Langages Dynamiques

Cours 6

kn@lri.fr
http://www.lri.fr/~kn

Plan

Pourquoi ne pas se contenter de `list`?

Les tableaux (type `list`) Python sont des objets très flexibles. Cependant:

- Gros problèmes de performance
- Problème de disposition : pas facile d'extraire une colonne lorsqu'lon fait des tableaux de tableaux
- Peuvent avoir des tailles différentes au sein d'un même tableau de tableaux
- Pas d'opérateurs directement accessibles pour l'analyse de données (moyennes, aggrégats, groupes, ...).

Numpy 

Bibliothèque *optimisée* de tableaux à `n` dimensions

- Tableaux à une dimension (vecteur)
- Tableau à deux dimensions (matrice). Comme un tableau de tableaux, mais force chaque ligne à avoir la même taille
- Tableau à trois dimensions (cube).
- ...

Permet une extraction rapide de sous-parties

Typé! (les colonnes ont un type, il n'est pas possible de mettre une valeur d'un mauvais type dans une case)

Et on ne va pas l'utiliser *directement* !

Pourquoi ne pas se contenter de `numpy`?

- Trop bas niveau
- En général, on n'a pas besoin de tableaux à n dimensions (2 suffisent)
- Pas beaucoup d'opérateurs de données (même si plus que pour les listes)

Numpy est plutôt une bibliothèque à utiliser pour construire sa propre bibliothèque de traîtement de données (statistiques, machine-learning, ...)

pandas

Bibliothèque construite au dessus de `numpy`

- Expose le concept de *DataFrame* (penser table de base de données)
- Import/Export en CSV simplifié (et d'autres formats)
- Intégration fine avec Jupyter et `matplotlib` (conception de graphiques simplifiés)

🧠 Interface ultra compacte, mais assez cryptique

🧠 Bibliothèque très très riche

🧠 On va illustrer les fonctionnalité uniquement sur un (long) exemple

Notre fichier de données

On considère le fichier `produits.csv` :

```
nom;categorie;qtite;prix
Chai;boisson;10;18.00
Chang;boisson;24;19.00
Aniseed Syrup;condiment;12;10.00
Chef Anton's Cajun Seasoning;condiment;1;22.00
Chef Anton's Gumbo Mix;condiment;36;21.35
Grandma's Boysenberry Spread;condiment;12;25.00
Uncle Bob's Organic Dried Pears;fruit_legume;12;30.00
Northwoods Cranberry Sauce;condiment;12;40.00
Mishi Kobe Niku;viande;18;97.00
Ikura;poisson;12;31.00
Queso Cabrales;produit_laitier;1;21.00
Queso Manchego La Pastora;produit_laitier;10;38.00
Konbu;poisson;1;6.00
…
```


Définition du schéma

Avant de charger le fichier, on doit définir un *schéma* : une liste de nom de colonne et de types :

```
schema = {'nom' : 'U', 'categorie' : 'U', 'qtite' : 'i', 'prix' : 'f'}
```

On donne dans un dictionnaire Python, pour chaque colonne du fichier son type

`'U'` : Chaîne de caractères utf-8
`'i'` : Entier
`'f'` : Flottant
`'?'` : Booléen
`'b'` : Octet (byte)
`'m'` : Durée
`'M'` : Date

Chargement du fichier

```
import pandas
produits = pandas.read_csv("produits.csv", delimiter=";", dtype=schema)
```

Produit est un objet de type *DataFrame*.

	nom	categorie	qtite	prix
0	Chai	boisson	10	18.000000
1	Chang	boisson	24	19.000000
2	Aniseed Syrup	condiment	12	10.000000
3	Chef Anton's Cajun Seasoning	condiment	1	22.000000
4	Chef Anton's Gumbo Mix	condiment	36	21.350000
5	Grandma's Boysenberry Spread	condiment	12	25.000000
6	Uncle Bob's Organic Dried Pears	fruit_legume	12	30.000000
7	Northwoods Cranberry Sauce	condiment	12	40.000000

Indexation de *DataFrame*

L'opérateur `[ … ]` de Python est étendu aux *DataFrame*

`produits[n:m]` : Extrait les lignes `n` à `m`. Si `n` est absent, il vaut 0. Si `m` est absent, il vaut la taille de la table

`produits[ 'nom' ]` : Extrait la colonne `nom` comme un vecteur.

`produits[ ['prix', nom', 'qtite'] ]` : Extrait les trois colonnes données dans cet ordre `nom` comme une DataFrame.

`produits.loc[ 10:20, ['prix','nom']]` : Combine les accès par ligne et par colonne

```
produits.loc[1:3,['nom', 'qtite']]
```

	nom	qtite
1	Chang	24
2	Aniseed Syrup	12
3	Chef Anton's Cajun Seasoning	1

Opérations statistiques (exemples)

```
#Calcul de la moyenne d'une colonne
produits['qtite'].mean()

#Calcul du max d'une colonne
produits['prix'].max()

#Calcul de la somme
produits['prix'].sum()
```

Il y a bien d'autres fonctions statistiques ...

Indexation booléene et filtrage

On peut écrire `produits[`*`tab`*`]`, où `tab` est un tableau de booléens qui doit *avoir le même nombre de lignes* que `produits`. Cela renvoie la DataFrame contenant toutes les lignes *i* de `produits` pour lesquelles `tab[`*`i`*`]` vaut `True`.

```
un_sur_trois = [ x % 3 == 0 for x in range(len(produits)) ]
# vaut [True, False, False, True, False, False, True, … ]

print(produit[un_sur_trois])
```

	nom	categorie	qtite	prix
0	Chai	boisson	10	18.000000
3	Chef Anton's Cajun Seasoning	condiment	1	22.000000
6	Uncle Bob's Organic Dried Pears	fruit_legume	12	30.000000
9	Ikura	poisson	12	31.000000
12	Konbu	poisson	1	6.000000
15	Pavlova	dessert	32	17.450001
18	Teatime Chocolate Biscuits	dessert	10	9.200000
21	Gustaf's Knäckebröd	cereale	24	21.000000
24	NuNuCa Nuß-Nougat-Creme	dessert	20	14.000000
27	Rössle Sauerkraut	fruit_legume	25	45.599998
30	Gorgonzola Telino	produit_laitier	12	12.500000
33	Sasquatch Ale	boisson	24	14.000000
36	Gravad lax	poisson	12	26.000000
39	Boston Crab Meat	poisson	24	18.400000
42	Ipoh Coffee	boisson	16	46.000000
45	Spegesild	poisson	4	12.000000
48	Maxilaku	dessert	24	20.000000
51	Filo Mix	cereale	16	7.000000

Opération binaires sur les tableaux

Les opérateurs binaires de Python (+, -, ..., <, ..., >=,...) sont modifiés pour fonctionner sur des tableaux et DataFrame :

```
q10 = produits['qtite'] + 10
# contient le tableau des valeurs de la colonnes quantités chacune
# augmentée de 10

chers = produits[ 'prix' ] >= 100
# contient un tableau de booléens valant True si le prix de la
# ligne est >= 100 et False sinon
```

⚠C'est ***très*** déroutant. Le code ci-dessus se comporte comme :

```
q10 = [ q + 10 for q in produits['qtite'] ]

chers = [ p >= 100 for p in produits[ 'prix' ] ]
```


Utilité ?

Filtrage par condition :

```
produits_chers = produits[ produits['prix'] >= 100 ]
#                           ^tableau de booléen de la bonne taille
```

Mise à jour :

```
produits['qtite'] = produits['qtite'] + 10
#                       ^tableau d'entiers de la bonne taille
```

⚠ Toutes les DataFrame renvoyée jusqu'à présent ne sont que des « vues » sur la DataFrame produits. Les mises à jour, même sur des sous-tables, modifient la DataFrame originale ⇒ faire des copies si besoin.

```
produits2 = produits.copy()
```


Opérations sur les chaînes

L'objet `str` d'une DataFrame contient une version « tableau » des méthodes sur les chaînes de caractères .

```
produits['nom'].str.contains('Choco')
#Renvoie un tableau de booléens valant vrai si la ligne
#correspondante contient 'Choco'
```

La plupart des méthodes de chaînes existent (`.split`, `.startswith`, `.endswith`, ...)

Tri

On peut trier selon une colonne, de manière croissante ou décroissante

```
parqtite = produits.sort_values(by='qtite', acending=False)
#trie par quantité décroissante

parprix = produits.sort_values(by='prix')
#trie par prix croissant (pas obligé de mettre acending=True)


parprixnom = produits.sort_values(by=['prix', 'nom'])
#trie par prix, puis utilise le nom en cas de même prix.
```

Opérations de groupe

Permet de réaliser des *opérations d'aggrégat* (ou de résumé) sur des ensembles de valeurs groupées selon une clé

- Le produit le plus cher par catégorie
- La moyenne des prix par catégorie
- ...

```
moy_par_cat = produits.groupby('categorie', as_index=False)['prix'].mean()
print(moy_par_cat)
```

	categorie	prix
0	boisson	37.979168
1	cereale	20.250000
2	condiment	23.062500
3	dessert	25.160000
4	fruit_legume	32.369999
5	poisson	20.682499
6	produit_laitier	28.730000
7	viande	54.006668

Bibliothèque permettant de tracer des graphiques.

- Plusieurs types de graphiques (courbes, points, nuages de points, barres, camemberts, ...)
- Permet d'afficher dans une fenêtre graphique ou de sauver des fichiers d'images.
- Ne fait pas partie de la bibliothèque standard Python (doit être installé séparément)
- Open-Source
- Constitue le standard pour afficher des graphiques en Python

Le but est juste de faire une introduction ultra-rapide, la documentation est gigantesque, les possibilités d'utilisation très nombreuses.

Plan

Chargement de la bibliothèque

```
from matplotlib import pyplot
```

`pyplot` est lui-même un sous-module de `matplotlib`. On appelera toutes les fonctions en les préfixant par `pyplot`.

Il existe plusieurs façons d'utiliser la bibliothèque. On va voir la plus simple pour notre utilisation

Modèle de dessin

Il y a une notion de « graphique courant » qui est celui sur lequel on est en train de dessiner. Tant qu'on n'efface pas ce graphique, toutes les instructions de dessins vont s'ajouter les unes aux autres.
Forme générale du programme :

```
from matplotlib import pyplot

pyplot.plot(...)              #instructions de dessin
pyplot.grid()
pyplot.xlabel("temps")
pyplot.ylabel("energie")
pyplot.savefig("image1.png")  #sauvegarde dans un premier fichier
pyplot.clf()                  #on efface tout

...                           #autres instructions

pyplot.savefig("image2.png")  #sauvegarde dans un autre fichier
...
```


pyplot.plot

Graphique le plus basique. Trace des points (x,y) et les relie entre eux (ou pas)

```
tx = range(-10, 11)
ty = [ x * x for x in tabx ]
pyplot.plot(tx, ty, label='ma courbe', color='red', ls='--', marker='>')
pyplot.legend()
pyplot.savefig('g1.png')
```


pyplot.plot (2)

```
pyplot.plot(tx, ty, label=lab, color=col, ls=s, marker=m)
```

- `tx` est un tableau de nombres représentant les abscisses
- `ty` est un tableau de nombres représentant les ordonées. Il doit avoir la même taille que `tx`
- `lab` (optionnel) une chaîne de caractère représentant le nom de la coubre pour la légende.
- `col` (optionnel) la couleur de la courbe.
- `ls` (optionnel) le style des traits
- `m` (optionnel) le style des marqueurs

Les marqueurs

Il y a plein de styles de marqueurs. On en donne quelques uns:

symbole	description
'o'	cercle
'D'	diamant
's'	carré
'*'	étoile
'+'	plus
'x'	croix oblique
'<', '>', '^', 'v'	triangles orientés

```
tx = range(10)
pyplot.plot(tx, [x/2 for x in tx], marker='o')
pyplot.plot(tx, [x for x in tx], marker='D')
pyplot.plot(tx, [3*x/2 for x in tx], marker='s')
pyplot.plot(tx, [2*x for x in tx], marker='*')
pyplot.plot(tx, [5*x/2 for x in tx], marker='+')
pyplot.plot(tx, [3*x for x in tx], marker='x')
pyplot.savefig('g2.png')
```

Les couleurs

On peut donner les couleurs :

- par nom symbolique: 'red', 'blue', 'purple', ...
- en hexadécimal: `'#xxyyzz'`, avec `00 ≤ xx,yy,zz ≤ ff`
- en décimal avec transparence: `(x, y, z, a)`, avec `0 ≤ x,y,z,a ≤ 1`

Si on mets plusieurs courbes sur un même dessin sans donner de couleurs, les couleurs sont choisies automatiquement.

Les styles de traits

le paramètre nommé `ls` (*linestyle*) peut prendre plusieurs valeurs :

symbole	description
''	pas de trait
'-'	trait plein
'--'	tirets séparés
':'	pointillés
'-.'	tirets-points

```
tx = range(10)
pyplot.plot(tx, [x/2 for x in tx], ls='', marker='o')
pyplot.plot(tx, [x for x in tx], ls='-', marker='D')
pyplot.plot(tx, [3*x/2 for x in tx], ls='--', marker='s')
pyplot.plot(tx, [2*x for x in tx], ls=':', marker='*')
pyplot.plot(tx, [5*x/2 for x in tx], ls='-.', marker='+')
pyplot.savefig('g3.png')
```


`pyplot.bar` et `pyplot.barh`

```
tx = [1, 2, 3, 4 ]
ty = [10, 20, 2, 50 ]
tl = ['A', 'B', 'C', 'D']
pyplot.bar(tx, ty, tick_label=tl, color='red', label='mon graphe')
pyplot.legend()
pyplot.savefig('g4.png')

pyplot.clf()
pyplot.barh(tx, ty, 0.5, tick_label=tl, color='blue', label='mon graphe')
pyplot.legend()
pyplot.savefig('g5.png')
```


`pyplot.bar` et `pyplot.barh` (2)

```
pyplot.bar(tx, ty, w, tick_label=tl, color=c, label=lab)
```

- `tx` est un tableau de nombres représentant les abscisses
- `ty` est un tableau de nombres représentant les hauteurs des barres. Il doit avoir la même taille que `tx`
- `w` (optionnel) est un flottant indiquant la largeur des barres
- `tl` est un tableau de chaînes représentant les étiquettes des barres. Il doit avoir la même taille que `tx`
- `lab` (optionnel) une chaîne de caractère représentant le nom de la coubre pour la légende.
- `col` (optionnel) la couleur de la courbe.

`plot.barh` trace les barres horizontalement

pyplot.scatter

```
from random import randint
tx = range(100)
ty = [ randint(0, 100) for x in tx ]
ta = [ randint(1, 15)**2 for x in tx ]
pyplot.scatter(tx, ty, s=ta, color=(0.8, 0.8, 0, 0.5))
pyplot.savefig('g6.png')
```


pyplot.scatter (2)

Nuage de points

```
pyplot.scatter(tx, ty, s=ta, color=col, label=lab,)
```

- `tx` est un tableau de nombres représentant les abscisses
- `ty` est un tableau de nombres représentant les ordonnées des points. Il doit avoir la même taille que `tx`
- `s` (optionnel) est un tableau indiquant l'aire de chaque point (par défaut constante). Il doit avoir la même taille que `tx`

Les autrs paramètres sont les même (`color=`, `legend=`)

Autres commandes

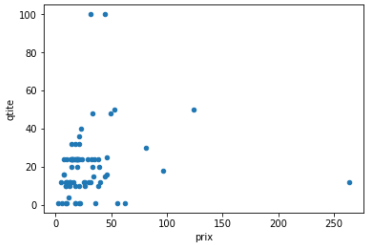
- `pyplot.xlabel(s)` : utilise la chaîne `s` comme étiquette pour l'axe d'es X
- `pyplot.ylabel(s)` : utilise la chaîne `s` comme étiquette pour l'axe d'es Y
- `pyplot.legend()` : ajoute une légende (il faut que les courbes aient été crées avec le paramètre nommé `label=…`)
- `pyplot.grid()` dessine une grille
- `pyplot.clf()` efface le dessin entièrement et supprime tous les graphes
- `pyplot.savefig(f)` sauve le dessin dans un fichier dont le nom est `f`. Le type d'image est déduit de l'extension (`.png`, `.pdf`, `.svg`, `.jpg`, …)

```
from math import sin, cos
tx = [ x * 0.1 for x in range(-30,30) ]
ty1 = [ sin(x) for x in tx ]
ty2 = [ cos(x) for x in tx ]
pyplot.plot(tx, ty1, label='sin')
pyplot.plot(tx, ty2, label='cos')
pyplot.grid()
pyplot.legend()
pyplot.savefig('g7.png')
```


Objet `plot` d'une DataFrame

```
import matplotlib.pyplot

produits.plot.scatter(x='prix', y='qtite')
```




```
moy_par_cat = produits.groupby('categorie', as_index=False)['prix'].mean()
moy_par_cat.plot.bar(x='categorie', y='prix')
```