TP4 : analyse de données avec Pandas

TBD : Manu / Ronan

- lecture / ecriture dans fichier plat et fichier csv
- lecture sequentielle / flux de données
- extraction / mise en forme / tableaux pandas

Installation et importation de modules

Les notebooks Jupyter

Ce travail en autonomie sera réalisé à l'aide de "notebooks" fonctionnant sur l'interpréteur "jupyter". Les notebooks permettent d'écrire et d'exécuter des scripts python à l'aide d'un simple navigateur web. Les résultats d'exécution sont conservés et peuvent être retrouvés d'une session à l'autre.

Ouvrez un terminal dans votre dossier de travail et tapez :

```
jupyter-notebook
```

Ceci ouvre un onglet de l'interpréteur jupyter dans votre navigateur.

- Créez un notebook vierge via le menu new –> `python 3`
- Ou bien cliquez sur le notebook sur lequel vous souhaitez travailler.

Introduction aux notebooks et à Python (en anglais)

Pour utiliser un notebook, voir :

- 1. What is the Jupyter notebook?
- 2. Notebook basics
- 3. Running code
- 4. Working with Markdown cells

- Une vidéo en anglais

Numpy : vecteurs, matrices et algèbre linéaire en Python¶

Le module 'numpy' permet des calculs numériques rapides en Python. Il fournit des vecteurs, matrices et les opérateurs matriciels sont rapides. Importez le module numpy.

```
import numpy as np
```

- Numpy User Guide: index.html

Conseils importants :

- N'utilisez pas la classe `matrix` de numpy. Utilisez à la place des vecteurs à dimensions multiples. Cela fonctionne mieux.
- Si vous travaillez avec des vecteurs, utilisez `y=numpy.zeros(5)` et pas `y=numpy.zeros( (5, 1) )`. Avec la première manière vous pouvez écrire `y[5]` et pas avec la seconde.
- L'indexation des vecteurs commence à 0.

Créer des ''arrays'' numpy¶

Les vecteurs et les matrices peuvent être créés à partir de :

- listes ou tuples
- à l'aide de functions
- en lisant des fichiers de données

Copie et "copie profonde"

Pour de bonnes performances, les affectations en Python ne copient pas les objets mais uniquement des références d'objets. Ainsi, quand des objets sont passés entre les fonctions, cela évite de surcharger la mémoire.

Testez le code suivant:

- une matrice 3x2 :

```
A = np.array([[1, 2, 3], [4, 5, 6]])
print(A)
```

- Affecter A à B (Ceci ne crée pas une copie, mais une référence)

```
B = A
print(B)
```

- Si on mofifie B, qu'advient-il de A?

```
B[1,1]=-5
print(A)
print(B)
```

Copier un array

Pour éviter ce comportement, afin d'obtenir un objet B indépendant de A, il est nécessaire d'utiliser la fonction `copy`:

```
C = np.copy(A)
print(A)
print(C)
```

```
C[-1,-1] = -10
print(A)
print(C)
```

Attributs des arrays

Les vecteurs et les matrices de numpy sont des objets disposant d'un certain nombre d'attributs et de méthodes attachés:

- Les vecteurs et matrices ont une taille. Ceci est l'attribut `shape`:

```
A.shape
```

- Le nombre d'éléments est disponible à l'aide de l'attribut `size`:

```
A.size
```

- Le type de l'array peut être défini explicitement à l'aide de l'argument `dtype`:

```
A.dtype
```

Le type peut être défini lors de l'initialisation:

```
Mc = np.array([[1, 2], [3, 4]], dtype=complex)
print(Mc)
```

Fonctions permettant de générer des vecteurs et des matrices

Pour éviter de définir manuellement des matrices de grande taille, de nombreuses fonctions numpy permettent de générer des arrays de différentes formes. Parmi les plus courantes:

- `arange` (la valeur finale n'est pas contenue dans le vecteur):

```
# create a range
x = np.arange(0, 10, 2) # arguments: start, stop, step
print(x)
```

```
x = np.arange(-1, 1.1, 0.1)
print(x)
```

- `linspace`:

```
# avec linspace, le début ET la fin sont inclus
print(np.linspace(0, 10, 25))
```

- `logspace`:

```
print(np.logspace(0, 10, 10, base=np.e))
```

- Données aléatoires:

```
from numpy import random
# uniform random numbers in [0,1]
print(random.rand(5,5))
# standard normal distributed random numbers
print(random.randn(5,5))
```

- zeros et uns:

```
print(np.zeros((3,3)))
print(np.ones((3,3,3),dtype=int)*5)
```

Manipuler les arrays

Indexation

en Python, l'indexation commence à 0.

```
# M is a matrix, or a 2 dimensional array, taking two indices
print(A)
print(A[1,1])
```

- Si on omet un indice d'un array multidimensionnel, la ligne complète est retournée (ou, plus généralement, un array de dimension N-1):

```
print(A[1])
```

- La même chose est obtenue en utilisant `:` à la place d'un indice:

```
print(A[1,:]) # row 1
print(A[:,0]) # column 0
```

- Grâce à l'indexation, de nouvelles valeurs peuvent être affectées aux éléments:

```
A[0,0] = -1
print(A)
```

- Marche aussi pour les lignes et les colonnes:

```
A[1,:] = 0
A[:,2] = -1
print(A)
```

Algèbre linéaire

Les opérations doivent doivent prendre en argument des vecteurs ou des matrices pour bénéficier d'une vitesse optimale (fonctions compilées).

- Opérations avec des scalaires:

```
v1 = np.arange(0, 5)
print(v1)
print(v1 * 2)
print(v1 + 2)
```

- Le comportement par défaut correspond à des opérations terme à terme:

```
print(v1 * v1)
```

- Produit scalaire:

```
print np.dot(v1, v1)
```

Pour aller plus loin :

- Tutoriel inspiré de : 03b_numpy_tutorial.html
- Tentative_NumPy_Tutorial
- scientific-python-lectures

```
git clone https://github.com/jrjohansson/scientific-python-lectures/
```

Matplotlib

Matplotlib est une librairie de visualisation des données extrêmement flexible et paramétrable, proposant un grand nombre de modes de visualisation des données (trait, barres, barres d'erreur, histogrammes, images, etc).

Pour importer cette librairie :

```
import matplotlib.pyplot as plt
```

dans un notebook, on ajoutera la commande "magique":

```
%matplotlib inline
```

Un affichage simple

```
import numpy as np
from numpy import random

data = random.rand(20)
plt.plot(np.arange(20), data, 'bo-')
plt.show()
```

Autres exemples

- Fermer toutes les figures:

```
plt.close('all') # ferme toutes les figures
```

- Affichage simple:

```
# Common plot
x=np.linspace(0,2*np.pi,100)
y=x
z=np.sin(x)

plt.figure(1) # new figure
plt.plot(x,y,'g') # set x=y to the buffer
plt.plot(x,z,'rx-') # add z=sin(x) to the buffer
plt.xlabel('Axe des X')
plt.ylabel('Axe des Y')
plt.title('Figure de test')
plt.legend(('y=x','y=sin(x)'),'best')
plt.show() # we print everything!
```

- Multiplot:

```
plt.figure(2)
sub1=plt.subplot(121)
plt.plot(x,y,'r')
sub1.set_title('Graphe1')
sub2=plt.subplot(122)
plt.plot(x,y*y,'g')
sub2.set_title('Graphe2')

plt.show()
```

- Affichage logarithmique :

```
plt.figure(3)
plt.semilogx(x,y) # X log scale
plt.figure(4)
plt.semilogy(x,y) # Y log scale
plt.figure(5)
plt.loglog(x,y) # X and Y log scale

plt.show()
```

Pour aller plus loin

De nombreux exemples de scripts d'affichages sont disponibles |ici. Testez-en quelques-uns.

Pandas

L'utilisation de données structurées dans un programme Python nécessite de faire appel à des librairies spécialisées. Nous utiliserons ici la librairie `pandas` qui sert à la mise en forme et à l'analyse

des données.

```
import pandas
```

On considère une série d'enregistrements concernant des ventes réalisées par un exportateur de véhicules miniatures. Pour chaque vente, il entre dans son registre de nombreuses informations :

- nom de la société cliente
- nom et prénom du contact, adresse, téléphone
- nombre d'unités vendues
- prix de vente
- etc…

Ces informations sont stockées dans un fichier au format 'csv' (comma separated values) : `ventes.csv`. Téléchargez ce fichier.

Dans un premier temps, regardez son contenu avec un editeur de texte (**geany**, **gedit** ou autre…). La première ligne contient les noms des attributs (`NUM_COMMANDE`, `QUANTITE`,…). Les ligne suivantes contiennent les valeurs d'attributs correspondant à une vente donnée. en tout plus de 2000 ventes sont répertoriées dans ce fichier.

Ouvrez-le maintenant à l'aide d'un tableur (par exemple **`localc`**). Les données sont maintenant "rangées" en lignes et colonnes pour faciliter la lecture.

Créez un dossier `data` dans votre répertoire de travail (celui qui contient les notebooks) et déplacez le fichier `ventes.csv` dans le dossier `data`.

Lecture des données

Les données sontau format `csv`, on utilise:

- `pandas.read_csv`. Voir dataframes pandas. Pandas permet également de lire les données au format `xls` et `xlsx` (Excel).
- Pandas permet de manipuler des données de type `DataFrame`. Si `d` est votre `Dataframe` et que vous avez besoin de passer au format `array` de `numpy` : `m = d.as_matrix()`

```
with open('ventes.csv') as f:
    data = pandas.read_csv(f)
print(data)
```

avec `data` une structure de données de type `DataFrame`

Testez les commandes suivantes :

```
print(len(data))
```

```
print(data.columns)
```

Syntaxe de type dictionnaire :

```
print(data["VILLE"])
```

```
print(data[["VILLE", "PAYS"]])
```

Autre syntaxe :

```
print(data.VILLE)
```

```
print(data.VILLE.head(10))
```

Çà marche aussi avec la syntaxe "dictionnaire":

```
print(data["VILLE"].head(10))
```

Modifier les données

La colonne montant contient des données aberrantes. Modifions les :

```
data.MONTANT = data.PRIX_UNITAIRE
data.MONTANT *= data.QUANTITE
print(data.MONTANT)
```

Sélectionner les données

```
selection = data[data.MONTANT > 6000]
print(selection[["MONTANT","DATE_COMMANDE","VILLE","PAYS","NOM_CONTACT","PRE
NOM_CONTACT"]])
```

Opérateurs d'agrégation

- usage : statistique sur les données
- principe :
    - opérateur d’aggrégation :
        - tout type de données : comptage (attention aux doublons)

```
print(data["VILLE"].count())
print(data["VILLE"].drop_duplicates().count())
```

- données quantitatives (et non qualitatives) : somme, moyenne, ecart-type (count, sum, mean, std, min, max, ...)

```
print(data["MONTANT"].mean())
print(data["MONTANT"].std())
```

Affichage

```
data["MONTANT"].hist()
plt.show()
```

Calcul par classes

ici la date:

```
grouped = data.groupby('DATE_COMMANDE')
print(grouped["NUM_COMMANDE"].count())
```

- groupage multiples :

```
grouped_multiple = data.groupby(['PAYS', 'VILLE'])
print(grouped["NUM_COMMANDE"].count())
```

Pour aller plus loin :

- Une introduction très détaillée aux DataFrames (en Français)
- Introduction to Pandas (en anglais)

A faire

- Trouvez le nombre de ventes, le nombre de clients référencés (sans doublons), et le nombre de références produits (sans doublons).
- Afficher le nombre de client et le chiffre d'affaires
  - par pays
  - par pays puis par état
  - par pays puis par état puis par ville
- Donnez le nombre de ventes en fonction du mois pour l'année 2004
- Donnez le chiffre d’affaires par année et trimestre pour les ventes réalisées aux états unis
- Quelle est la catégorie de véhicules la plus vendue?

Tables Pivot

Agrégation des données selon différents attributs/dimensions

exemple : on représente les ventes selon (1) la dimension géographique (pays, état) et (2) la dimension temporelle (les trimestres et mois de l'année)

```
T = pandas.pivot_table(data, values = 'MONTANT', index = ['PAYS','VILLE'],
columns = ['TRIMESTRE','MOIS'], aggfunc=np.sum)
print(T)
```

```
T.plot(kind='bar', subplots = 'True')
plt.show()
```

Evolution des ventes au cours de l'année pour la France seulement:

```
selection = data[data.PAYS == "France"]
T2 = pandas.pivot_table(selection, values = 'MONTANT', index =
['TRIMESTRE','MOIS'], columns = ['PAYS','VILLE'],  aggfunc=np.sum)
print(T2)

T2.plot(kind='bar', subplots = 'True')
plt.show()
```

A faire

- Donnez le nombre de ventes par catégorie pour chaque trimestre et mois de l'année. Choisissez le graphique le plus adapté pour représenter les données.
- Donnez le montant moyen des ventes par pays et par ville, pour chaque catégorie de produits. Choisissez le graphique le plus adapté pour représenter les données.

Pour aller plus loin:

- Cartographier les données
- Géopandas

Exercice complet

Télécharger un jeux de données Télécharger l'archive GES12_Flatfile.zip Déplacer et extraire le dossier. Nous nous intéresserons au fichier PERSONT.TXT

Dans un terminal faire:

ls

head 'PATH_TO/GES12_Flatfile/PERSON.TXT'

```
input_file_path = os.path.join("PATH_TO", '/GES12_Flatfile/PERSON.TXT')

df = pd.read_csv(input_file_path, delimiter='\t')
print (df)
```

df est une structure panda

```
# Les types des colonnes d'un data frame sont inférés à partir des valeurs
print(df.dtypes)
```

```
# Vous pouvez connaitre les colonnes et les attributs d'un data frame avec
```

```
TAB
# en tapant df.<TAB>
print(df.AGE)
```

Visualiser les données

```
#Premiers et derniers éléments
print(df.head())

print(df.tail(2))
```

```
print(df[:2])
print(df.index)
print(df.describe())
```

```
#transposition
print(df.T)
```

```
#Tri suivant un axe
print(df.sort_index(axis=1, ascending=False))
```

```
# Le tableau numpy sous jacent au data frame
print(df.values)
print(df.values.shape)
```

Selection Accès aux lignes et colonnes par loc et iloc

```
print (df['AGE'])

print (df.AGE)
```

```
print(df.loc[:,['VEH_NO','PER_NO', 'PER_TYP']])
print(df.loc[3:5,['VEH_NO','PER_NO', 'PER_TYP']])
```

```
# Accès aux cases par l'index entier
print(df.iloc[3])
print(df.iloc[3:5,0:2])
```

```
print(df[df.AGE > 80])
```

Entrer des valeurs

```
df2 = df.copy()
df2['PER_NO'] =1
print(df2)
```

```
print (df2.shape)

# Eliminer les lignes avec un champs quelconque indéfini
df2 = df2.dropna(how='any')
```

```
print (df2.shape)

# Remplacer les valeurs manquantes par 5
print (df2.fillna(value=5))
```

```
print (pd.isnull(df2))
```

Calculs simples

```
print (df.mean())
print( df.mean(1))
```

Histograms

```
# Histogramme des types de dommages
s = df.INJ_SEV.value_counts()
print(s)
```

```
s.hist()

print (s.index, s.values)
p = pd.Series(s.index, index=s.values)
print (p)
p.hist()

plt.hist( s.values)
plt.show()
```

Merge jointure

```
# Créer un fichier ou l'on dispose d'une ligne par passager, avec ses infos,
et avec les infos associées à son conducteur
# Par jointure sur nO vehicule et casenum des drivers = infos sur les
conducteurs
drivers = df[df.PER_TYP==1]
# et des passagers
passengers =df[df['PER_TYP']==2]
passengers = passengers[['CASENUM' ,'VEH_NO', 'PER_NO','INJ_SEV']]
new_data = pd.merge (drivers, passengers, on=['CASENUM','VEH_NO' ],
how='inner',left_index=True,copy=False)
print(new_data.head())
```

```
# L'opérateur sum ne s'applique qu'aux attributes de type nombre.
# on peut grouper sur plusieurs attributs avec ['Atribut A', 'Attribut B']
print(df.groupby('AGE').sum())
```

Input/output

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

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

Pour aller plus loin

Tutoriel :

- 03_overview.html

Données :

- hamlet.txt

Données supplémentaires

- users.dat
- movies.dat
- ratings.dat
- cars.csv

From:
https://wiki.centrale-med.fr/informatique/ - **WiKi informatique**

Permanent link:
https://wiki.centrale-med.fr/informatique/tc_info:tpa3

Last update: **2019/11/29 14:32**