W6-S6-M4-Mini-projet-meteo-data

December 14, 2014

1 Visualisation de données météo

1.1 Mini-Projet

1.1.1 Introduction

Ce mini-projet vise à vous donner accès à des données assez riches pour vous permettre d'expérimenter avec `matplotlib`, et mettre en oeuvre une visualisation de données en 2D et en 3D.

Nous allons pour cela travailler avec des données obtenues auprès de OpenWeatherMap.

Spécifiquement, ce site publie un échantillon statique de données qui est disponible à cette URL

`http://78.46.48.103/sample/daily_14.json.gz`

Comme nous le verrons plus bas, il n'est pas nécessaire pour vous d'aller chercher ces données chez OpenWeatherMap, nous les republions à plusieurs échelles. L'échantillon complet couvre le monde entier et expose des données météo sur une période d'environ deux semaines en Mars 2014.

1.1.2 `matplotlib`

Par manque de temps nous n'avons pas pu introduire `matplotlib` dans une vidéo ou un complément. Nous allons en dire quelques mots (très rapides), si vous n'avez aucune idée de comment l'utiliser.

Que vous choisissiez ce sujet ou non, le point important ici est qu'une fois que vous avez obtenu des données avec votre programme python - et pourvu que vous ayez installé `matplotlib` qui ne fait pas partie de la librairie standard -, vous avez à votre disposition un outil très puissant et vraiment très simple pour visualiser très rapidement ces données sous des formes diverses.

Pour toute la suite je vous recommande d'avoir sous la main * le tutoriel matplotlib * la documentation `matplotlib` * le tutorial pour les visualisation 3D

Toutes les fonctions de visualisation attendent en argument des listes de valeurs pour X, pour Y, et le cas échéant pour Z. La plupart du temps vous pouvez obtenir une visualisation simplement en choisissant la fonction (par exemple `plot` affiche les données avec une courbe standard, `bar` affiche les données en barres ..) et en passant X et Y; voyons cela :

```
In []: import matplotlib.pyplot as plot

       import math

In []: # on découpe à la main l'intervalle [0, 6.3] en pas de 0.1
       X = [ 0.1 * x for x in range (64)]
       # les valeurs de sin() correspondantes
       Y = [ math.sin(x) for x in X]

       # une courbe qui représente sin(x) entre 0 et 2.pi
       plot.plot(X, Y)
```

Vous voyez que `matplotlib` s'occupe de montrer les échelles, etc... Vous avez la possiblité de modifier ce rendu par défaut mais dans la plupart des cas, vous obtenez de cette façon quelque chose de raisonnablement bien présenté.

`show()` Dans le contexte des notebooks cela n'est pas nécessaire mais dans votre application n'oubliez pas d'invoquer la méthode `matplotlib.pyplot.show()` pour que votre illustration s'affiche.

1.1.3 numpy

Comme vous le voyez il n'est pas nécessaire d'utiliser `numpy` pour visualiser des données.

Nous n'utilisons pas du tout `numpy` dans cet exercice, mais sachez tout de même, si par ailleurs vous utilisez déjà `numpy`, que le mécanisme de passage de données est très commode avec `numpy`, qui manipule déjà nativement les données sous cette forme.

```
In []: import numpy
```

```
In []: # avec la méthode 'numpy.linspace'
       # on peut facilement découper
       # un intervalle en morceaux
       numpy.linspace(0, 10, 21)
```

```
In []: # aussi pour afficher une fonction
       # sur un intervalle c'est extrêmement simple

       X = numpy.linspace (0, 2*numpy.pi, 101 )

       # d'autant que les fonctions numpy
       # font implicitement l'équivalent de 'map'
       # sur la liste d'entrée

       plot.plot (X, numpy.sin(X))
```

1.1.4 En 3D

On peut tout aussi simplement visualiser des données en 3D, voici un exemple de visualisation en 3D d'une gaussienne, que nous montrons ici à un échelle 6 pour simplifier le code

$ z = e^{-(x^2+y^2)/36}$

```
In []: import math
       from itertools import product

       # la grille de départ en X et en Y
       xscale = range ( -10, 11 )
       yscale = range ( -10, 11 )

       # tous les points X,Y (soit 21 x 21 = 441 au total )
       # il faut recréer un itérateur pour chaque boucle
       def XY():
           return product(xscale, yscale)

       # on calcule les tableaux d'entrée pour matplotlib
       X = [ x for x, y in XY() ]
       Y = [ y for x, y in XY() ]
       Z = [ math.exp(-(x**2+y**2)/36.) for x, y in XY() ]
```

Ici encore, il y a une différence de comportement entre le rendu dans un notebook qui est statique, et ce que vous obtenez sur votre ordinateur; dans ce dernier cas vous pouvez "faire tourner" la figure en 3D pour la voir sous l'angle que vous voulez.

```
In []: ### la visu
       from mpl_toolkits.mplot3d import Axes3D
       from matplotlib import cm
       import matplotlib.pyplot as plot

       figure = plot.figure()
       axes = figure.gca(projection='3d')

       # on fixe cmap pour des couleurs plus gaies
       axes.plot_trisurf(X, Y, Z, cmap=cm.jet, linewidth=0.2)

       plot.show()
```

1.1.5 `matplotlib` *vs* `gnuplot`

En guise de digression, signalons que **`gnuplot`** est un outil de visualisation assez ancien et qui a fait ses preuves. Si vous connaissez et maîtrisez bien cet outil, vous pouvez parfaitement faire le miniprojet en utilisant **`gnuplot`** plutôt que **`matplotlib`**. je vous invite à regarder `Gnuplot.py`, vous éviterez ainsi l'erreur qui consiste à générer directement des fichiers pour **`gnuplot`**. Par ailleurs **`gnuplot`** est un programme indépendant de python, qu'il faut naturellement donc avoir installé séparément.

1.1.6 Les données

Pour revenir à l'échantillon de OpenWeatherMap, une fois décompressé et décodé, il contient, pour un grand nombre de villes (22631 exactement), des données de type : * champ **`city`** : position géographique, nom, etc.. * champ **`time`** : date (vous pouvez ignorer ce champ pour l'exercice) * champ **`data`** : une liste de mesures disponibles concernant ce point, sous la forme d'une liste de mesures; l'échantillon contient en moyenne 16 mesures par point; * élément de **`data`** : une mesure correspond à un jour donné, et vient comme un dictionnaire contenant typiquement une valeur pour * l'heure des mesures (champ **`dt`**, pour data time) - voir aussi plus bas, * la vitesse et la direction du vent (**`speed`** et **`deg`**) * l'humidité et la pression * et s'agissant de la température, à nouveau un dictionnaire pour décrire les températures à divers moments de la journée

Je vous laisse deviner les unités utilisées - je rappelle juste que

$ 0°C = 273.15°K $

Pour être plus explicite encore, voici un pretty-print d'un objet correspondant à une ville (Cayenne en Guyanne), en ne montrant que la première mesure :

```
{u'city': {u'coord': {u'lat': 49.558578, u'lon': 1.62803},
           u'country': u'FR',
           u'id': 3028097,
           u'name': u'Cayenne'},
 u'data': [{u'clouds': 80,
            u'deg': 330,
            u'dt': 1394884800,
            u'humidity': 85,
            u'pressure': 1028.47,
            u'speed': 5.41,
            u'temp': {u'day': 282.3,
                      u'eve': 282.86,
```

```
                    u'max': 283.22,
                    u'min': 279.7,
                    u'morn': 279.7,
                    u'night': 281.96},
             u'weather': [{u'description': u'broken clouds',
                           u'icon': u'04d',
                           u'id': 803,
                           u'main': u'Clouds'}]},
           '... other similar dicts ...'],
 u'time': 1394865585}
```

Format concret L'échantillon complet décompressé contient autant de lignes que de villes (22631 lignes donc), et chaque ligne est un encodage JSON de la structure que nous venons de voir. Vous pouvez donc charger un fichier en quelques lignes de code qui combinent * un fichier d'entrée considéré comme un itérable, * et les fonctions du module `json`; vous remarquerez à cet égard la différence entre `load` et `loads`

Rappel sur les dates S'agissant des dates, on retrouve ici notre nombre de secondes depuis le 1^{er} Janvier 1970, et voici comment vous pouvez afficher ce genre de valeurs.

```
In []: import time
       # *Y*ear *m*onth *d*ay *H*our *M*inute
       date_format="%Y-%m-%d:%H-%M UTC"

       # city['city']['data'][0]['dt']
       dt = 1394884800

       # gmtime pour afficher en heure UTC (formerly GMT)
       print 'champ dt', time.strftime(date_format, time.gmtime(dt))
```

1.1.7 Mise en place

Comme l'échantillon est très gros, je vous ai préparé des versions de plus en plus petites :

- Monde entier (échantillon complet)
- cities_world.json.gz (11 Mb)
- cities_world.json (98 Mb)
- Europe (62, 33, 34, -11) (limites nord, est, sud, et ouest)
- cities_europe.json.gz (3 Mb)
- cities_europe.json (31 Mb)
- France (51.2, 8.3, 42.3, -5.3)
- cities_france.json.gz (480 kb)
- cities_france.json (6 Mb)
- Ile-de-France (49, 2.75, 48.5, 2)
- cities_idf.json.gz (17 kb)
- cities_idf.json (900 kb)

Sachant que pour mette au point il est très conseillé de commencer avec un petit fichier car le chargement du fichier complet peut prendre dans les 10 secondes.

1.1.8 Le sujet

Le programme que nous avons écrit s'utilise de la manière suivante :

$ ./meteodata.py –help usage: meteodata.py [-h] [-c CROP] [-n SELECT_NAMES] [-a] [-l] [-1] [-2] [-3] filename

positional arguments: filename input JSON file - might be gzipped

optional arguments: -h, –help show this help message and exit -c CROP, –crop CROP specify a region for cropping regions are rectangular areas and can be specified as a comma-separated list of 4 numbers for north, east, south, west -n SELECT_NAMES, –name SELECT_NAMES cumulative - select cities by this name(s) -a, –all select all cities -l, –list If set, selected city names get listed

-1, –1d Display a bar chart for the pressure in the first selected city -2, –2d Display a 2D diagram of the positions of selected cities -3, –3d Display a 3D diagram of the pressure in all cities

1.1.9 Entrées

- il faut donner au programme un des fichiers d'entrée (le paramètre obligatoire `filename`)
- on peut aussi avec l'option `-c` restreindre à un rectangle, ici par exemple les USA (on y reviendra); lorsqu'on utilise cette option les villes qui ne sont pas dans la zone sont **complètement ignorées** :

  $ ./meteodata.py -c 50,-70,25,-125 data/cities_world.json ———- From data/cities_world.json dealing with 22631 cities ———- After cropping with Top 50.0 Right -70.0 Bottom 25.0 Left -125.0 dealing with 3146 cities ———- No city selected

- avec l'option -n (que vous pouvez répéter) vous pouvez **sélectionner** des villes par leur nom; vous pouvez répéter l'option pour en sélectionner plusieurs;

Les deux mécanismes sont indépendants; on considère toutes les villes qui sont dans la région, et parmi elles celles qui sont sélectionnées seront mises en évidence :

```
$ ./meteodata.py -c 50,-70,25,-125 -n 'new york' -n dallas data/cities_world.json
---------- From data/cities_world.json
    dealing with 22631 cities
---------- After cropping with Top 50.0 Right -70.0 Bottom 25.0 Left -125.0
    dealing with 3146 cities
---------- Selected 3 cities
    [u'New York', u'New York', u'Dallas']
```

Dans cet exemple on va travailler sur un total de 3146 villes, dont trois sont sélectionnées (deux s'appellent 'New York' et une s'appelle 'Dallas')

Les options `-a` et `-l` sont très accessoires vous n'êtes pas obligés de vous en occuper.

1.1.10 Visualisations

Les 3 autres options (-1, -2 et -3) correspondent à trois modes de visualisation. Comme vous allez le voir ce sont des choix très arbitraires, n'hésitez pas à broder et à changer les spécifications.

Visualiser les températeures dans une ville donnée (option -1) Avec l'option -1, le programme affiche sous forme de diagramme à barres, les températures de la première ville sélectionnée, et ce sur l'ensemble de la période.

Pour chaque date on affiche, dans l'ordre, les champs suivants de `temp` dans `data`

```
'morn', 'day', 'eve', 'night'
```

Ainsi par exemple

```
./meteodata.py -1 -n brest data/cities_france.json
```

affiche le diagramme
Pour l'implémentation de l'option -1 nous avons utilisé matplotlib.pyplot.bar

---

Visualiser les villes par leur position en 2 dimensions Avec l'option -2, le programme affiche l'ensemble des villes par leur position, en mettant en évidence les villes sélectionnées avec une taille plus importante et une couleur différente.

Ainsi par exemple

```
./meteodata.py -2 -n nice -n toulouse -n bordeaux -n brest -n 'le havre'  -n strasbourg data/cities_fra
```

montre ceci
alors que

```
./meteodata.py -2 -n paris -n berlin -n roma  data/cities_europe.json
```

affiche ceci
où vous pouvez voir que la densité de couverture n'est pas uniforme dans tous les pays européens - ni sur tout le territoire français d'ailleurs.

Pour l'implémentation de l'option -2 nous avons utilisé matplotlib.pyplot.scatter

---

Visualiser la pression en 3D Enfin avec l'option -3 notre programme affiche la pression mesurée sur la zone en 3D. Il n'y a dans ces données aucune garantie que toutes les villes ont des mesures pour exactement les mêmes dates. Cependant pour faire simple et comme notre but ici est uniquement d'expérimenter avec la visualisaion : * on a pris pour chaque ville la pression dans la première mesure listée, * et on a calculé la date sur la base de la première mesure de la première ville listée.

Par exemple si on lance

```
./meteodata.py -3 data/cities_europe.json
```

on obtient une visu en 3D qui selon le point de vue ressemble à ceci
ceci
où vous devinez les contours de l'Europe avec, à nouveau, un peu d'imagination et de bonne volonté, sachant que la zone en bleu correspond aux Alpes, où la pression est plus faible très vraisemblablement à cause de l'altitude.

Pour l'implémentation de l'option -3 nous avons utilisé plot_trisurf

---

1.1.11 Épilogue : le visuel du MOOC

Le visuel du MOOC a été réalisé entièrement en `matplotlib`. Si cela vous intéresse vous pouvez voir le code ci-dessous.

Remarquez notamment le style `xkcd` qui donne le petit coté amusant. Pour que cela rende correctement toutefois, il faut avoir installé les bonnes fontes sur son ordinateur - ce qui n'est pas le cas de la plateforme qui héberge les notebooks malheureusement :)

```
In []: import xkcd
```

```
In []: from inspect import getsource
```

```
       for line in getsource(xkcd).split("\n"):
           print line
```
