partir d'une liste de dictionnaires en renomma
df = pd.DataFrame(
    [{'a':10, 'b':15,'c':30,'d':40 },
     {'a':25, 'b':32, 'd':60}],index=['premier', 'second'])
print ("\nNoter le NaN (Not a Numeric number) quand il n'y a pas de valeur
```

```
69  print ("\nUn dataframe pandas à partir d'une liste de dictionnaires en ren
70  print(df)
71
72  # Création d'un dataframe à partir d'une liste de dictionnaires et sélecti
73  data=[{'a':10, 'b':15,'c':30,'d':40 },
74       {'a':25, 'b':32, 'd':60}]
75  df = pd.DataFrame(data,index=['premier', 'second'],columns=['a','d'])
76  print ("\nNoter le NaN (Not a Numeric number quand il n'y a pas de valeur"
77  print ("\nUn dataframe pandas à partir d'une liste de dictionnaires en séle
78  print(df)
79
80
```

```
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 ty
page :
        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 inde
x :
    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 renomma
nt 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 sélecti
onnant 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 [10]:

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

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

```

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]:

```
#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 [12]:

```
# 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 attribut
print (df)


# lecture du fichier en mettant des noms aux colonnes
df = pd.read_csv('exemple.csv',sep=';',skiprows=1,names=['Nom','Age','Note
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 [13]:

```
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 :

     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]:

```
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 nom pas au nom de l'inde
print ("\nLes lignes 2 et 3 du dataframe avec iloc : ")
print (df.iloc[1:3])
```

```
La ligne corrspondant 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]:

```
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.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]:

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

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 ('\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)
```

```
19
20  print ("\nNoms des colonnes\n")
21  print (df.columns)
22  print ('\nNom des index\n')
23  print (df.index)
24  print ('\nStatistiques élémentaires\n')
25  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      Spe
cies
0            5.1           3.5            1.4           0.2  Iris-se
tosa
1            4.9           3.0            1.4           0.2  Iris-se
tosa

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

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', 'PetalWidth
Cm',
       '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]:

```
#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 [18]:

```
# 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 [19]:

```
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 [20]:

```
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 [21]:

```
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 [22]:

```
# 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 [23]:

```
# 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[23]:

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

Supression d'une colonne

In [24]:

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

Out[24]:

	Colonne
0	1
1	2
2	3
3	4
4	5

Ajout d'une ligne

In [25]:

```
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 inde
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 [26]:

```
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 [27]:

```
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 [28]:

```
import numpy as np
print ("Ajout d'une ligne à la fin ne contenant rien (utilisant de numpy na
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 [29]:

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

```
#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",
```

```
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]:

```
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 [32]:

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


```

In [33]:

```
# il est possible de boucler sur les colonnes
for col in df.columns:
                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]:

```
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 :

    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]:

```
# 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
    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]:

```
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 ('\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 :
     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]:

```
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 S
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 :
   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 Suje
t_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 [39]:

```
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 [40]:

```
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

,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
```

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.

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 [1]:

Exemples de création de séries

In [2]:

```
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
```

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]:

In [4]:

```
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]:

```
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]:

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

Il est possible d'indexer plusieurs valeurs

In [7]:

```
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 [8]:

Exemple de création de dataframe

In [9]:

```
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

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 [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]:

In [12]:

```
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.
```

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]:

```
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]:

```
La ligne corrspondant 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 :
```

Il est possible de spécifier les colonnes dans le résultat

In [15]:

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.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]:

```
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      Spe
cies
0            5.1           3.5            1.4           0.2  Iris-se
.
```

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]:

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

Changement du nom de colonne

In [18]:

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

Selection par valeur

In [19]:

```
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]:

```
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]:

```
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
```

Ajout d'une colonne

In [22]:

```
   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]:

Out[23]:

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

Supression d'une colonne

In [24]:

Out[24]:

	Colonne
0	1
1	2
2	3
3	4
4	5

Ajout d'une ligne

In [25]:

```
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]:

```
   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]:

```
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
```

Suppression d'une ligne dont la valeur est NaN

In [28]:

```
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]:

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]:

```
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
```

Application d'une fonction à un dataframe

In [31]:

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

Boucler sur les colonnes

In [32]:

In [33]:

```
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]:

```
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
```

Groupby

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

In [35]:

```
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]:

```
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
```

Jointure

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

In [37]:

```
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 Suje
t_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
```

Sauvegarde des dataframes

Un dataframe peut être sauvegardé dans un fichier CSV.

In [38]:

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent cal
l last)
<ipython-input-38-97765b5cee23> in <module>()
----> 1 df = pd.to_csv('myFile.csv')

AttributeError: module 'pandas' has no attribute 'to_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 [ ]: