Introduction à Pandas & tips associés !

Import et création de série :

```
>>> import pandas as pd
>>> import numpy as np
>>> series = pd.Series([1,2,3,4,5, np.nan, "a string", 6])
>>> series
0           1
1           2
2           3
3           4
4           5
5         NaN
6    a string
7           6
 dtype: object
```

Formats divers en série :

```
>>> series = pd.Series([1,2,np.nan, 4])
>>> series
0    1.0
1    2.0
2    NaN
3    4.0
 dtype: float64
```

De la serie au Dataframe :

```
>>> df = pd.DataFrame(np.array([1,2,3,4,5,6]).reshape(2,3))
>>> df
   0  1  2
0  1  2  3
1  4  5  6

>>> df.dtypes
0    int32
1    int32
2    int32
 dtype: object
```

Paramétrer son Dataframe :

```
>>> df = pd.DataFrame(np.array([1,2,3,4,5,6]).reshape(2,3),
columns=list('ABC'), index=list('XY'))

>>> df
  A  B  C
X  1  2  3
Y  4  5  6
```

Options d'affichage du Dataframe :

```
>>> df2 = pd.DataFrame(np.arange(1, 7501).reshape(500,15))
>>> df2.head(2)
  0   1   2   3   4   5   6   7   8   9   10  11  12  13  14
0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
1  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30

>>> df2.head()
  0   1   2   3   4   5   6   7   8   9   10  11  12  13  14
0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
1  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30
2  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45
3  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60
4  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75

>>> df2.tail(1)
      0     1     2     3     4     5     6     7     8     9     10    11  \
499  7486  7487  7488  7489  7490  7491  7492  7493  7494  7495  7496  7497

      12    13    14
499  7498  7499  7500
```

Statistique descriptive du Dataframe :

```
>>> df3 = pd.DataFrame(np.arange(1, 100, 0.12).reshape(33,25))
>>> df3.describe()
 0           1           2           3           4           5  \
count  33.000000  33.000000  33.000000  33.000000  33.000000  33.000000
mean   49.000000  49.120000  49.240000  49.360000  49.480000  49.600000
std    29.008619  29.008619  29.008619  29.008619  29.008619  29.008619
min     1.000000   1.120000   1.240000   1.360000   1.480000   1.600000
25%    25.000000  25.120000  25.240000  25.360000  25.480000  25.600000
50%    49.000000  49.120000  49.240000  49.360000  49.480000  49.600000
75%    73.000000  73.120000  73.240000  73.360000  73.480000  73.600000
max    97.000000  97.120000  97.240000  97.360000  97.480000  97.600000
```

Indexation et selection du Dataframe :

```
>>> df3 = pd.DataFrame(np.arange(1, 100, 0.12).reshape(33,25))
>>> df3.iloc[:5,:10] # 5 premières lignes/ 10 première colonnes
     0     1     2     3     4     5     6     7     8     9
0   1.0   1.12  1.24  1.36  1.48  1.6   1.72  1.84  1.96  2.08
1   4.0   4.12  4.24  4.36  4.48  4.6   4.72  4.84  4.96  5.08
2   7.0   7.12  7.24  7.36  7.48  7.6   7.72  7.84  7.96  8.08
3  10.0  10.12 10.24 10.36 10.48 10.6  10.72 10.84 10.96 11.08
4  13.0  13.12 13.24 13.36 13.48 13.6  13.72 13.84 13.96 14.08

>>> df3.iloc[-5:] # idem → df3.tail(5)
 0     1     2     3     4     5     6     7     8  9   ...    \
28  85.0  85.12 85.24 85.36 85.48 85.6  85.72 85.84 85.96 86.08 ...
29  88.0  88.12 88.24 88.36 88.48 88.6  88.72 88.84 88.96 89.08 ...
30  91.0  91.12 91.24 91.36 91.48 91.6  91.72 91.84 91.96 92.08 ...
31  94.0  94.12 94.24 94.36 94.48 94.6  94.72 94.84 94.96 95.08 ...
32  97.0  97.12 97.24 97.36 97.48 97.6  97.72 97.84 97.96 98.08 ...
```

```
df4 = df3.rename(columns=lambda c: chr(65+c))
```

```
>>> df4.loc[:5, 'A':'D'] # sélection des lignes <=5 & des colonnes de A à D
     A      B      C      D
0   1.0   1.12   1.24   1.36
1   4.0   4.12   4.24   4.36
2   7.0   7.12   7.24   7.36
3  10.0  10.12  10.24  10.36
4  13.0  13.12  13.24  13.36
5  16.0  16.12  16.24  16.36

>>> df4.loc[:5, ('A','D')] # sélection des lignes <=5 & des colonnes de A et
D
     A      D
0   1.0   1.36
1   4.0   4.36
2   7.0   7.36
3  10.0  10.36
4  13.0  13.36
5  16.0  16.36
```

`df.loc[:,['A', 'C']]` : toutes les lignes et seulement les colonnes A et B.

`df.loc['a2', 'C']` : accès à la valeur de la ligne a2 et de la colonne C.

`df.loc[:,['A', 'b']].iloc[0:2]` : Accès à certaines colonnes et certaines lignes par numéros.

`df.loc[df.index[3], 'A']` etc..

CSV & Dataframe :

```
>>> baby_names = pd.read_csv('baby_names.csv')
>>> baby_names.head()
  BRTH_YR    GNDR    ETHCTY          NM  CNT  RNK
0    2011  FEMALE  HISPANIC  GERALDINE   13   75
1    2011  FEMALE  HISPANIC         GIA   21   67
2    2011  FEMALE  HISPANIC      GIANNA   49   42
3    2011  FEMALE  HISPANIC     GISELLE   38   51
4    2011  FEMALE  HISPANIC       GRACE   36   53
```

`df.to_csv('myFile.csv', sep = '\t')` : écrit le dataframe avec une tabulation comme séparateur (le défaut est une virgule).

Tri du Dataframe :

```
>>> baby_names.sort_values(by='CNT', ascending=False).head()

     BRTH_YR  GNDR    ETHCTY      NM  CNT  RNK
1504    2011  MALE  HISPANIC  JAYDEN  426    1
5430    2011  MALE  HISPANIC  JAYDEN  426    1
7393    2011  MALE  HISPANIC  JAYDEN  426    1
3505    2011  MALE  HISPANIC  JAYDEN  426    1
9385    2012  MALE  HISPANIC  JAYDEN  364    1
```

`df[df['A'] > 2]` : attention, ce n'est pas une copie qui est renvoyée, mais

une vue, donc, on ne peut pas modifier le résultat !

`df.loc[df['A'] > 2,:]` : c'est une copie qui est renvoyée ici.

`df[df['A'].isin([5.3, 2.7])]` : renvoie un dataframe avec seulement les lignes où la valeur de A est parmi celles listées.

Opérations sur Dataframe :

`df.dropna(how = 'any')` ou `df.dropna()` : renvoie un dataframe avec les lignes contenant au moins une valeur NaN supprimée (how = 'all' : supprime les lignes où toutes les valeurs sont NaN).

`df.dropna(axis = 1, how = 'any')` : supprime les colonnes ayant au moins un NaN plutôt que les lignes (le défaut est axis = 0).

`df.dropna(inplace = True)` : ne renvoie rien, mais fait la modification en place.

`df.fillna(0)` : renvoie un dataframe avec toutes les valeurs NaN remplacées par 0.

`df['A'].fillna(0, inplace = True)` : remplace tous les NA de la colonne A par 0, sur place.

`df.isnull()` : renvoie un dataframe de booléens, avec True dans toutes les cellules non définies.

Copie du Dataframe :

`df2 = df.copy()` : df2 est alors un dataframe indépendant.par contre, si on fait : `df2 = df` et que l'on modifie df2, df est également modifié (df et df2 pointent vers le même objet).

Gestion des doublons du Dataframe :

`df.drop_duplicates()` : renvoie un dataframe avec les lignes redondantes enlevées en n'en conservant qu'une seule.

`df.drop_duplicates(subset = ['A', 'B'], keep = 'last')` : on conserve la dernière ligne plutôt que la première (keep = first, qui est le défaut).

Fonction sur Dataframe :

si `df = pandas.DataFrame({'A': [1, 2, 3], 'B': [9, 8, 7]})` :

	A	B
0	1	9
1	2	8
2	3	7

alors `df.apply(lambda x: x + 1)` renvoie :

	A	B
0	2	10
1	3	9
2	4	8

(apply prend une fonction qui prend en argument une série)

on peut aussi appeler une fonction qui calcule un agrégat : `df.apply(lambda x: x.max())`

```
A    3
B    9
```

Concaténation & jointure :

on peut concaténer ainsi des dataframes même si les index sont identiques, ils seront juxtaposés avec des valeurs d'index répétées :

```
df1 = pandas.DataFrame({'A': [3, 5], 'B': [1, 2]})
df2 = pandas.DataFrame({'A': [6, 7], 'B': [4, 9]})
```

`pandas.concat([df1, df2])`:

```
   A  B
0  3  1
1  5  2
0  6  4
1  7  9
```

si les dataframes n'ont pas les mêmes colonnes et qu'on veut conserver seulement les colonnes communes, intersection (sans avoir de NaN) :

`pandas.concat([df1, df2], join = 'inner')`:

```
   A
0  3
1  5
0  6
1  7
```

Eviter de faire des concaténations répétées, préférer construire une liste de dataframes et faire une seule concaténation, pour des raisons de performances.

Pour les jointures, le principe est le même qu'en SQL :

inner : jointure interne pour retourner les enregistrements quand la condition est vrai dans les 2 df. C'est l'une des jointures les plus communes.

outer : jointure externe pour retourner les résultats quand la condition est vrai dans au moins une des 2 df.

left : jointure externe pour retourner tous les enregistrements de la table de gauche (LEFT = gauche) même si la condition n'est pas vérifié dans l'autre df.

right : jointure externe pour retourner tous les enregistrements de la table de droite (RIGHT = droite) même si la condition n'est pas vérifié dans l'autre df.

Groupements et Agrégats du Dataframe :

On peut grouper un dataframe par une ou plusieurs colonne.
Si `df = pandas.DataFrame({'A': ['a', 'b', 'a', 'a', 'b'], 'B': [8, 4, 5, 10, 8], 'C': ['x', 'x', 'y', 'y', 'x'], 'D': [0, 1, 2, 3, 4]})` :

`df.groupby('A')` : renvoie un objet de la classe pandas.core.groupby.DataFrameGroupBy.

`df.groupby('A').sum()` : groupe avec les valeurs de A et fait la somme, pour les colonnes pour lesquelles c'est possible :

```
    B  D
A
a  23  5
b  12  5
```

On peut inclure plusieurs colonnes :

`df.groupby(['A', 'C']).sum()`:

```
      B  D
A C
a x   8  0
  y  15  5
b x  12  5
```

`df.groupby('A').mean()` : marche même si certaines cellules sont vides.