1 Correction du TD 7

Première demi-heure : Récupérer les données .

Lire les données :

```
import pandas
df = pandas.read_table(file, header = False, sep = "\t", decimal = ",", parse_dates = ["last_update"],
                       date_parser = lambda s : datetime.datetime.strptime (s, "%d/%m/%Y %H:%M"))
```

La première et la dernière dates :

```
print ("min_date", df["last_update"].min())
print ("max_date", df["last_update"].max())
```

Le nombre maximum de vélos et de places vides :

```
print ("max velo", df["available_bikes"].max())
print ("max_place", df["available_bike_stands"].max())
```

La proportion du temps durant lequel la station n'a pas de vélos disponibles :

```
print ("pas de vélo", len(df [ df["available_bikes"]==0 ]) / len(df) )
```

La proportion du temps durant lequel la station n'a pas de places disponibles :

```
print ("pas de place", len(df [ df["available_bike_stands"]==0 ]) / len(df) )
```

Le nombre d'emplacements de vélos à cette station :

```
add = df["available_bikes"] + df["available_bike_stands"]
print (add.max())
```

Première demi-heure : Visualisation des données .

```
import matplotlib.pyplot as plt
df = df.set_index("last_update")
ts = df.ix[:,"available_bikes"]
ts.plot()
plt.show()
```

Le dessin montre plusieurs jours contigüs pour lesquels la courbe est plate : il s'agit de données manquantes. Pour une raison quelconque, il est possible de savoir ce qu'il s'est durant ces périodes. Cela ne remet pas en cause les résultat précédents car il n'y a pas de données pour ces dates-là. Le nombre d'emplacements n'est pas altéré par la présence de valeur par défaut ou aberrantes.

```
import numpy
from matplotlib.mlab import csv2rec
import matplotlib.pyplot as plt
import matplotlib.cbook as cbook
from matplotlib.ticker import Formatter

class MyFormatter(Formatter):
    def __init__(self, dates, fmt='%Y-%m-%d'):
        self.dates = dates
        self.fmt = fmt

    def __call__(self, x, pos=0):
        'Return the label for time x at position pos'
        ind = int(round(x))
        if ind>=len(self.dates) or ind<0: return ''

        return self.dates[ind].strftime(self.fmt)

formatter = MyFormatter(df["last_update"])

fig, ax = plt.subplots()
ax.xaxis.set_major_formatter(formatter)
ax.plot(numpy.arange(len(ts)), ts, '-')
fig.autofmt_xdate()
plt.show()
```

Troisième demi-heure : Visualisation des semaines superposées .

```
df2["weekday"] = df2['last_update'].map(lambda d: str(d.weekday()) + "-" + d.strftime("%H:%M"))
```

Pour aligner le nobre de vélos disponibles par heure et jour de la semaine, on ajoute d'abord une colonne contenant le numéro de la semaine. On se débarrasse de la colonne `last_update`

```
df2["weekno"] = df2['last_update'].map(lambda d: d.isocalendar()[1])
sp = "available_bikes weekday weekno".split()
sem = df2.ix[:,sp]
piv = sem.pivot ('weekday', "weekno", "available_bikes")

piv.plot()    # pour visualiser les données
plt.show()
```

Quatrième demi-heure : Données manquantes et lissage .

Exercice 1

```
# on somme la série complète 12 fois mais décalé à chaque fois
nbperiod = 12
mat = numpy.matrix ( [0.0] * (len( df2 ) - nbperiod+1) ).transpose()
for i in range(0,nbperiod) :
    r = df2.ix[i:len(df2)-nbperiod+i,'available_bikes']
    m = numpy.matrix(list(r)).transpose()
    mat += m
```

```
mat /= 1.0 * nbperiod

# on élimine les premières et dernières valeurs
df2 = df2[nbperiod/2:-nbperiod/2+1]
df2["lisse"] = mat

# on répète la même manipulation pour superposer les semaines
sp = "lisse weekday weekno".split()
sem = df2.ix[:,sp]
piv = sem.pivot ('weekday', "weekno", "lisse")
piv.plot()
plt.show()
```

Une seconde version utilise la fonction de `rolling_mean` de `pandas`.

```
df2 = df.ix[:, ["last_update","available_bikes"]]

# on ajoute une colonne lissage qui contient la moyenne mobile sur 12 périodes via la fonction rolling_mean
df2["lissage"] = pandas.stats.moments.rolling_mean(df2["available_bikes"],12)

# on fait la meme transformation que dans l'exercice précédent pour afficher la valeur lissée par semaine, par timebox
df2["weekday"] = df2['last_update'].map(lambda d: str(d.weekday()) + "-" + d.strftime("%H:%M"))
df2["weekno"] = df2['last_update'].map(lambda d: d.isocalendar()[1])
serietemp = df2.ix[:,["lissage","weekday","weekno"]]
piv = serietemp.pivot ("weekday", "weekno", "lissage")
piv.plot() # pour visualiser les données
plt.show()
```

Exercice 2

```
beg = "4-16:00"
piv = thispiv
piv.ix[beg:,31] = numpy.NaN
mx = numpy.array(piv.as_matrix())

for i,k in enumerate(piv.index) :
    if k >= beg and numpy.isnan(piv.ix[k,31]):
        ave_week = numpy.average( mx[~numpy.isnan(mx[:,2]),2])
        ave_hour = numpy.average( mx[i, ~numpy.isnan(mx[i,:])] )
        piv.ix[k,31] = (ave_week*ave_hour)**0.5

piv.plot()
plt.show()
```

Pour aller plus loin ou pour ceux qui ont fini plus tôt .

Les jours de semaines et les week-ends montrent deux comportements différents, en amplitude tout du moins. Il vaudrait mieux ne pas moyenner sur la semaine complète mais sur les jours de la semaine si la valeur manquante à compléter est un jour en semaine.

On peut supposer que des vélos seront déposés ou rétirés en grand nombre sur une période de cinq minutes. Cela permettrait de détecter les ajustements fait par la société qui gère les vélib et de savoir dans quel sens ils sont fait pour telle ou telle station. On peut aussi regarder si les stations sont du type : les vélos se vident le matin et se remplissent le soir ou l'inverse. Cela éviterait de dépendre d'un processus d'ajustement.

fin correction TD ?? □