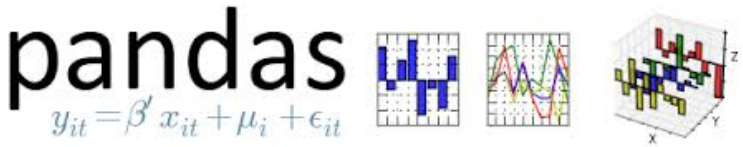
Présentation de la bibliothèque Pandas

Fedi Wannes

ERM Partners®
Risk Management, Actuariat



Décembre 2015

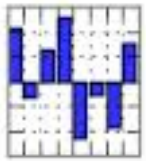
Plan

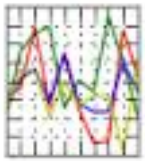
Plan

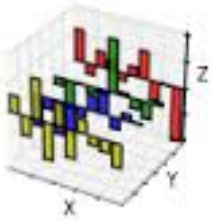
Introduction

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$







Caractéristiques

- Bibliothèque écrite pour le langage de programmation Python
- Permettre la manipulation et l'analyse des données
- Logiciel libre

Introduction

Principales structures de données

- Séries
- DataFrames
- Panels

Plan

Fonctionnalités

Points forts

- Manipuler des données aisément et efficacement
- Lire et écrire des données depuis fichiers CSV,tableurs Excel,...
- Fusion et jointure de large volume de données
- Représentation graphique

Plan

Importation des bibliothèques

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Habituellement on importe :

- **Pandas**
- **Numpy** : qui contient des fonctions mathématiques de haut niveau
- **Matplotlib** : Pour la représentation graphique

Création des objets

Création Series

Création d'une série en passant une liste de valeurs.

```
s = pd.Series([1,3,5,np.nan,6,8])
print s
```

```
>pythonw -u "Series.py"
0    1
1    3
2    5
3  NaN
4    6
5    8
dtype: float64
>Exit code: 0
```

Création des objets

Création DataFrames

Création d'une DataFrame en passant un tableau numpy,avec un indice de datetime et colonnes intitulées.

```
dates = pd.date_range('20130101', periods=6)
#6 jours a partir de 2013-01-01
df = pd.DataFrame(np.random.randn(6,4),
index=dates, columns=list('ABCD'))

print df
```

```
>pythonw -u "DataFrames.py"
                   A         B         C         D
2013-01-01  1.303875  0.013666  0.893579  0.353337
2013-01-02  0.092020 -0.651396  1.041044 -0.085049
2013-01-03 -0.452200 -0.587162 -0.602563  0.709463
2013-01-04 -0.345419 -0.714958 -0.570235 -1.189434
2013-01-05 -0.442426  0.954329 -1.182346  1.020836
2013-01-06  1.472266 -0.164019  0.333764  0.323356
>Exit code: 0
```

Création des objets

Création DataFrames

Création d'une trame de données en passant un dictionnaire d'objets.

```
df2 = pd.DataFrame({ 'A' : 1.,
'B' : pd.Timestamp('20130102'),
'C' : pd.Series(1,index=list(range(4)),dtype='float32'),
'D' : np.array([3] * 4,dtype='int32'),
'E' : pd.Categorical(["test","train","test","train"]),
'F' : 'foo' })
print df2
```

```
>pythonw -u "dataframe-dict.py"
  A       B  C  D      E    F
0  1 2013-01-02  1  3   test  foo
1  1 2013-01-02  1  3  train  foo
2  1 2013-01-02  1  3   test  foo
3  1 2013-01-02  1  3  train  foo
>Exit code: 0
```

Plan

Visualisation des données

Méthodes utiles

- **df.head(),df.tail()** : Voir les lignes du haut et du bas du table
- **df.index** : Afficher l'index
- **df.columns** : Afficher les colonnes
- **df.values** : Afficher les valeurs
- **df.describe()** : Faire une statistique sommaire rapide des données
- **df.T** : Transposer les données
- **df.sort_index(axis=1, ascending=False)** : Tri par un axe
- **df.sort_values(by='B')** : Tri par valeurs

Sélection des données

Sélection des colonnes

Pour sélectionner une seule colonne on utilise **df['nom_colonne']**
Par exemple : **df['B']**

```
>pythonw -u "selection_données.py"
0   2013-01-02
1   2013-01-02
2   2013-01-02
3   2013-01-02
Name: B, dtype: datetime64[ns]
>Exit code: 0
```

Sélection des données

Sélection des lignes

Pour sélectionner plusieurs colonnes on utilise **df[debut :fin]**
Par exemple : **df[0 :3]**

```
>pythonw -u "selection_données.py"
  A       B  C  D    E    F
0  1 2013-01-02  1  3   test  foo
1  1 2013-01-02  1  3  train  foo
2  1 2013-01-02  1  3   test  foo
>Exit code: 0
```

Sélection des données

Sélection des parties

Pour obtenir une coupe de données à l'aide d'une étiquette
Par exemple :**df.loc[dates[0]]**

```
>pythonw -u "selection_données.py"
                   A         B         C         D
2013-01-01 -0.279440  0.096331  0.725737 -0.052774
2013-01-02  0.739814 -0.266202  0.639459 -2.827497
2013-01-03 -0.174948  0.217638 -1.867000  1.051235
2013-01-04  0.521292 -1.044204 -0.903881 -0.283809
2013-01-05 -2.009349  0.008180  0.446847  0.183903
2013-01-06 -0.510199 -0.418709  0.856372  1.507790

df.loc[dates[0]]=
 A   -0.279440
B    0.096331
C    0.725737
D   -0.052774
```

Sélection des données

Sélection sur un multi-axe

Exemple : **df.loc['20130102' :'20130104',['A','B']]**

```
>pythonw -u "selection_données.py"
                   A         B         C         D
2013-01-01  1.685517  0.332647  0.094832 -0.181480
2013-01-02  0.337738  1.078385 -1.684338 -0.376405
2013-01-03 -0.039757  1.325646  1.393760 -0.181452
2013-01-04  0.785178  0.081069  1.346114 -1.216195
2013-01-05  1.359535  0.891112 -0.074281  0.537419
2013-01-06  0.194109  1.773524 -0.226001 -0.642103

df.loc['20130102':'20130104',['A','B']]=
                   A         B
2013-01-02  0.337738  1.078385
2013-01-03 -0.039757  1.325646
2013-01-04  0.785178  0.081069
>Exit code: 0
```

Sélection des données

Sélection par position

Exemple : **df.iloc[[1,2,4],[0,2]]**
[1,2,4]= lignes 1,2 et 4
[0,2]= colonnes 0 et 2 (A,C)

```
>pythonw -u "selection_données.py"
                   A         B         C         D
2013-01-01 -0.971264 -0.143694 -1.129902  0.355884
2013-01-02 -0.856211  0.520039 -0.520338 -0.433769
2013-01-03 -0.458965 -0.352665 -1.642765 -0.018014
2013-01-04  0.988710  0.405008 -3.095456 -1.910941
2013-01-05 -0.319650 -0.962238 -0.697276  0.071899
2013-01-06  0.081776  0.883352 -0.243186 -0.827238
                   A         C
2013-01-02 -0.856211 -0.520338
2013-01-03 -0.458965 -1.642765
2013-01-05 -0.319650 -0.697276
>Exit code: 0
```

indexation booléenne

Exemple

df[df $>$ 0] : éliminer les valeurs inférieures à 0

```
>pythonw -u "selection_données.py"
                   A         B         C         D
2013-01-01  2.009467  0.099382  0.351986  0.011063
2013-01-02 -0.694066 -0.381292 -0.449137  1.383951
2013-01-03 -0.307665  0.324283  1.086667 -0.426112
2013-01-04 -2.011102 -1.178102 -0.088923 -1.108444
2013-01-05  0.941298  0.553681 -0.983771  2.433335
2013-01-06  0.352260 -0.089108 -0.324905  1.781706
                   A         B         C         D
2013-01-01  2.009467  0.099382  0.351986  0.011063
2013-01-02       NaN       NaN       NaN  1.383951
2013-01-03       NaN  0.324283  1.086667       NaN
2013-01-04       NaN       NaN       NaN       NaN
2013-01-05  0.941298  0.553681       NaN  2.433335
2013-01-06  0.352260       NaN       NaN  1.781706
>Exit code: 0
```

Filtration des données

Exemple

df2[df2['E'].isin(['two','four'])]

```
df2 = df.copy()#Faire une copie de df
df2['E'] = ['one', 'one','two','three','four','three']
#Ajout d'une colonne
print df2
print "isin result=", df2[df2['E'].isin(['two','four'])]
```

```
>pythonw -u "isin-filtration.py"
                   A         B         C         D      E
2013-01-01 -0.842503  0.703442  1.613971 -1.923771    one
2013-01-02  0.937882  1.399504  0.017399 -0.355362    one
2013-01-03  0.359068  1.912678  0.545076 -0.278817    two
2013-01-04 -0.645973  0.794910 -0.636948 -0.301719  three
2013-01-05 -2.017871  1.397934 -1.482305 -1.040118   four
2013-01-06  1.084484  1.507472 -0.353257  0.291928  three
isin result=
                   A         B         C         D     E
2013-01-03  0.359068  1.912678  0.545076 -0.278817   two
2013-01-05 -2.017871  1.397934 -1.482305 -1.040118  four
>Exit code: 0
```

Application des fonctions aux données

Exemple

```
df.apply(lambda x: x.max() - x.min())
print df3
```

```
>pythonw -u "selection_données.py"
                   A         B         C         D
2013-01-01  0.656104 -0.900623  0.560918 -0.303748
2013-01-02  0.840643  0.196769 -0.764356  0.348011
2013-01-03  0.231195 -0.618192 -1.255859  1.478399
2013-01-04  0.647821  0.394618  0.568641 -2.368887
2013-01-05  1.103911  0.114164 -0.421129 -0.874519
2013-01-06  1.255406  0.614916  0.239951  1.019650
A   1.024210
B   1.515539
C   1.824500
D   3.847286
dtype: float64
>Exit code: 0
```

Fusionnement

Méthode concat

```
df = pd.DataFrame(np.random.randn(10, 4))
Pieces = [df[:3], df[3:7], df[7:]]
#couper en sous tables
df2=pd.concat(pieces)#enchaînement
```

```
>pythonw -u "selection_données.py"
      0         1         2         3
0 -0.449318 -1.819485 -1.780742  1.239416
1 -0.273314  1.253514  1.249525 -1.591887
2 -1.195855 -0.163541  0.400275 -1.596917
3 -0.990186  0.492572  0.623607  0.139897
4 -2.331856  0.590736  0.545794 -0.232740
5  0.364708 -1.284259  1.256362  0.922255
6  0.386371 -0.058883  0.578478 -0.529081
7 -0.213919 -0.549789  0.883591  0.320393
8 -0.416455  0.066634  0.717787  1.171424
9  0.417612 -1.165331 -0.292413 -0.073338
>Exit code: 0
```

Jointures

Méthode Merge

merge(left, right, how='inner', on=None, left_on=None, right_on=None, left_index=False, right_index=False, sort=True, suffixes=('_x', '_y'), copy=True, indicator=False)

Jointures

Exemple

```
right = pd.DataFrame({'key': ['foo', 'foo'],
'rval': [4, 5]})
left = pd.DataFrame({'key': ['foo', 'foo'],
'lval': [1, 2]})
df=pd.merge(left, right, on='key')
```

```
>pythonw -u "selection_données.py"
-------right--------
  key  rval
0 foo     4
1 foo     5
-------left---------
  key  lval
0 foo     1
1 foo     2
-------result ---------
  key  lval  rval
0 foo     1     4
1 foo     1     5
2 foo     2     4
3 foo     2     5
>Exit code: 0
```

Ajout des lignes

Méthode Append

```
df = pd.DataFrame(np.random.randn(8, 4),
columns=['A','B','C','D'])
s = df.iloc[3] #prend une copie de la ligne 3
df.append(s, ignore_index=True)
#Ajout d'une ligne a la fin du DataFrame
```

```
>pythonw -u "selection_données.py"
df=
          A         B         C         D
0 -1.856953  1.761762  1.046424  0.680780
1 -0.097877 -0.234265  0.531393  0.969157
2  2.456011  0.227842  1.092999 -1.611563
3  1.516697 -1.895093  0.323395 -0.427653
4 -0.630770 -0.078677 -0.582348  1.541813
5  1.659195 -0.623955  0.136817 -0.615754
6  1.102998 -2.934579 -0.427926 -0.521744
7 -2.291191 -2.401457  0.312693 -1.153717

RESULT=
          A         B         C         D
0 -1.856953  1.761762  1.046424  0.680780
1 -0.097877 -0.234265  0.531393  0.969157
2  2.456011  0.227842  1.092999 -1.611563
3  1.516697 -1.895093  0.323395 -0.427653
4 -0.630770 -0.078677 -0.582348  1.541813
5  1.659195 -0.623955  0.136817 -0.615754
6  1.102998 -2.934579 -0.427926 -0.521744
7 -2.291191 -2.401457  0.312693 -1.153717
8  1.516697 -1.895093  0.323395 -0.427653
>Exit code: 0
```

Regroupement des données(1/2)

Group by

Diviser les données en groupes en fonction de certains critères.

```
df = pd.DataFrame({'A' : ['foo', 'bar', 'foo', 'bar','foo',
'B' : ['one', 'one', 'two', 'three',
'two', 'two', 'one', 'three'],
'C' : np.random.randn(8),
'D' : np.random.randn(8)})
print df
print df.groupby('A').sum()
```

Regroupement des données(2/2)

Résultat

```
>pythonw -u "selection_données.py"
   A      B      C        D
0  foo    one  0.546358  0.996651
1  bar    one  0.847766  1.276778
2  foo    two -1.108711 -2.021341
3  bar  three  1.293110 -1.019042
4  foo    two  0.257199  0.511371
5  bar    two -0.382047 -0.029779
6  foo    one  0.369605 -0.270834
7  foo  three  1.352416 -1.050615
df.groupby('A').sum()=

          C        D
A
bar  1.758829  0.227957
foo  1.416868 -1.834768
>Exit code: 0
```

Méthode stack (1/2)

Compresser les colonnes

La méthode stack () " comprime " un niveau dans les colonnes d'un DataFrame.L'inverse de la méthode stack est **unstack()**

Exemple :

```
tuples = list(zip(*[['bar', 'bar', 'baz', 'baz',
'foo', 'foo', 'qux', 'qux'],
['one', 'two', 'one', 'two',
'one', 'two', 'one', 'two']]))
index = pd.MultiIndex.
from_tuples(tuples, names=['first', 'second'])
df = pd.DataFrame(np.random.randn(8, 2), index=index,
columns=['A', 'B'])
df2 = df[:4]
stacked = df2.stack()
print df2
print stacked
```

Méthode stack (2/2)

Résultat

```
>pythonw -u "selection_données.py"
                A         B
first second
bar   one   -1.503647  0.132986
      two   -1.483768  0.895974
baz   one    0.506455  1.671557
      two   -0.130789 -0.284740
--------------------------------
first  second
bar    one     A  -1.503647
               B   0.132986
       two     A  -1.483768
               B   0.895974
baz    one     A   0.506455
               B   1.671557
       two     A  -0.130789
               B  -0.284740
dtype: float64
>Exit code: 0
```

Catégories

Exemple

Convertir les entrées en types de données catégoriques .

```
df = pd.DataFrame({"id":[1,2,3,4,5,6],
"raw_grade":['a', 'b', 'b', 'a', 'a', 'e']})
df["grade"] = df["raw_grade"].astype("category")
df["grade"].cat.categories =
["very_good", "good", "very_bad"]
print df.sort_values(by="grade")
```

```
>pythonw -u "selection_données.py"
  id raw_grade      grade
0  1         a  very good
3  4         a  very good
4  5         a  very good
1  2         b       good
2  3         b       good
5  6         e   very bad
>Exit code: 0
```

Pivoter les Tableaux(1/2)

Exemple

Une meilleure représentation là où les colonnes sont les variables uniques et l'index identifiant Pour remodeler les données dans ce forme, utilisez la fonction de pivot

```
Out[1]:
         date variable     value
0  2000-01-03        A  0.469112
1  2000-01-04        A -0.282863
2  2000-01-05        A -1.509059
3  2000-01-03        B -1.135632
4  2000-01-04        B  1.212112
5  2000-01-05        B -0.173215
6  2000-01-03        C  0.119209
7  2000-01-04        C -1.044236
8  2000-01-05        C -0.861849
9  2000-01-03        D -2.104569
10 2000-01-04        D -0.494929
11 2000-01-05        D  1.071804
```

```
df.pivot(index='date', columns='variable', values='value')
```

Pivoter les Tableaux(2/2)

Résultat

```
Out[3]:
variable           A         B         C         D
date
2000-01-03  0.469112 -1.135632  0.119209 -2.104569
2000-01-04 -0.282863  1.212112 -1.044236 -0.494929
2000-01-05 -1.509059 -0.173215 -0.861849  1.071804
```

Chaînes de caractères

Exemple

```
import pandas as pd
import numpy as np
s = pd.Series(['A', 'B', 'C', 'Aaba', 'Baca', np.nan,
'CABA', 'dog', 'cat'])
print s.str.lower()
```

```
>pythonw -u "10-String_Methods.py"
0      a
1      b
2      c
3   aaba
4   baca
5    NaN
6   caba
7    dog
8    cat
dtype: object
>Exit code: 0
```

Remodelage des Tableaux(1/2)

Exemple

```
import pandas as pd
import numpy as np
tuples = list(zip(*[['bar', 'bar', 'baz', 'baz'
,'foo', 'foo', 'qux', 'qux'],
['one', 'two', 'one', 'two','one', 'two', 'one', 'two']]))
print tuples
index = pd.MultiIndex.from_tuples(
tuples, names=['first', 'second'])
df = pd.DataFrame(np.random.randn(8, 2),
index=index, columns=['A', 'B'])
df2 = df[:4]
print df2
```

Remodelage des Tableaux(2/2)

Résultat

```
>pythonw -u "Reshaping.py"
[('bar', 'one'), ('bar', 'two'), ('baz', 'one'), ('baz', 'two'), ('foo', 'one'), ('foo', 'two'), ('qux', 'one'), ('qux', 'two')]
                A         B
first second
bar   one     1.141955  0.049717
      two     0.536495  1.561605
baz   one     2.907292  0.987574
      two    -0.580721 -0.098379
>Exit code: 0
```

Données manquantes

Exemple

```
import pandas as pd
import numpy as np
dates = pd.date_range('20130101', periods=6)
df = pd.DataFrame(np.random.randn(6,4), index=dates,
columns=list('ABCD'))
df1 = df.reindex(index=dates[0:4],
columns=list(df.columns) + ['E'])
df1.loc[dates[0]:dates[1],'E'] = 1
print df1
print df1.dropna(how='any')#Effacer les lignes avec
#~ donnee manquants
print df1.fillna(value=5)#Remplir les champs manquants
print pd.isnull(df1)#Masque booleen
```

Données manquantes

Résultat

```
>pythonw -u "Missing-Data.py"
                   A         B         C         D  E
2013-01-01 -1.664224 -0.201775  0.966145 -0.600721  1
2013-01-02 -0.787307 -0.268409 -0.557495 -1.385163  1
2013-01-03  0.414929  2.067612 -1.480687 -0.708598 NaN
2013-01-04  0.481365  1.017673  0.108631  2.759890 NaN
                   A         B         C         D  E
2013-01-01 -1.664224 -0.201775  0.966145 -0.600721  1
2013-01-02 -0.787307 -0.268409 -0.557495 -1.385163  1
                   A         B         C         D  E
2013-01-01 -1.664224 -0.201775  0.966145 -0.600721  1
2013-01-02 -0.787307 -0.268409 -0.557495 -1.385163  1
2013-01-03  0.414929  2.067612 -1.480687 -0.708598  5
2013-01-04  0.481365  1.017673  0.108631  2.759890  5
                A      B      C      D      E
2013-01-01  False  False  False  False  False
2013-01-02  False  False  False  False  False
2013-01-03  False  False  False  False   True
2013-01-04  False  False  False  False   True
>Exit code: 0
```

Histogrammes

Exemple

```
import pandas as pd
import numpy as np
s = pd.Series(np.random.randint(0, 7, size=10))
print s.value_counts()
```

```
>pythonw -u "Histogramming.py"
5   5
3   2
0   2
6   1
dtype: int64
>Exit code: 0
```

Plan

Series de dates

Traitements

pandas dispose d'une fonctionnalité simple, puissant et efficace pour réaliser des opérations de ré-échantillonnage lors de la conversion des Dates

Exemple :

```
rng = pd.date_range('1/1/2016', periods=100, freq='S')
#prendre 100 secondes
ts = pd.Series(np.random.randint
(0, 500, len(rng)), index=rng)
#prendre des valeurs aleatoires entre[0:500]
ts.resample('5Min', how='sum')
#somme des valeurs de Frequence 5 Min
```

```
>pythonw -u "selection_données.py"
2016-01-01   24808
Freq: 5T, dtype: int32
>Exit code: 0
```

Time zone

Exemple

```
rng = pd.date_range('20/6/2015 00:00', periods=5,freq='D')
ts = pd.Series(np.random.randn(len(rng)), rng)
ts_utc = ts.tz_localize('UTC')
print ts_utc
ts_utc2=ts_utc.tz_convert('US/Eastern')
print ts_utc2
```

```
>pythonw -u "selection_données.py"
2015-06-20 00:00:00+00:00  -0.107909
2015-06-21 00:00:00+00:00   1.112620
2015-06-22 00:00:00+00:00  -0.258292
2015-06-23 00:00:00+00:00   0.233670
2015-06-24 00:00:00+00:00   1.765891
Freq: D, dtype: float64
2015-06-19 20:00:00-04:00  -0.107909
2015-06-20 20:00:00-04:00   1.112620
2015-06-21 20:00:00-04:00  -0.258292
2015-06-22 20:00:00-04:00   0.233670
2015-06-23 20:00:00-04:00   1.765891
Freq: D, dtype: float64
>Exit code: 0
```

Conversion en périodes

Exemple

```
rng = pd.date_range('1/1/2012', periods=5, freq='M')
#5 mois a partir du 1/1/2012
ts = pd.Series(np.random.randn(len(rng)), index=rng)
ps = ts.to_period()#conversion en periodes
```

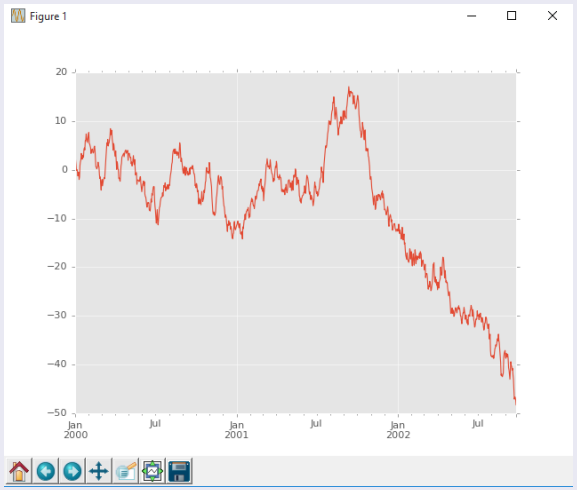
```
>pythonw -u "selection_données.py"
2012-01-31    0.200549
2012-02-29    0.271343
2012-03-31   -1.098638
2012-04-30    1.131335
2012-05-31    0.721582
Freq: M, dtype: float64
ts.to_period()=
2012-01    0.200549
2012-02    0.271343
2012-03   -1.098638
2012-04    1.131335
2012-05    0.721582
Freq: M, dtype: float64
>Exit code: 0
```

Plan

Traçage des courbes

Exemple

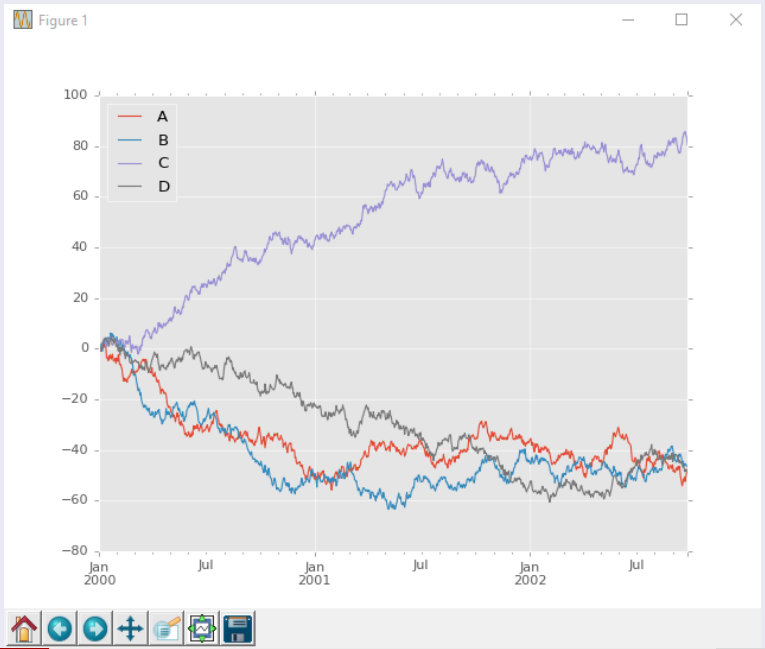
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
ts = pd.Series(np.random.randn(1000),
index=pd.date_range('1/1/2000', periods=1000))
ts = ts.cumsum()
ts.plot()
plt.show()
```



Traçage des colonnes

Exemple

```
ts = pd.Series(np.random.randn(1000),
index=pd.date_range('1/1/2000', periods=1000))
df = pd.DataFrame(np.random.randn(1000, 4),
index=ts.index, columns=list('ABCD'))
df = df.cumsum()
df.plot(); plt.show()
```



Plan

CSV

Lecture à partir d'un CSV

```
pd.read_csv('foo.csv')
```

Ecriture

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

EXCEL

Lecture à partir d'un Excel

```
pd.read_excel('foo.xlsx', 'Sheet1',
index_col=None, na_values=['NA'])
```

Ecriture

```
df.to_excel('foo.xlsx', sheet_name='Sheet1')
```

Plan

Conclusions

Bibliothéque Pandas

- Attirer davantage d'utilisateurs de python
- Manipuler facilement les données
- Haute performance lors de l'analyse des données