td2a_visualisation

January 21, 2022

1 2A.data - Matplotlib

Tutoriel sur matplotlib.

[1]:
```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```

[1]:
```
<IPython.core.display.HTML object>
```

Aparté
Les librairies de visualisation en python se sont beaucoup développées (10 plotting librairies).
La référence reste matplotlib, et la plupart sont pensées pour être intégrées à ses objets (c'est par exemple le cas de seaborn, mpld3, plotly et bokeh). Il est donc utile de commencer par se familiariser avec matplotlib.
Pour reprendre les termes de ses développeurs : *"matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. matplotlib can be used in python scripts, the python and ipython shell (ala MatLab or mathematica), web application servers, and six graphical user interface toolkits."*
La structure sous-jacente de matplotlib est très générale et personnalisable (gestion de l'interface utilisateur, possibilité d'intégration dans des applications web, etc.). Heureusement, il n'est pas nécessaire de maîtriser l'ensemble de ces méthodes pour produire un graphe (il existe pas moins de 2840 pages de documentation).
Pour générer des graphes et les modifier, il suffit de passer par l'interface pyplot.
L'interface pyplot est inspirée de celle de MATLAB. Ceux qui la connaissent s'y retrouveront rapidement.
Pour résumer : - matplotlib - accès "low level" à la librairie de visualisation. Utile si vous souhaitez créer votre propre librairie de visualisation python ou faire des choses très custom. - matplotlib.pyplot - interface proche de celle de Matplab pour produire vos graphes - pylab - matplotlib.pyplot + numpy

[2]:
```
#Pour intégrer les graphes à votre notebook, il suffit de faire
%matplotlib inline
```

[3]:
```
#ou alors
%pylab inline
#pylab charge également numpy. C'est la commande du calcul scientifique python.
```

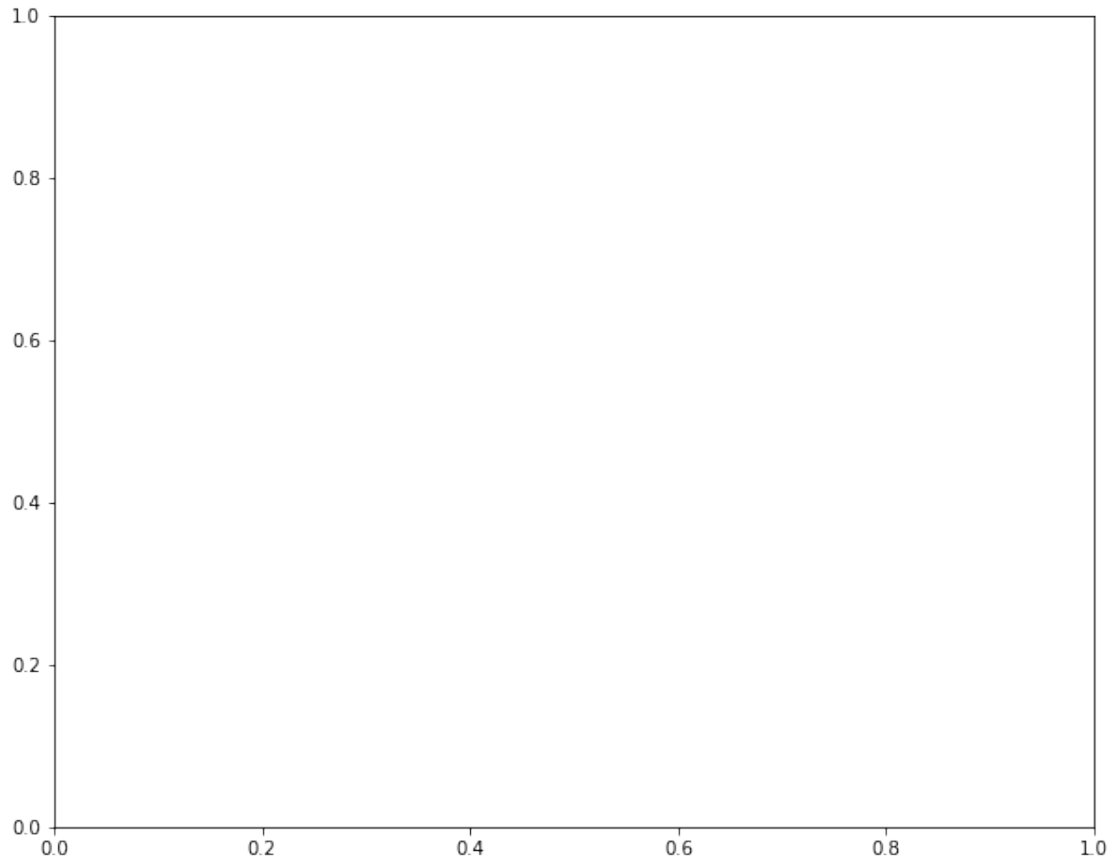
```
Populating the interactive namespace from numpy and matplotlib
```

La structure des objets décrits par l'API est très hiérarchique, comme illustré par ce schéma : - "Figure" contient l'ensemble de la représentation visuelle. C'est par exemple grâce à cette méta-structure que l'on peut facilement ajouter un titre à une représentation qui contiendrait plusieurs graphes ; - "Axes" (ou "Subplots") décrit l'ensemble contenant un ou pusieurs graphes (correspond à l'objet subplot et aux méthodes add_subplot) - "Axis" correspond aux axes d'un graphique (ou instance de subplot) donné.
Une dernière remarque d'ordre général : pyplot est une machine à état. Cela implique que les méthodes pour tracer un graphe ou éditer un label s'appliquent par défaut au dernier état en cours (dernière instance de subplot ou dernière instance d'axe par exemple).
Conséquence : il faut concevoir ses codes comme une séquence d'instructions (par exemple, il ne faut pas séparer les instructions qui se rapportent au même graphique dans deux cellules différentes du Notebook).

1.0.1 Figures et Subplots

[4]:
```
from matplotlib import pyplot as plt
plt.figure(figsize=(10,8))
plt.subplot(111) # Méthode subplot : pour définir les graphiques appartenant à l'objet␣
 ↪figure, ici 1 X 1, indice 1
#plt.subplot(1,1,1) fonctionne aussi
#attention, il est nécessaire de conserver toutes les instructions d'un même graphique␣
 ↪dans le même bloc
#pas besoin de plt.show() dans un notebook, sinon c'est nécessaire
```

[4]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2219e493c88>
```

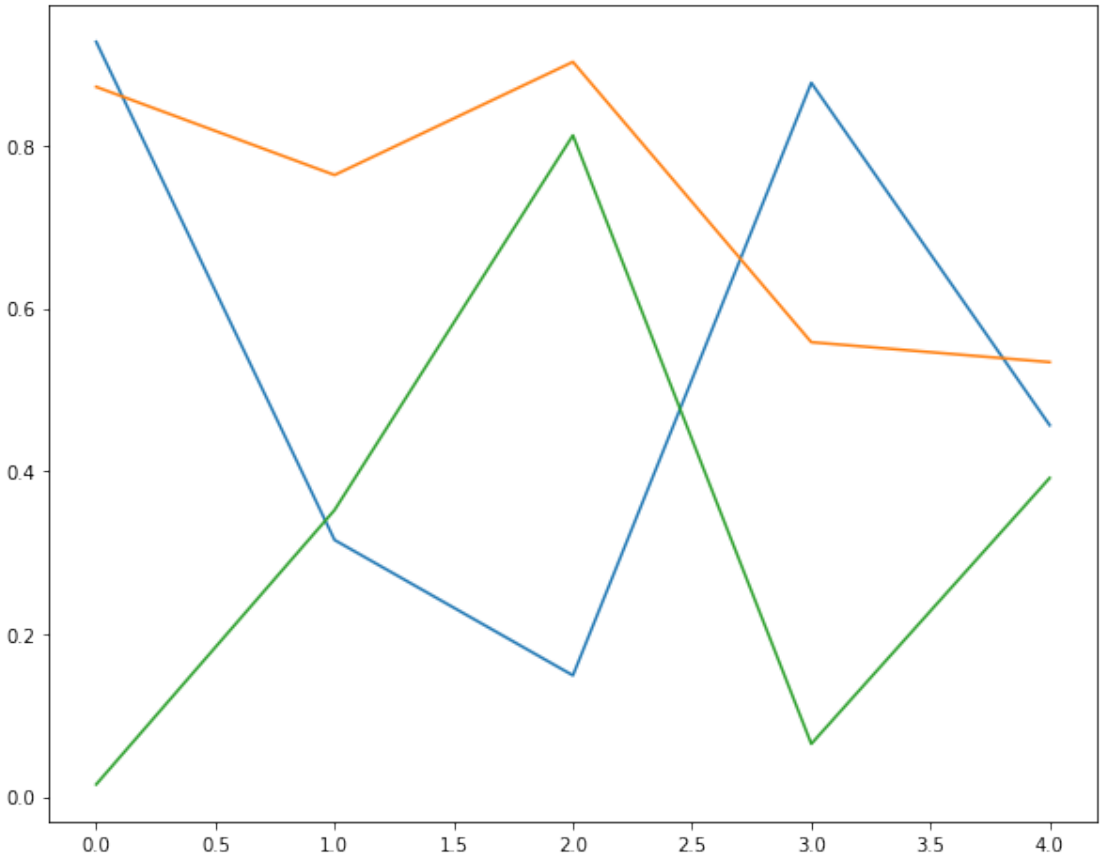


Un graphique (très) simple avec l'instruction plot.

[5]:
```
from numpy import random
import numpy as np
import pandas as p

plt.figure(figsize=(10,8))
plt.subplot(111)
plt.plot([random.random_sample(1) for i in range(5)])
#Il est possible de passer des listes, des arrays de numpy, des Series et des␣
 ↪Dataframes de pandas
```

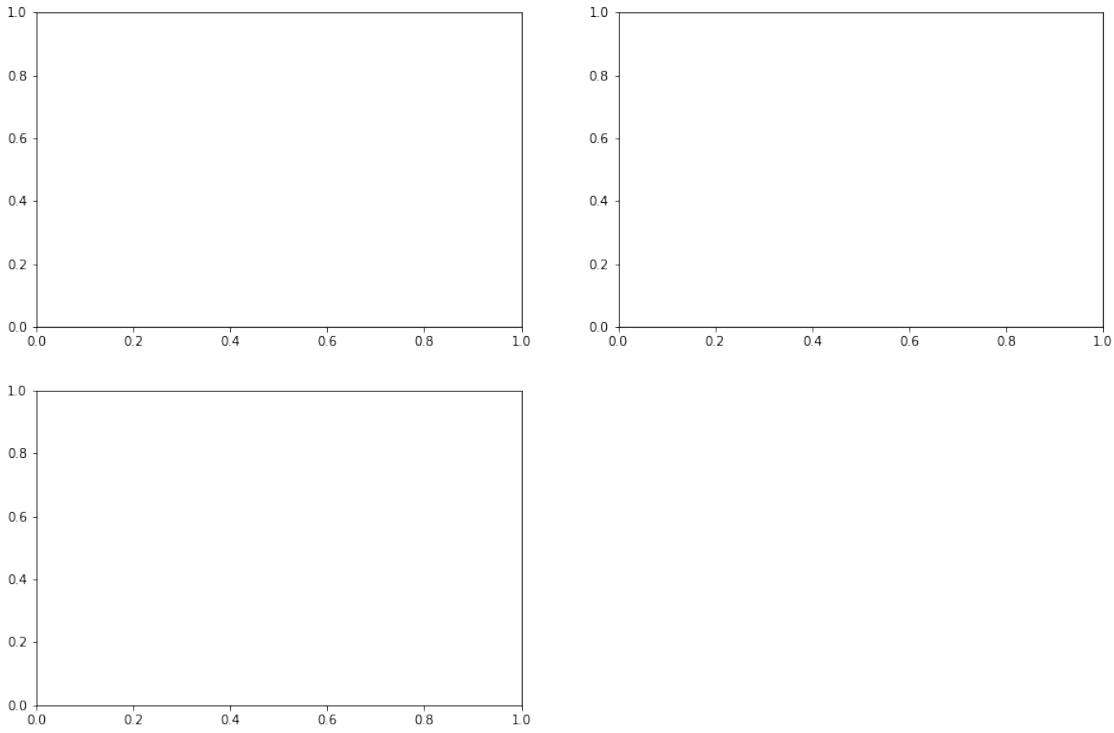
```
plt.plot(np.array([random.random_sample(1) for i in range(5)]))
plt.plot(p.DataFrame(np.array([random.random_sample(1) for i in range(5)])))
#pour afficher plusieurs courbes, il suffit de cumuler les instructions plt.plot
#plt.show()
```

[5]: [<matplotlib.lines.Line2D at 0x2219f57e748>]



Pour faire plusieurs sous graphes, il suffit de modifier les valeurs des paramètres de l'objet subplot.

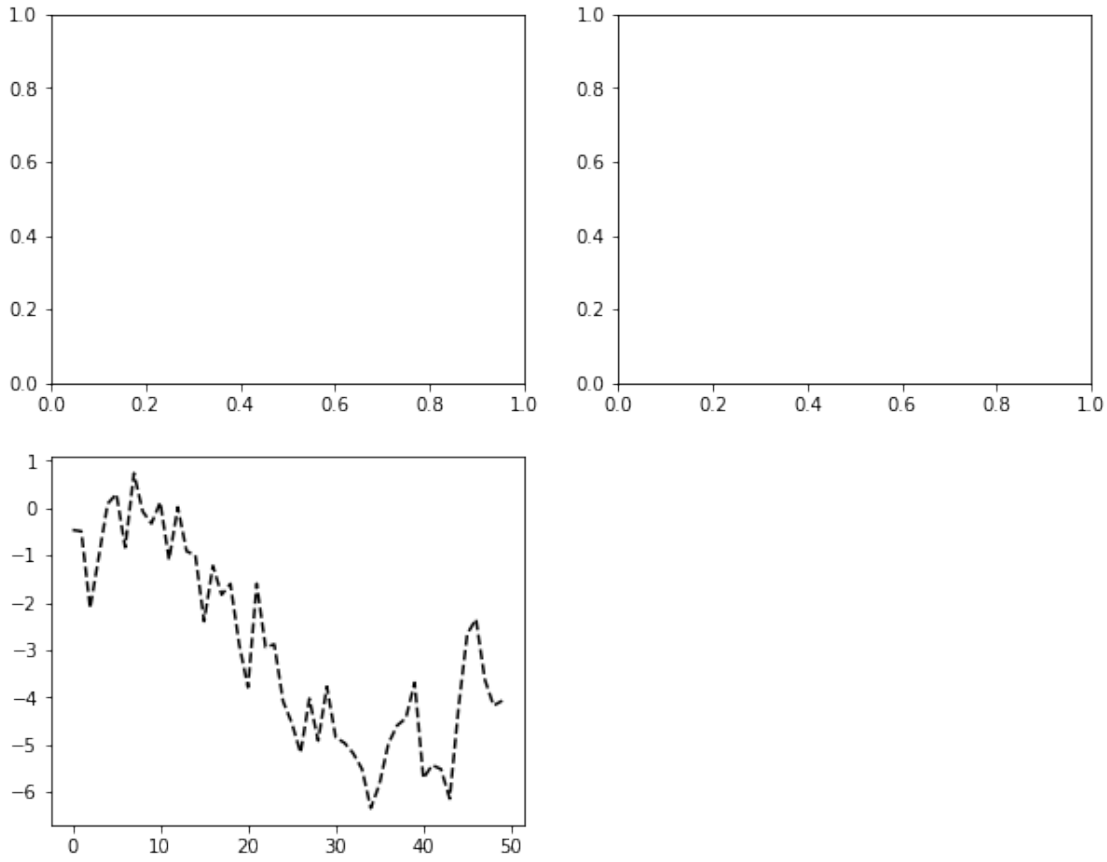
[6]:
```
fig = plt.figure(figsize=(15,10))
ax1 = fig.add_subplot(2,2,1) #modifie l'objet fig et créé une nouvelle instance de␣
 ↪subplot, appelée ax1
#vous verrez souvent la convention ax comme instance de subplot : c'est parce que l'on␣
 ↪parle aussi d'objet "Axe"
#à ne pas confondre avec l'objet "Axis"
ax2 = fig.add_subplot(2,2,2)
ax3 = fig.add_subplot(2,2,3)
```



Si aucune instance d'axes n'est précisée, la méthode plot est appliquée à la dernière instance créée.

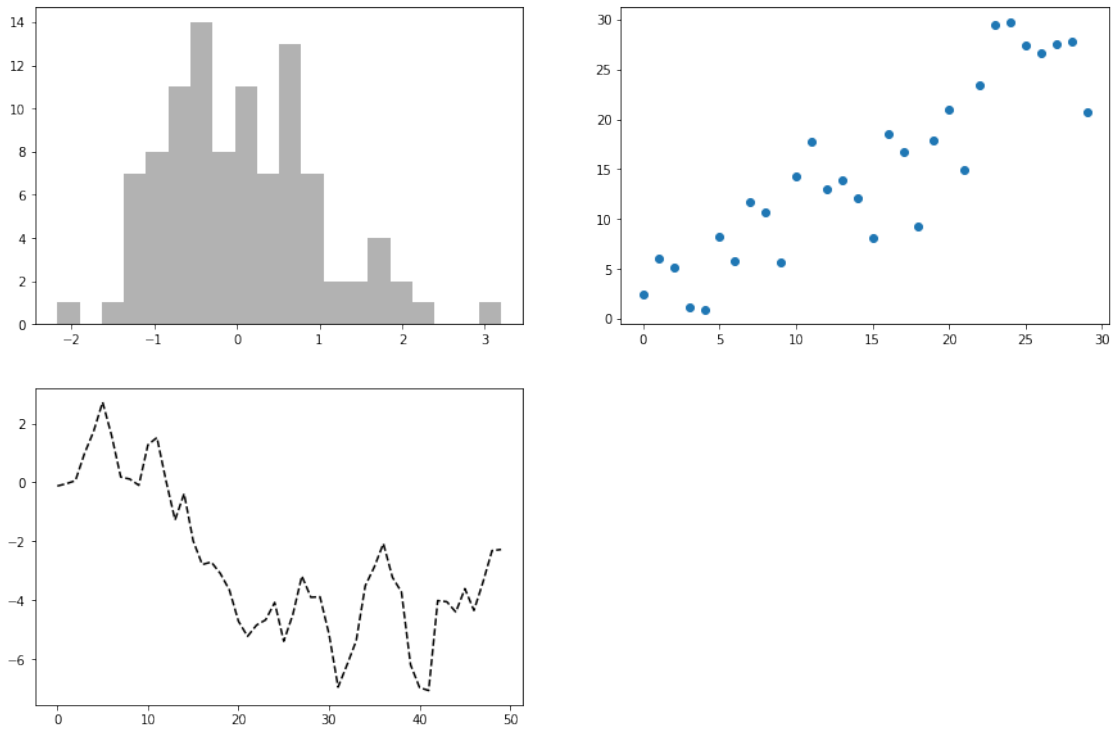
[7]:
```
from numpy.random import randn

fig = plt.figure(figsize=(10,8))
ax1 = fig.add_subplot(2,2,1)
ax2 = fig.add_subplot(2,2,2)
ax3 = fig.add_subplot(2,2,3)
plt.plot(randn(50).cumsum(),'k--')
# plt.show()
```

[7]:
```
[<matplotlib.lines.Line2D at 0x2219f71c4e0>]
```



[8]:
```
from numpy.random import randn

fig = plt.figure(figsize=(15,10))
ax1 = fig.add_subplot(2,2,1)
ax2 = fig.add_subplot(2,2,2)
ax3 = fig.add_subplot(2,2,3)

# On peut compléter les instances de sous graphiques par leur contenu.
# Au passage, quelques autres exemples de graphes
ax1.hist(randn(100),bins=20,color='k',alpha=0.3)
ax2.scatter(np.arange(30),np.arange(30)+3*randn(30))
ax3.plot(randn(50).cumsum(),'k--')
```

[8]:
```
[<matplotlib.lines.Line2D at 0x2219f96e240>]
```

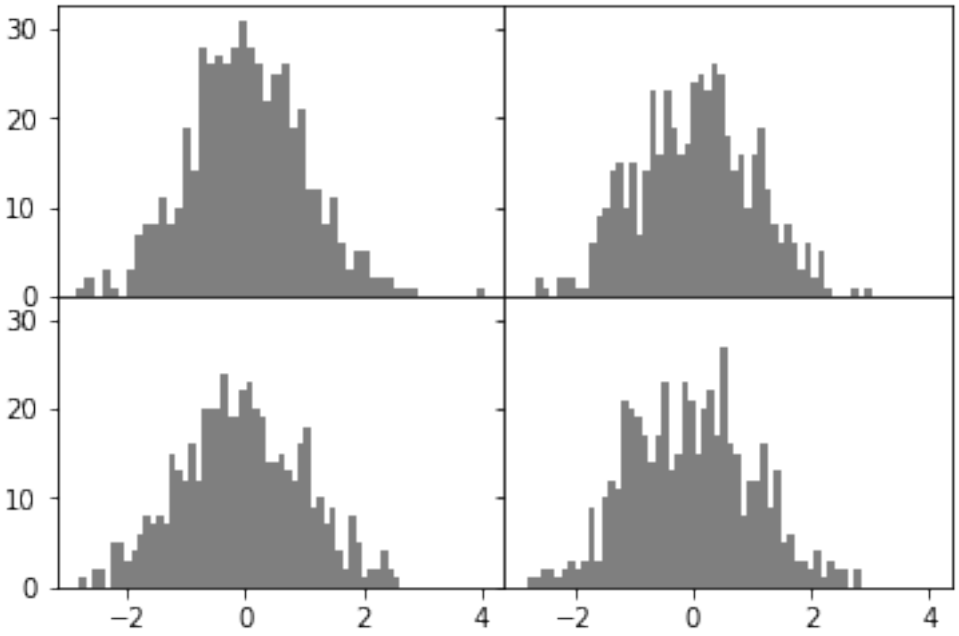


Pour explorer l'ensemble des catégories de graphiques possibles : Gallery. Les plus utiles pour l'analyse de données : scatter, scatterhist, barchart, stackplot, histogram, cumulative distribution function, boxplot, , radarchart.

1.0.2 Ajuster les espaces entre les graphes

[9]:
```
fig,axes = plt.subplots(2,2,sharex=True,sharey=True)
# Sharex et sharey portent bien leurs noms : si True, ils indiquent que les␣
 ↪sous-graphiques
# ont des axes paramétrés de la même manière
for i in range(2):
    for j in range(2):
        axes[i,j].hist(randn(500),bins=50,color='k',alpha=0.5)
# L'objet "axes" est un 2darray, simple à indicer et parcourir avec une boucle
print(type(axes))

# N'h'ésitez pas à faire varier les paramètres qui vous posent question. Par exemple,␣
 ↪à quoi sert alpha ?
plt.subplots_adjust(wspace=0,hspace=0)
# Cette dernière méthode permet de supprimer les espaces entres les sous graphes.
```

```
<class 'numpy.ndarray'>
```



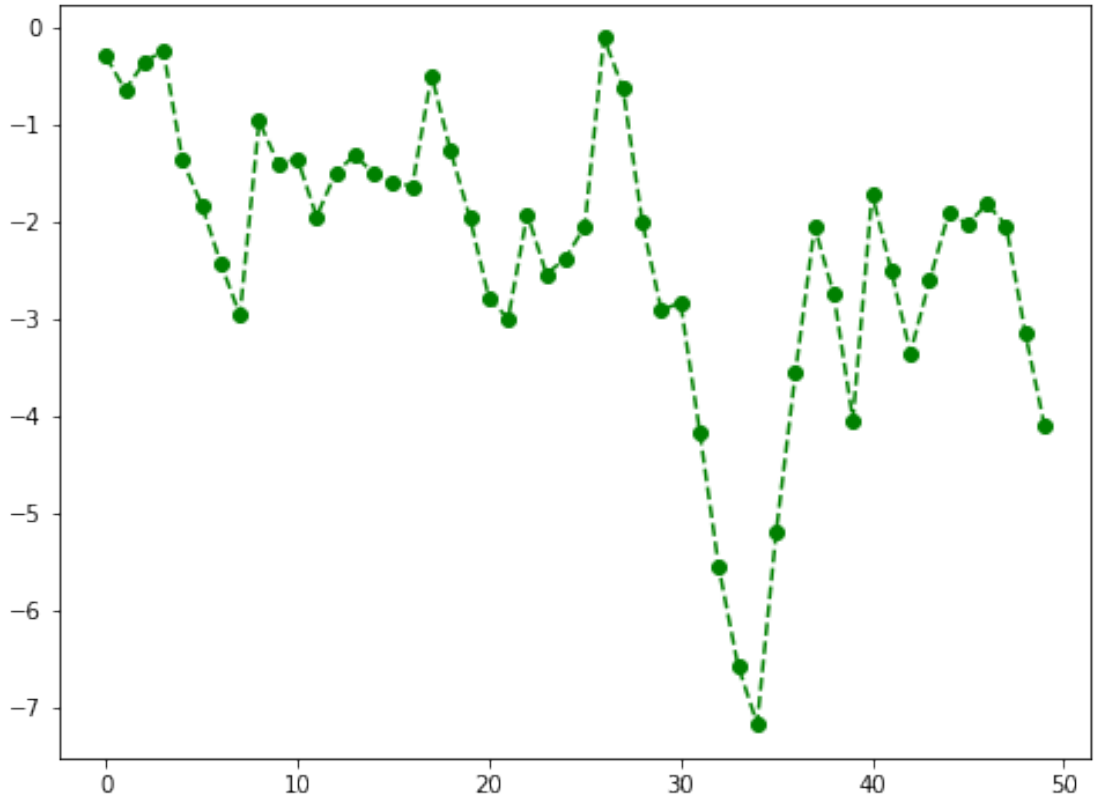
Pas d'autres choix que de paramétrer à la main pour corriger les chiffres qui se superposent.

1.0.3 Couleurs, Marqueurs et styles de ligne

MatplotLib offre la possibilité d'adopter deux types d'écriture : chaîne de caractère condensée ou paramétrage explicite via un système clé-valeur.

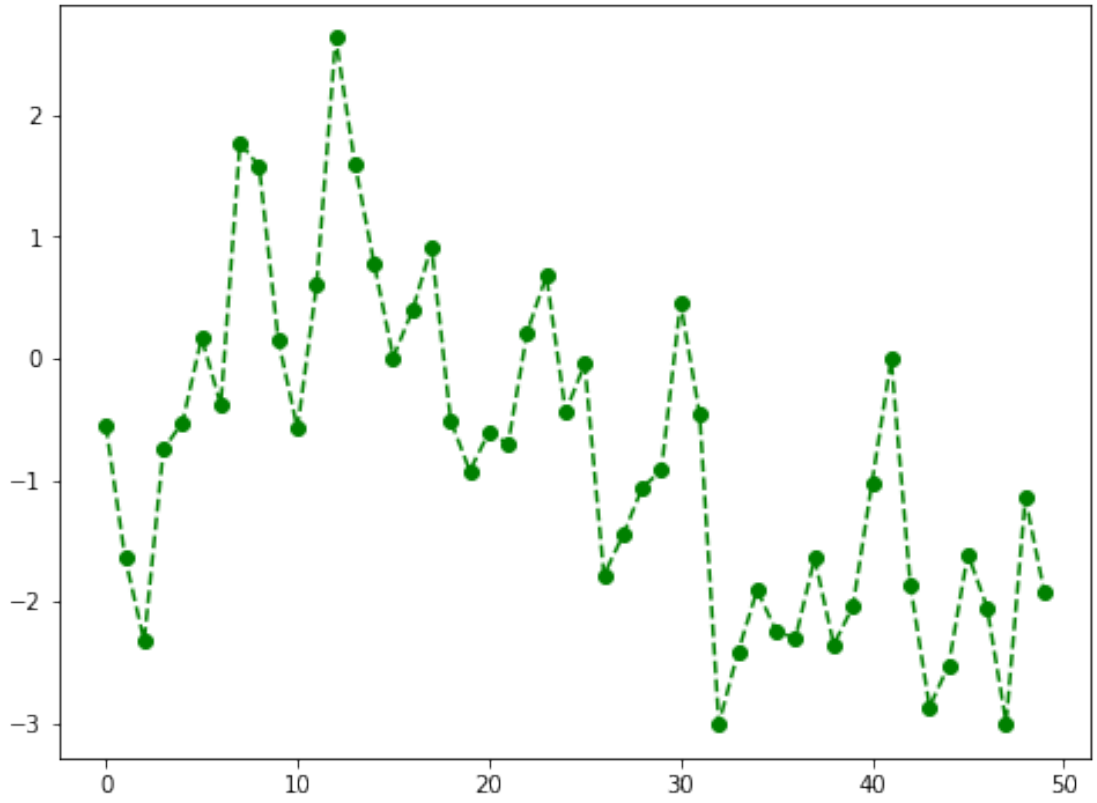
[10]:
```
from numpy.random import randn

fig = plt.figure(figsize=(8,6))
ax1 = fig.add_subplot(111)
ax1.plot(randn(50).cumsum(),color='g',marker='o',linestyle='dashed')
# plt.show()
```

[10]:
```
[<matplotlib.lines.Line2D at 0x2219fcc19b0>]
```



[11]:
```
from numpy.random import randn

fig = plt.figure(figsize=(8,6))
ax1 = fig.add_subplot(111)
ax1.plot(randn(50).cumsum(),'og--') #l'ordre des paramètres n'importe pas
```

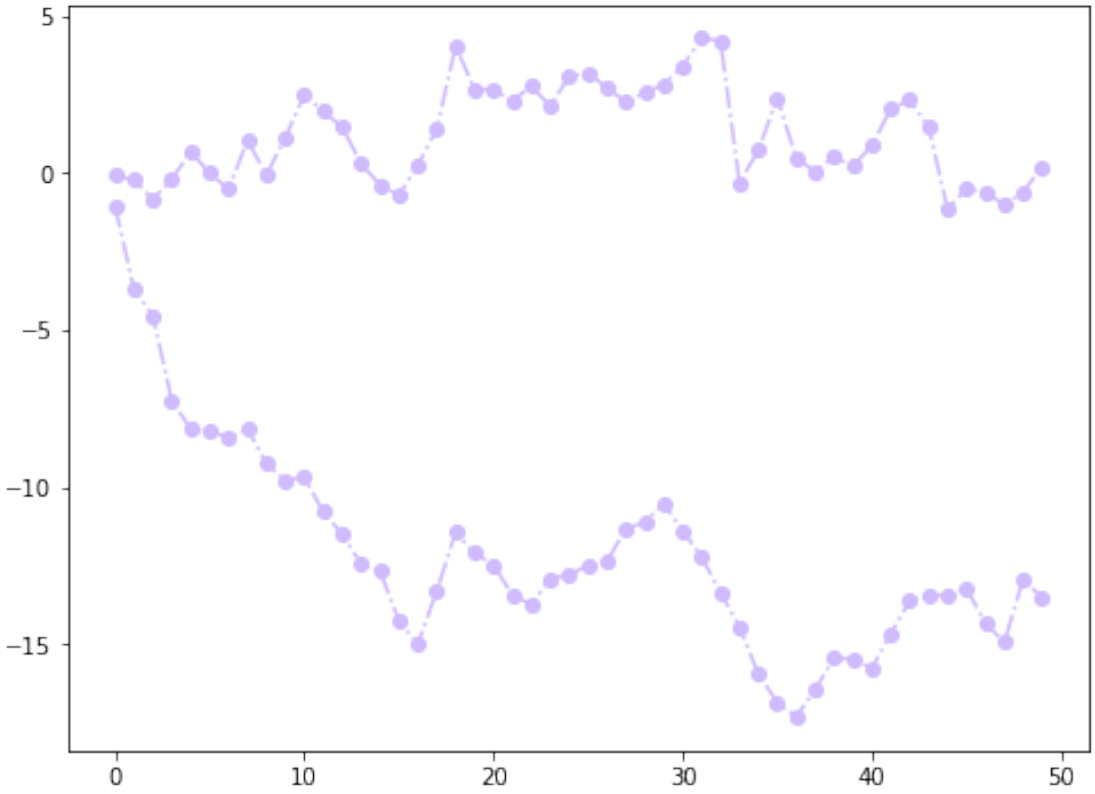
[11]: [<matplotlib.lines.Line2D at 0x2219fd19ba8>]



Plus de détails dans la documentation sur l'API de matplotlib pour paramétrer la couleur , les markers , et le style des lignes . MatplotLib est compatible avec plusieurs standards de couleur : - sous forme d'une lettre : 'b' = blue (bleu), 'g' = green (vert), 'r' = red (rouge), 'c' = cyan (cyan), 'm' = magenta (magenta), 'y' = yellow (jaune), 'k' = black (noir), 'w' = white (blanc). - sous forme d'un nombre entre 0 et 1 entre quotes qui indique le niveau de gris : par exemple '0.70' ('1' = blanc, '0' = noir). - sous forme d'un nom : par exemple 'red'. - sous forme html avec les niveaux respectifs de rouge (R), vert (G) et bleu (B) : '#ffee00'. Voici un site pratique pour récupérer une couleur en RGB hexadécimal. - sous forme d'un triplet de valeurs entre 0 et 1 avec les niveaux de R, G et B : (0.2, 0.9, 0.1).

[12]:
```
from numpy.random import randn

fig = plt.figure(figsize=(8,6))
ax1 = fig.add_subplot(1,1,1)
#avec la norme RGB
ax1.plot(randn(50).cumsum(),color='#D0BBFF',marker='o',linestyle='-.')
ax1.plot(randn(50).cumsum(),color=(0.8156862745098039, 0.7333333333333333, 1.
↪0),marker='o',linestyle='-.')
```

[12]:
```
[<matplotlib.lines.Line2D at 0x2219fd82550>]
```



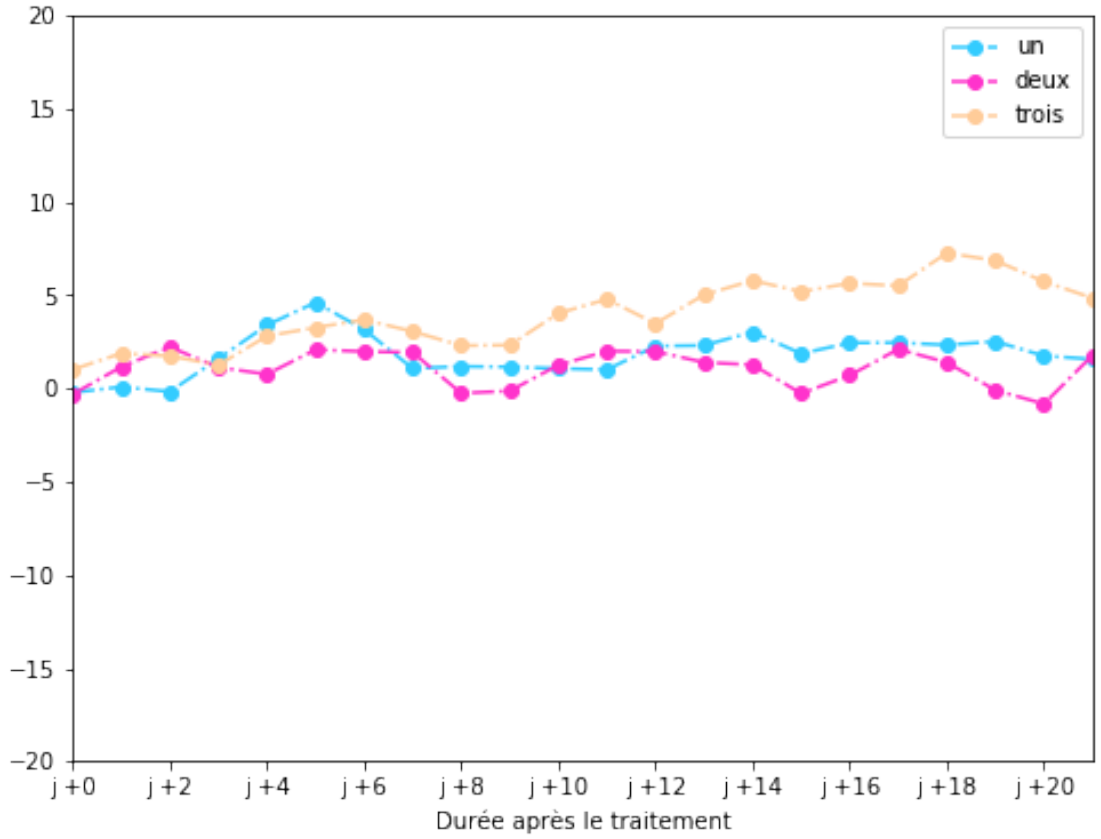
1.0.4 Ticks labels et legendes

3 méthodes clés : - xlim() : pour délimiter l'étendue des valeurs de l'axe - xticks() : pour passer les graduations sur l'axe - xticklabels() : pour passer les labels
Pour l'axe des ordonnées c'est ylim, yticks, yticklabels.
Pour récupérer les valeurs fixées : - plt.xlim() ou plt.get_xlim() - plt.xticks() ou plt.get_xticks() - plt.xticklabels() ou plt.get_xticklabels()
Pour fixer ces valeurs : - plt.xlim([start,end]) ou plt.set_xlim([start,end]) - plt.xticks(my_ticks_list) ou plt.get_xticks(my_ticks_list) - plt.xticklabels(my_labels_list) ou plt.get_xticklabels(my_labels_list)
Si vous voulez customiser les axes de plusieurs sous graphiques, passez par une instance de axis et non subplot.

[13]:
```
from numpy.random import randn

fig = plt.figure(figsize=(8,6))
ax1 = fig.add_subplot(1,1,1)

serie1=randn(50).cumsum()
serie2=randn(50).cumsum()
serie3=randn(50).cumsum()
ax1.plot(serie1,color='#33CCFF',marker='o',linestyle='-.',label='un')
ax1.plot(serie2,color='#FF33CC',marker='o',linestyle='-.',label='deux')
ax1.plot(serie3,color='#FFCC99',marker='o',linestyle='-.',label='trois')

#sur le graphe précédent, pour raccourcir le range
```

```
ax1.set_xlim([0,21])
ax1.set_ylim([-20,20])
#faire un ticks avec un pas de 2 (au lieu de 5)
ax1.set_xticks(range(0,21,2))
#changer le label sur la graduation
ax1.set_xticklabels(["j +" + str(l) for l in range(0,21,2)])
ax1.set_xlabel('Durée après le traitement')

ax1.legend(loc='best')
#permet de choisir l'endroit le plus vide
```

[13]: <matplotlib.legend.Legend at 0x2219ff38cf8>

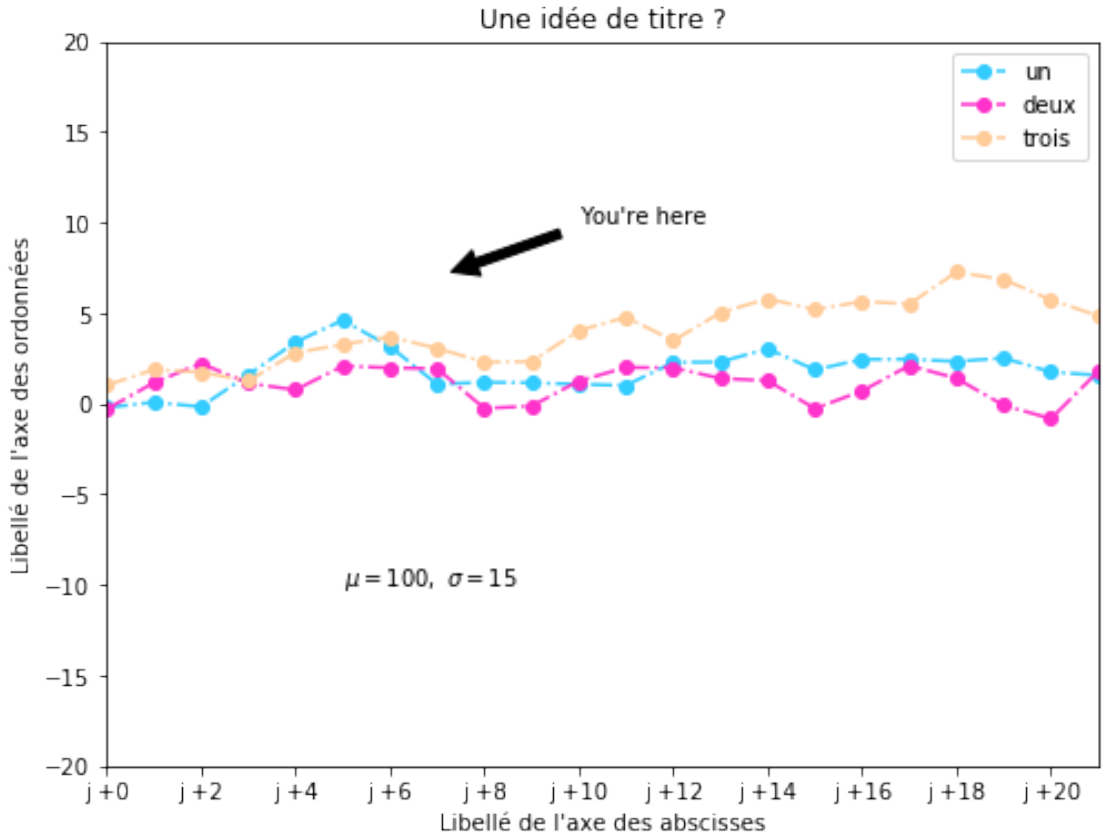


1.0.5 Inclusion d'annotation et de texte, titre et libellé des axes

[14]:
```
from numpy.random import randn

fig = plt.figure(figsize=(8,6))
ax1 = fig.add_subplot(1,1,1)
ax1.plot(serie1,color='#33CCFF',marker='o',linestyle='-.',label='un')
ax1.plot(serie2,color='#FF33CC',marker='o',linestyle='-.',label='deux')
ax1.plot(serie3,color='#FFCC99',marker='o',linestyle='-.',label='trois')
```

```
ax1.set_xlim([0,21])
ax1.set_ylim([-20,20])
ax1.set_xticks(range(0,21,2))
ax1.set_xticklabels(["j +" + str(l) for l in range(0,21,2)])
ax1.set_xlabel('Durée après le traitement')

ax1.annotate("You're here", xy=(7, 7), #point de départ de la flèche
             xytext=(10, 10),          #position du texte
            arrowprops=dict(facecolor='#000000', shrink=0.10),
            )

ax1.legend(loc='best')

plt.xlabel("Libellé de l'axe des abscisses")
plt.ylabel("Libellé de l'axe des ordonnées")
plt.title("Une idée de titre ?")
plt.text(5, -10, r'$\mu=100,\ \sigma=15$')

# plt.show()
```

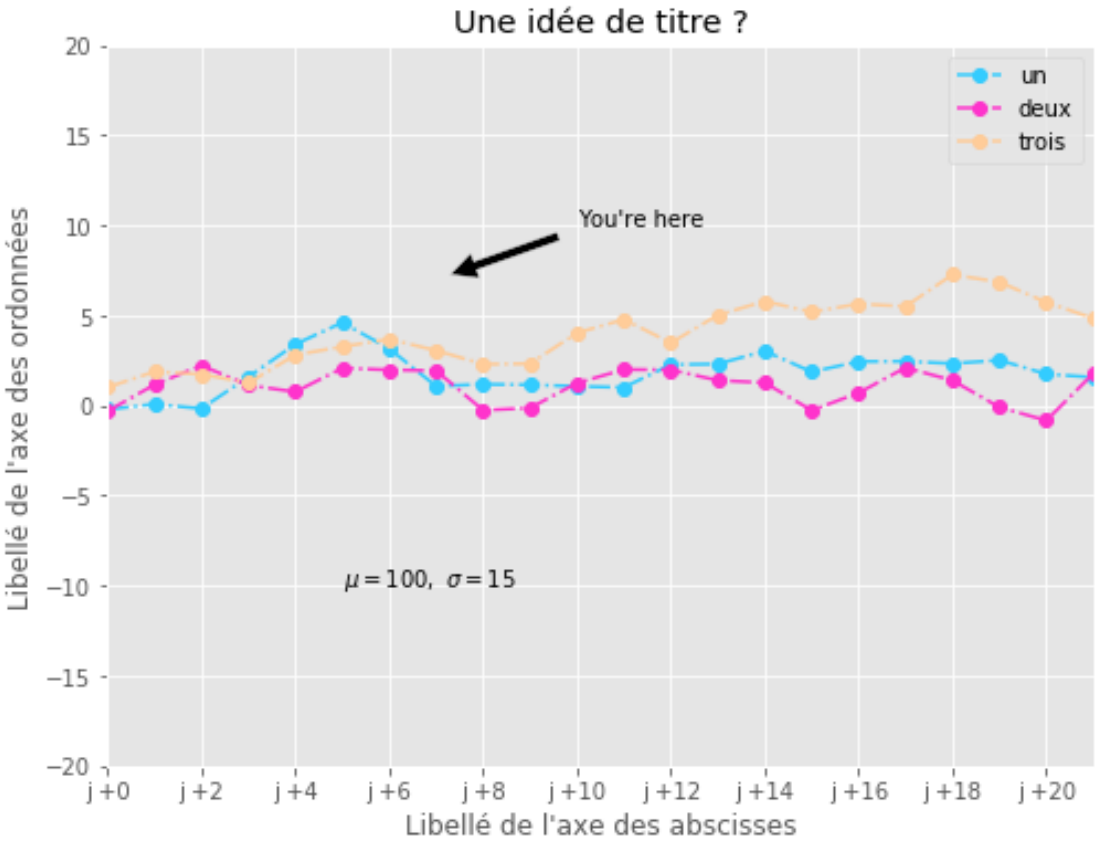
[14]: `Text(5,-10,'$\\mu=100,\\ \\sigma=15$')`



1.0.6 matplotlib et le style

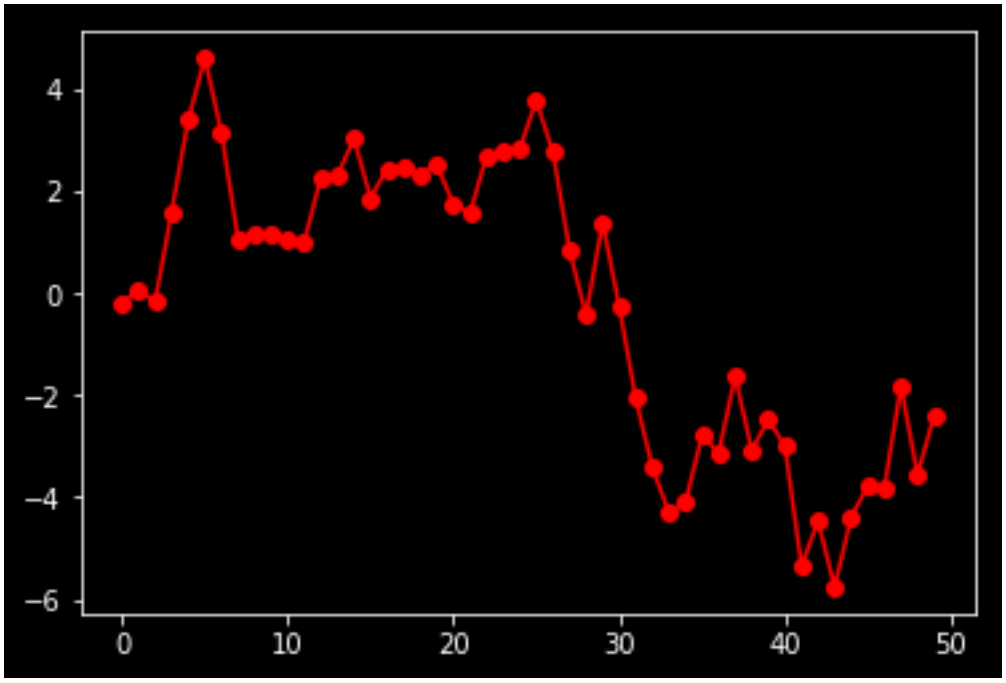
Il est possible de définir son propre style. Cette possibilité est intéressante si vous faîtes régulièrement les mêmes graphes et voulez définir des templates (plutôt que de copier/coller toujours les mêmes lignes de code). Tout est décrit dans style_sheets.

[15]:
```
from numpy.random import randn

#pour que la définition du style soit seulement dans cette cellule notebook
with plt.style.context('ggplot'):
    fig = plt.figure(figsize=(8,6))
    ax1 = fig.add_subplot(1,1,1)
    ax1.plot(serie1,color='#33CCFF',marker='o',linestyle='-.',label='un')
    ax1.plot(serie2,color='#FF33CC',marker='o',linestyle='-.',label='deux')
    ax1.plot(serie3,color='#FFCC99',marker='o',linestyle='-.',label='trois')

    ax1.set_xlim([0,21])
    ax1.set_ylim([-20,20])
    ax1.set_xticks(range(0,21,2))
    ax1.set_xticklabels(["j +" + str(l) for l in range(0,21,2)])
    ax1.set_xlabel('Durée après le traitement')

    ax1.annotate("You're here", xy=(7, 7), #point de départ de la flèche
                 xytext=(10, 10),          #position du texte
                arrowprops=dict(facecolor='#000000', shrink=0.10),
                )

    ax1.legend(loc='best')

    plt.xlabel("Libellé de l'axe des abscisses")
    plt.ylabel("Libellé de l'axe des ordonnées")
    plt.title("Une idée de titre ?")
    plt.text(5, -10, r'$\mu=100,\ \sigma=15$')

    #plt.show()
```



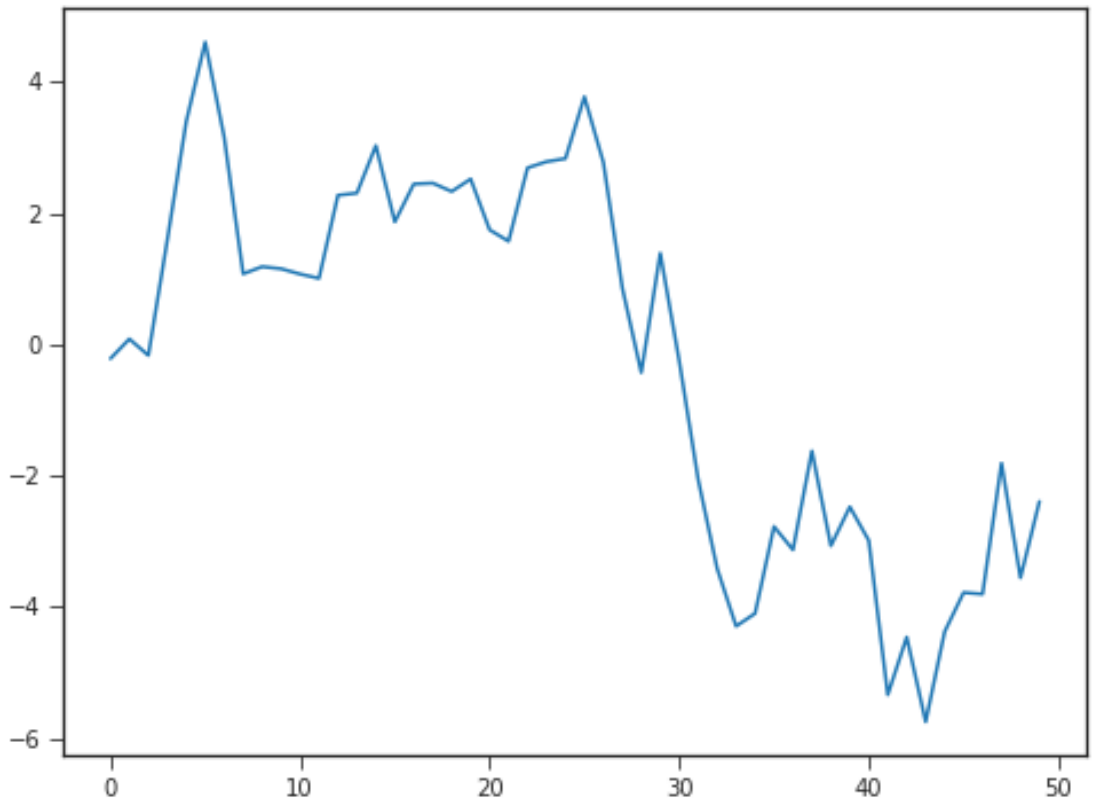
[16]:
```
import numpy as np
import matplotlib.pyplot as plt

print("De nombreux autres styles sont disponibles, pick up your choice! ", plt.style.
  ↪available)
with plt.style.context('dark_background'):
    plt.plot(serie1, 'r-o')

# plt.show()
```

```
De nombreux autres styles sont disponibles, pick up your choice!  ['bmh',
'classic', 'dark_background', 'fast', 'fivethirtyeight', 'ggplot', 'grayscale',
'seaborn-bright', 'seaborn-colorblind', 'seaborn-dark-palette', 'seaborn-dark',
'seaborn-darkgrid', 'seaborn-deep', 'seaborn-muted', 'seaborn-notebook',
'seaborn-paper', 'seaborn-pastel', 'seaborn-poster', 'seaborn-talk', 'seaborn-
ticks', 'seaborn-white', 'seaborn-whitegrid', 'seaborn', 'Solarize_Light2',
'_classic_test']
```



Comme suggéré dans le nom des styles disponibles dans matplotlib, la librairie seaborn, qui est une sorte de surcouche de matplotlib, est un moyen très pratique d'accéder à des styles pensés et adaptés pour la mise en valeur de pattern dans les données.
Voici quelques exemples, toujours sur la même série de données. Je vous invite également à explorer les palettes de couleurs.

[17]:
```
#on peut remarquer que le style ggplot est resté.
import seaborn as sns

#5 styles disponibles
#sns.set_style("whitegrid")
#sns.set_style("darkgrid")
#sns.set_style("white")
#sns.set_style("dark")
#sns.set_style("ticks")

#si vous voulez définir un style temporairement
with sns.axes_style("ticks"):
    fig = plt.figure(figsize(8,6))
    ax1 = fig.add_subplot(1,1,1)
    plt.plot(serie1)
```



En dehors du style et des couleurs, Seaborn a mis l'accent sur : - les graphes de distribution (univariés / bivariés). Particulièrement utiles et pratiques : les pairwiseplot - les graphes de régression - les graphes de variables catégorielles - les heatmap sur les matrices de données
Seaborn ce sont des graphes pensés pour l'analyse de données et la présentation de rapports à des collègues ou clients. C'est peut-être un peu moins customisable que matplotlib mais vous avez le temps avant de vous sentir limités dans les possibilités.

2 Matplotlib et pandas, intéractions avec seaborn

Comme vu précédemment, matplotlib permet de manipuler et de représenter sous forme de graphes toutes sortes d'objets : listes, arrays numpy, Series et DataFrame pandas. Inversement, pandas a prévu des méthodes qui intègrent les objets matplotlib les plus utiles pour le tracé de graphiques. Nous allons tester un peu l'intégration pandas/matplotlib. D'une amanière générale, tout un écosystème de visualisation s'est développé autour de pandas. Nous allons tester les différentes librairies évoquées. Télécharger les données de l'exercice 4 du TD sur pandas et disponible sur le site de l'INSEE Naissances, décès et mariages de 1998 à 2013.

[18]:
```
import urllib.request
import zipfile

def download_and_save(name, root_url):
    if root_url == 'xd':
        from pyensae.datasource import download_data
        download_data(name)
    else:
```

```
        response = urllib.request.urlopen(root_url+name)
        with open(name, "wb") as outfile:
            outfile.write(response.read())

def unzip(name):
    with zipfile.ZipFile(name, "r") as z:
        z.extractall(".")

filenames = ["etatcivil2012_mar2012_dbase.zip",
             "etatcivil2012_nais2012_dbase.zip",
             "etatcivil2012_dec2012_dbase.zip", ]

# Une copie des fichiers a été postée sur le site www.xavierdupre.fr
# pour tester le notebook plus facilement.
root_url = 'xd' # http://telechargement.insee.fr/fichiersdetail/etatcivil2012/dbase/'

for filename in filenames:
    download_and_save(filename, root_url)
    unzip(filename)
    print("Download of {}: DONE!".format(filename))
```

```
Download of etatcivil2012_mar2012_dbase.zip: DONE!
Download of etatcivil2012_nais2012_dbase.zip: DONE!
Download of etatcivil2012_dec2012_dbase.zip: DONE!
```

Penser à installer le module dbfread si ce n'est pas fait.

[19]:
```
import pandas
try:
    from dbfread import DBF
    use_dbfread = True
except ImportError as e :
    use_dbfread = False

if use_dbfread:
    print("use of dbfread")
    def dBase2df(dbase_filename):
        table = DBF(dbase_filename, load=True, encoding="cp437")
        return pandas.DataFrame(table.records)

    df = dBase2df('mar2012.dbf')
    #df.to_csv("mar2012.txt", sep="\t", encoding="utf8", index=False)
else :
    print("use of zipped version")
    import pyensae.datasource
    data = pyensae.datasource.download_data("mar2012.zip")
    df = pandas.read_csv(data[0], sep="\t", encoding="utf8", low_memory = False)

df.shape, df.columns
```

```
use of dbfread
```

[19]:
```
((246123, 16),
 Index(['ANAISH', 'DEPNAISH', 'INDNATH', 'ETAMATH', 'ANAISF', 'DEPNAISF',
        'INDNATF', 'ETAMATF', 'AMAR', 'MMAR', 'JSEMAINE', 'DEPMAR', 'DEPDOM',
        'TUDOM', 'TUCOM', 'NBENFCOM'],
       dtype='object'))
```

Dictionnaire des variables.

[20]:
```
vardf = dBase2df("varlist_mariages.dbf")
print(vardf.shape, vardf.columns)
vardf
```

```
(16, 4) Index(['VARIABLE', 'LIBELLE', 'TYPE', 'LONGUEUR'], dtype='object')
```

[20]:

	VARIABLE	LIBELLE	TYPE	\
0	AMAR	Année du mariage	CHAR	
1	ANAISF	Année de naissance de l'épouse	CHAR	
2	ANAISH	Année de naissance de l'époux	CHAR	
3	DEPDOM	Département de domicile après le mariage	CHAR	
4	DEPMAR	Département de mariage	CHAR	
5	DEPNAISF	Département de naissance de l'épouse	CHAR	
6	DEPNAISH	Département de naissance de l'époux	CHAR	
7	ETAMATF	État matrimonial antérieur de l'épouse	CHAR	
8	ETAMATH	État matrimonial antérieur de l'époux	CHAR	
9	INDNATF	Indicateur de nationalité de l'épouse	CHAR	
10	INDNATH	Indicateur de nationalité de l'époux	CHAR	
11	JSEMAINE	Jour du mariage dans la semaine	CHAR	
12	MMAR	Mois du mariage	CHAR	
13	NBENFCOM	Enfants en commun avant le mariage	CHAR	
14	TUCOM	Tranche de commune du lieu de domicile des époux	CHAR	
15	TUDOM	Tranche d'unité urbaine du lieu de domicile de…	CHAR	

	LONGUEUR
0	4
1	4
2	4
3	3
4	3
5	3
6	3
7	1
8	1
9	1
10	1
11	1
12	2
13	1
14	1
15	1

Représentez l'age des femmes en fonction de celui des hommes au moment du mariage.

[21]:
```
#Calcul de l'age (au moment du mariage)
df.head()
```

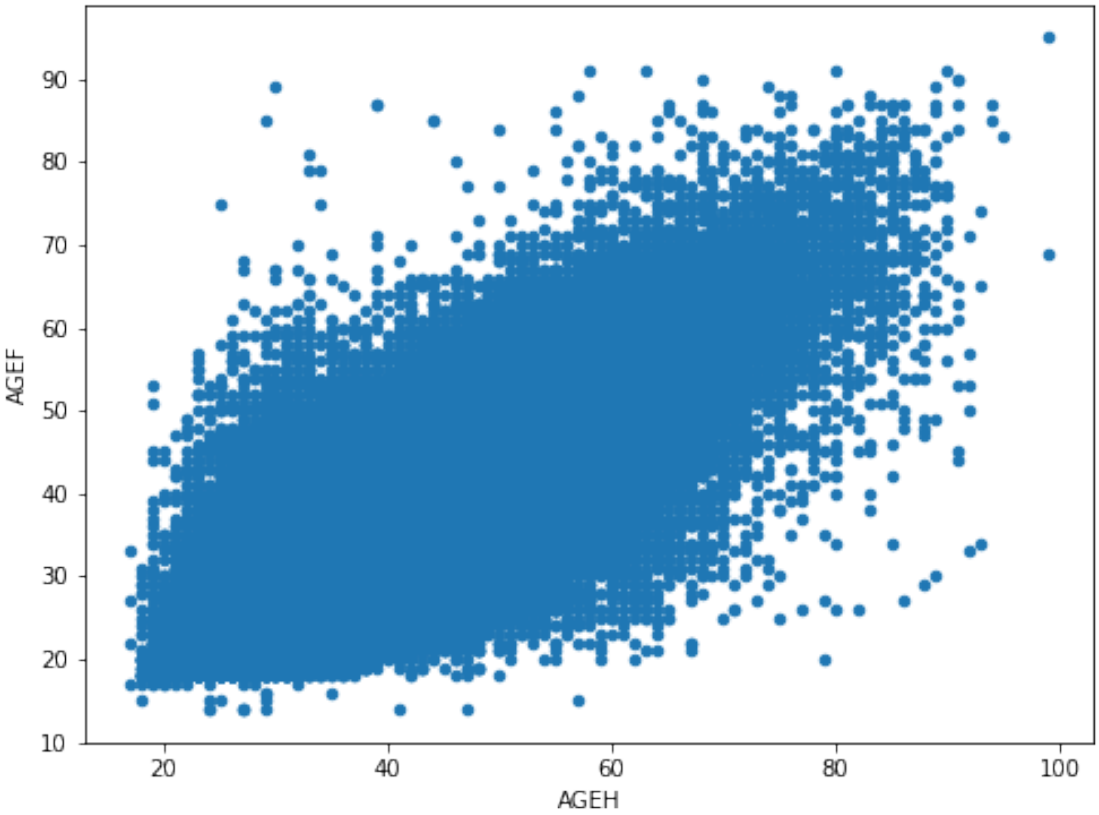
[21]:
```
  ANAISH DEPNAISH INDNATH ETAMATH ANAISF DEPNAISF INDNATF ETAMATF  AMAR MMAR  \
0   1982       75       1       1   1984       99       2       1  2012   01
1   1956       69       2       4   1969       99       2       4  2012   01
2   1982       99       2       1   1992       99       1       1  2012   01
3   1985       99       2       1   1987       84       1       1  2012   01
4   1968       99       2       1   1963       99       2       1  2012   01

  JSEMAINE DEPMAR DEPDOM TUDOM TUCOM NBENFCOM
0        1     29     99     9              N
1        3     75     99     9              N
2        5     34     99     9              N
3        4     13     99     9              N
4        6     26     99     9              N
```

[22]:
```
#conversion des années en entiers
for c in ['AMAR','ANAISF','ANAISH']:
    df[c]=df[c].apply(lambda x: int(x))

#calcul de l'age
df['AGEF'] = df['AMAR'] - df['ANAISF']
df['AGEH'] = df['AMAR'] - df['ANAISH']
```

Le module pandas a prévu un wrapper matplotlib

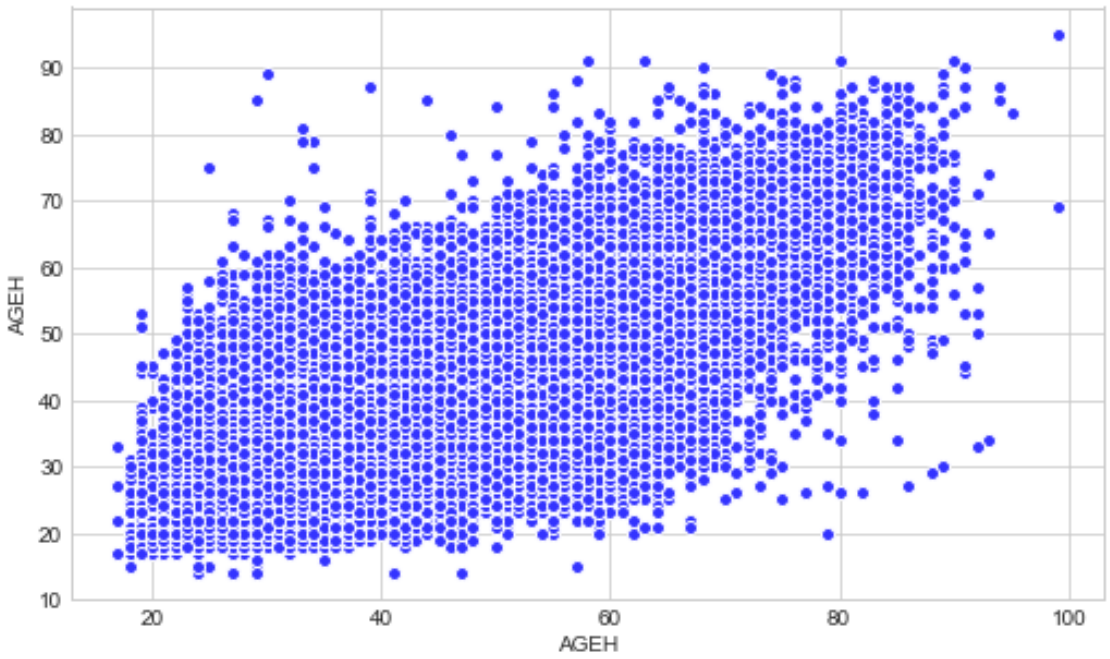
[23]:
```
#version pandas : df.plot()
#deux possibilités : l'option kind dans df.plot()
df.plot(x='AGEH',y='AGEF',kind='scatter')
#ou la méthode scatter()
#df.plot.scatter(x='AGEH',y='AGEF')
#ensemble des graphiques disponibles dans la méthode plot de pandas : df.plot.<TAB>
```

[23]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x221c091bb00>
```



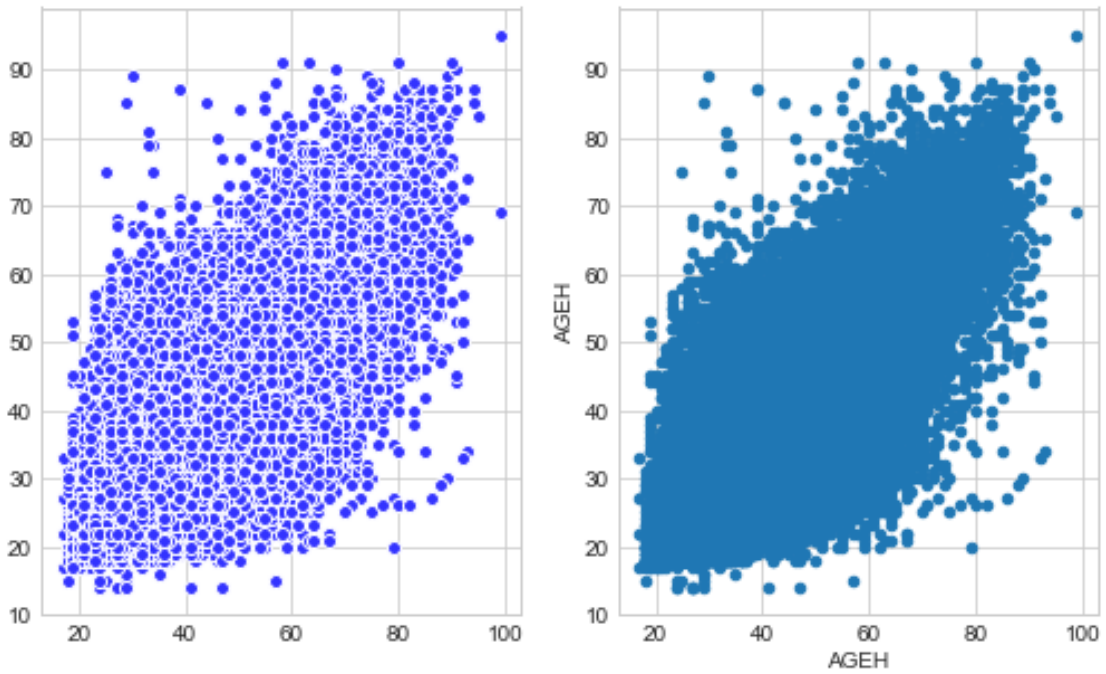
[24]:
```
#version matplotlib

from matplotlib import pyplot as plt

plt.style.use('seaborn-whitegrid')
fig = plt.figure(figsize(8.5,5))
ax = fig.add_subplot(1,1,1)
ax.scatter(df['AGEH'],df['AGEF'], color="#3333FF", edgecolors='#FFFFFF')
plt.xlabel('AGEH')
plt.ylabel('AGEH')
```

[24]:
```
Text(0,0.5,'AGEH')
```



[25]:
```
#Si vous voulez les deux graphes en 1, il suffit de reprendre la structure de␣
 ↪matplotlib
#(notamment l'objet subplot) et de voir comment il peut etre appelé dans
#chaque méthode de tracé (df.plot de pandas et sns.plot de searborn)

from matplotlib import pyplot as plt

plt.style.use('seaborn-whitegrid')
fig = plt.figure(figsize(8.5,5))

ax1 = fig.add_subplot(1,2,1)
ax2 = fig.add_subplot(1,2,2)

ax1.scatter(df['AGEH'],df['AGEF'], color="#3333FF", edgecolors='#FFFFFF')
df.plot(x='AGEH',y='AGEF',kind='scatter',ax=ax2)

plt.xlabel('AGEH')
plt.ylabel('AGEH')
```

[25]:
```
Text(0,0.5,'AGEH')
```

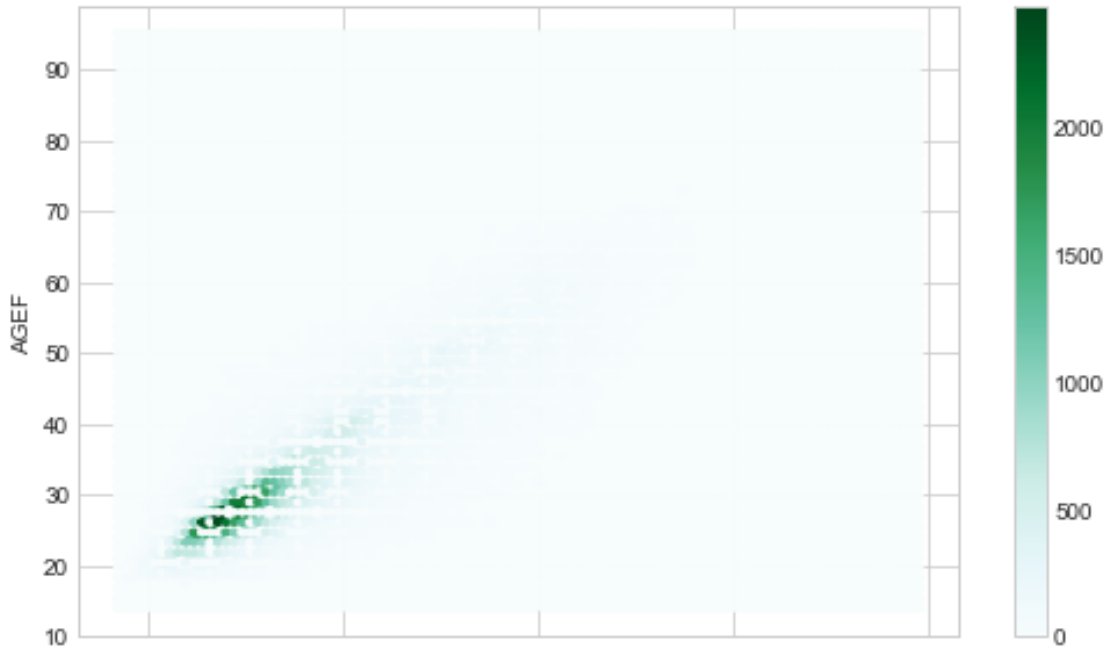


2.0.1 Exercice 1 : analyser l'âge des hommes en fonction de l'âge des femmes

Ajoutez un titre, changez le style du graphe, faites varier les couleurs (avec un camaïeu), faites une heatmap avec le wrapper pandas hexbin et avec seaborn.

[26]:
```
df.plot.hexbin(x='AGEH', y='AGEF', gridsize=100)
```

[26]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x221c6075550>
```

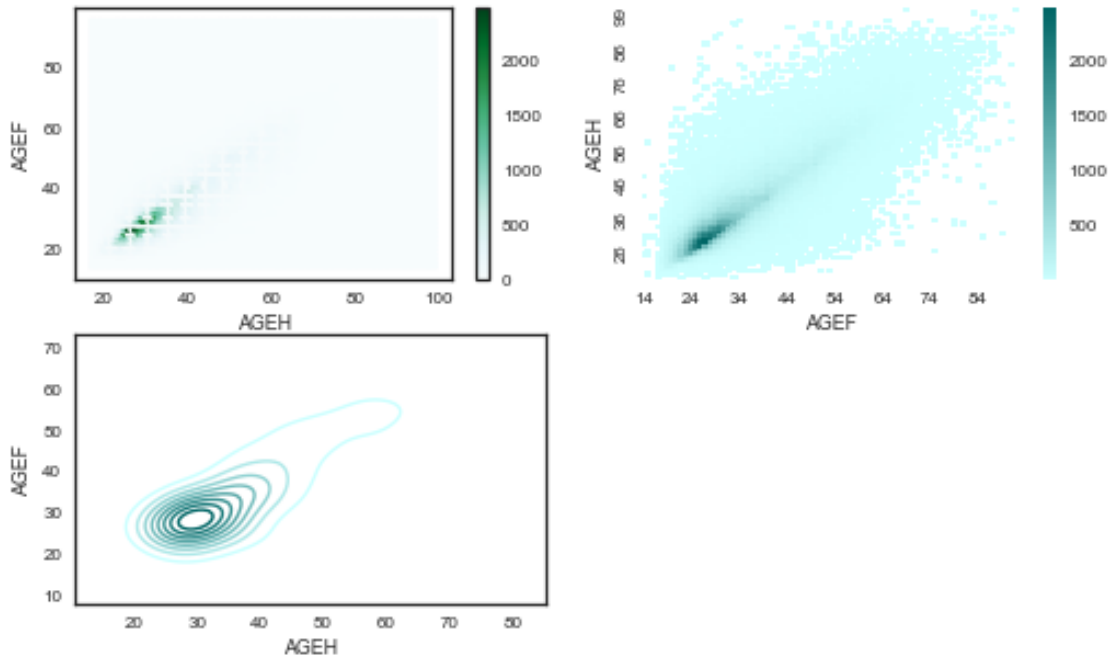


Avec seaborn

[27]:
```
import seaborn as sns

sns.set_style('white')
sns.set_context('paper')
#il faut créeer la matrice AGEH x AGEF
df['nb']=1
df[['AGEH','AGEF']]
df["nb"] = 1

#pour utiliser heatmap, il faut mettre df au frmat wide (au lieu de long) => df.pivot(.
 ↪..)
matrice = df[['nb','AGEH','AGEF']].groupby(['AGEH','AGEF'],as_index=False).count()
matrice=matrice.pivot('AGEH','AGEF','nb')
matrice=matrice.sort_index(axis=0,ascending=False)

fig = plt.figure(figsize(8.5,5))

ax1 = fig.add_subplot(2,2,1)
ax2 = fig.add_subplot(2,2,2)
ax3 = fig.add_subplot(2,2,3)

df.plot.hexbin(x='AGEH', y='AGEF', gridsize=100, ax=ax1)

cmap=sns.blend_palette(["#CCFFFF", "#006666"], as_cmap=True)
#Dans tous les graphes qui prévoient une fonction cmap vous pourrez intégrer votre␣
 ↪propre palette de couleur

sns.heatmap(matrice,annot=False, xticklabels=10,yticklabels=10,cmap=cmap,ax=ax2)
```

```
sample = df.sample(100)
sns.kdeplot(sample['AGEH'],sample['AGEF'],cmap=cmap,ax=ax3)
```

[27]: <matplotlib.axes._subplots.AxesSubplot at 0x221d1116fd0>



Seaborn est bien pensé pour les couleurs. Vous pouvez intégrer des palettes convergentes, divergentes. Essayez de faire un camaïeu entre deux couleurs au fur et à mesure de l'age, pour faire ressortir les contrastes.

2.0.2 Exercice 2 : représentez la répartition de la différence d'âge de couples mariés

[28]:
```
df["differenceHF"] = df["ANAISH"] - df["ANAISF"]
df["nb"] = 1
dist = df[["nb","differenceHF"]].groupby("differenceHF", as_index=False).count()
dist.tail()
```

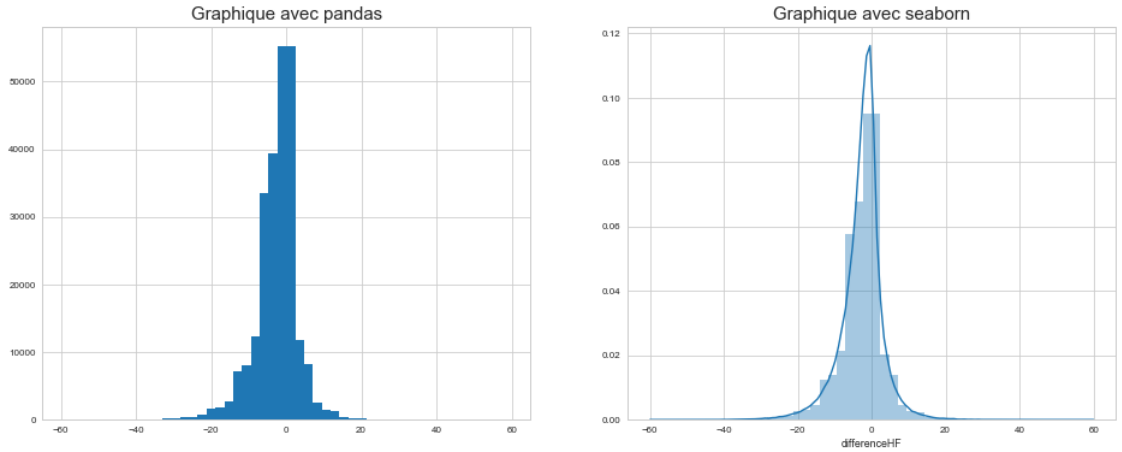
[28]:

	differenceHF	nb
95	46	1
96	48	3
97	50	1
98	56	1
99	59	1

[29]:
```
#version pandas
import seaborn as sns

sns.set_style('whitegrid')
sns.set_context('paper')
```

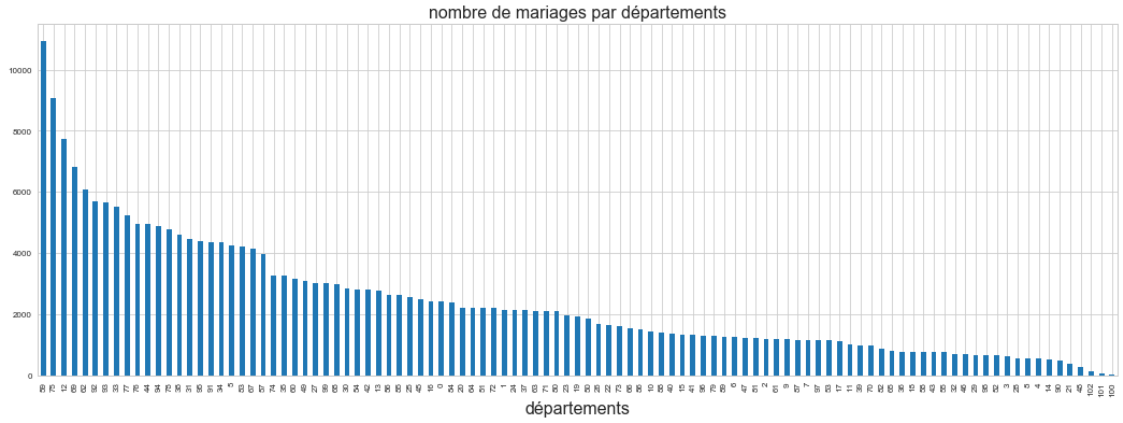
```
fig = plt.figure(figsize(8.5,5))

ax1 = fig.add_subplot(1,2,1)
ax2 = fig.add_subplot(1,2,2)

df["differenceHF"].hist(figsize=(16,6), bins=50, ax=ax1)
ax1.set_title('Graphique avec pandas', fontsize=15)

sns.distplot(df["differenceHF"], kde=True,ax=ax2)
#regardez ce que donne l'option kde
ax2.set_title('Graphique avec seaborn', fontsize=15)
```

[29]:
```
Text(0.5,1,'Graphique avec seaborn')
```



2.0.3 Exercice 3 : analyser le nombre de mariages par département

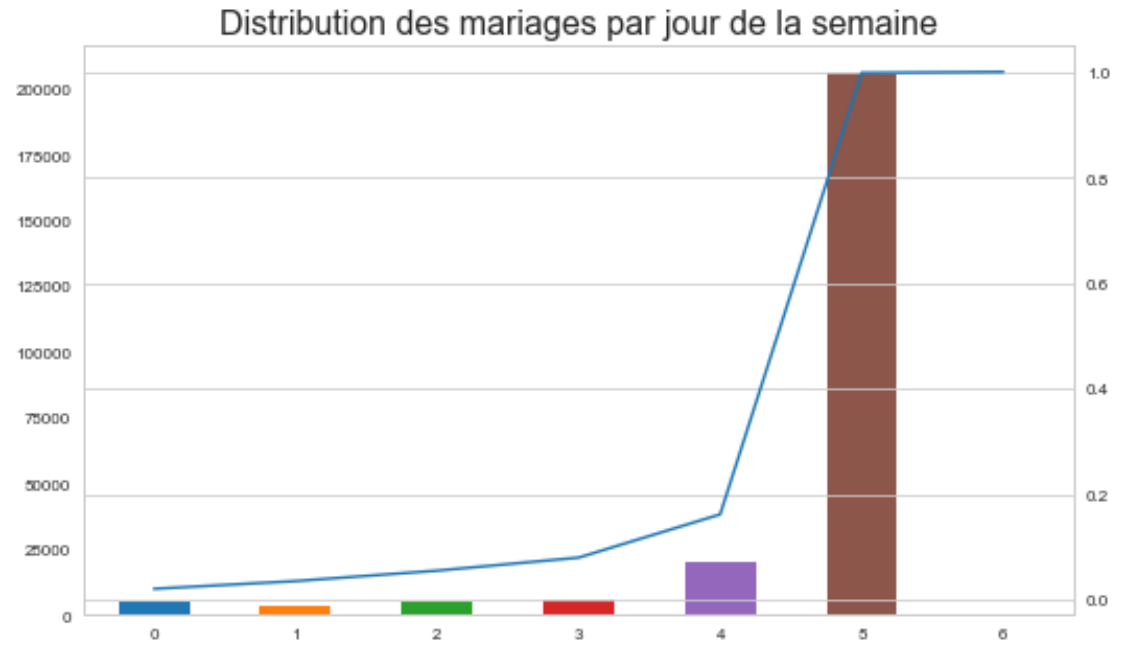
[30]:
```
df["nb"] = 1
dep = df[["DEPMAR","nb"]].groupby("DEPMAR", as_index=False).sum().
  ↪sort_values("nb",ascending=False)
ax = dep.plot(kind = "bar", figsize=(18,6))
ax.set_xlabel("départements", fontsize=16)
ax.set_title("nombre de mariages par départements", fontsize=16)
ax.legend().set_visible(False)  # on supprime la légende

# on change la taille de police de certains labels
for i,tick in enumerate(ax.xaxis.get_major_ticks()):
    if i > 10 :
        tick.label.set_fontsize(8)
```



3 Exercice 4 : répartition du nombre de mariages par jour

[31]:
```
df["nb"] = 1
dissem = df[["JSEMAINE","nb"]].groupby("JSEMAINE",as_index=False).sum()
total = dissem["nb"].sum()
repsem = dissem.cumsum()
repsem["nb"] /= total

sns.set_style('whitegrid')
ax = dissem["nb"].plot(kind="bar")
repsem["nb"].plot(ax=ax, secondary_y=True)
ax.set_title("Distribution des mariages par jour de la semaine",fontsize=16)
```

[31]:
```
Text(0.5,1,'Distribution des mariages par jour de la semaine')
```



[32]:
```
df.head()
```

[32]:

	ANAISH	DEPNAISH	INDNATH	ETAMATH	ANAISF	DEPNAISF	INDNATF	ETAMATF	AMAR	\
0	1982	75	1	1	1984	99	2	1	2012	
1	1956	69	2	4	1969	99	2	4	2012	
2	1982	99	2	1	1992	99	1	1	2012	
3	1985	99	2	1	1987	84	1	1	2012	
4	1968	99	2	1	1963	99	2	1	2012	

	MMAR	JSEMAINE	DEPMAR	DEPDOM	TUDOM	TUCOM	NBENFCOM	AGEF	AGEH	nb	\
0	01	1	29	99	9		N	28	30	1	
1	01	3	75	99	9		N	43	56	1	
2	01	5	34	99	9		N	20	30	1	
3	01	4	13	99	9		N	25	27	1	
4	01	6	26	99	9		N	49	44	1	

	differenceHF
0	-2
1	-13
2	-10
3	-2
4	5

4 Graphes intéractifs : bokeh, altair, bqplot

Pour faire simple, il est possible d'introduire du JavaScript dans l'application web locale créée par jupyter. C'est ce que fait D3.js. Les librairies interactives comme bokeh ou altair ont associé le design de matplotlib avec des librairies javascript comme vega-lite. L'exemple suivant utilise bokeh.

[33]:
```
from bokeh.plotting import figure, show, output_notebook
output_notebook()

fig = figure()

sample = df.sample(500)

fig.scatter(sample['AGEH'],sample['AGEF'])
fig.xaxis.axis_label = 'AGEH'
fig.yaxis.axis_label = 'AGEH'
show(fig)
```

La page callbacks montre comment utiliser les interactions utilisateurs. Seul inconvénient, il faut connaître le javascript.

[34]:
```
from bokeh.plotting import figure, output_file, show
from bokeh.models import ColumnDataSource, HoverTool, CustomJS


# define some points and a little graph between them
x = [2, 3, 5, 6, 8, 7]
y = [6, 4, 3, 8, 7, 5]
links = {
```

```
    0: [1, 2],
    1: [0, 3, 4],
    2: [0, 5],
    3: [1, 4],
    4: [1, 3],
    5: [2, 3, 4]
}

p = figure(plot_width=400, plot_height=400, tools="", toolbar_location=None,␣
 ↪title='Hover over points')

source = ColumnDataSource({'x0': [], 'y0': [], 'x1': [], 'y1': []})
sr = p.segment(x0='x0', y0='y0', x1='x1', y1='y1', color='olive', alpha=0.6,␣
 ↪line_width=3, source=source, )
cr = p.circle(x, y, color='olive', size=30, alpha=0.4, hover_color='olive',␣
 ↪hover_alpha=1.0)

# Add a hover tool, that sets the link data for a hovered circle
code = """
var links = %s;
var data = {'x0': [], 'y0': [], 'x1': [], 'y1': []};
var cdata = circle.data;
var indices = cb_data.index['1d'].indices;
for (i=0; i < indices.length; i++) {
    ind0 = indices[i]
    for (j=0; j < links[ind0].length; j++) {
        ind1 = links[ind0][j];
        data['x0'].push(cdata.x[ind0]);
        data['y0'].push(cdata.y[ind0]);
        data['x1'].push(cdata.x[ind1]);
        data['y1'].push(cdata.y[ind1]);
    }
}
segment.data = data;
""" % links

callback = CustomJS(args={'circle': cr.data_source, 'segment': sr.data_source},␣
 ↪code=code)
p.add_tools(HoverTool(tooltips=None, callback=callback, renderers=[cr]))

show(p)
```

Le module bqplot permet de définir des *callbacks* en Python. L'inconvénient est que cela ne fonction que depuis un notebook et il vaut mieux ne pas trop mélanger les librairies javascripts qui ne peuvent pas toujours fonctionner ensemble.

5 Plotly

- Plotly: https://plot.ly/python/
- Doc: https://plot.ly/python/reference/
- Colors: http://www.cssportal.com/css3-rgba-generator/

[35]:
```
import pandas as pd
import numpy as np
```

6 Creation dataframe

[36]:
```
indx = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
value1 = [0,1,2,3,4,5,6,7,8,9]
value2 = [1,5,2,3,7,5,1,8,9,1]

df = {'indx': indx, 'value1': value1, 'value2': value2}
df = pd.DataFrame(df)
df['rate1'] = df.value1 / 100
df['rate2'] = df.value2 / 100
df = df.set_index('indx')
df.head()
```

[36]:

indx	value1	value2	rate1	rate2
a	0	1	0.00	0.01
b	1	5	0.01	0.05
c	2	2	0.02	0.02
d	3	3	0.03	0.03
e	4	7	0.04	0.07

7 Bars and Scatter

[37]:
```
# installer plotly
```

[38]:
```
import plotly.plotly as py
import os
from pyquickhelper.loghelper import get_password
user = get_password("plotly", "ensae_teaching_cs,login")
pwd = get_password("plotly", "ensae_teaching_cs,pwd")
try:
    py.sign_in(user, pwd)
except Exception as e:
    print(e)
```

```
Sign in failed.
```

[39]:
```
import plotly
from plotly.graph_objs import Bar, Scatter, Figure, Layout
import plotly.plotly as py
import plotly.graph_objs as go

# BARS

trace1 = go.Bar(
    x = df.index,
    y = df.value1,
```

```
    name='Value1',                          # Bar legend
    #orientation = 'h',
    marker = dict(                          # Colors
        color = 'rgba(237, 74, 51, 0.6)',
        line = dict(
            color = 'rgba(237, 74, 51, 0.6)',
            width = 3)
    ))

trace2 = go.Bar(
    x = df.index,
    y = df.value2,
    name='Value 2',
    #orientation = 'h',                      # Uncomment to have horizontal bars
    marker = dict(
        color = 'rgba(0, 74, 240, 0.4)',
        line = dict(
            color = 'rgba(0, 74, 240, 0.4)',
            width = 3)
    ))

# SCATTER

trace3 = go.Scatter(
    x = df.index,
    y = df.rate1,
    name='Rate',
    yaxis='y2',                             # Define 2 axis
    marker = dict(                          # Colors
        color = 'rgba(187, 0, 0, 1)',
    ))

trace4 = go.Scatter(
    x = df.index,
    y = df.rate2,
    name='Rate2',
    yaxis='y2',                             # To have a 2nd axis
    marker = dict(                          # Colors
        color = 'rgba(0, 74, 240, 0.4)',
    ))

data = [trace2, trace1, trace3, trace4]

layout = go.Layout(
    title='Stack bars and scatter',
    barmode ='stack',                          # Take value 'stack' or 'group'
    xaxis=dict(
        autorange=True,
        showgrid=False,
        zeroline=False,
        showline=True,
        autotick=True,
        ticks='',
```

```
        showticklabels=True
    ),
    yaxis=dict(                                    # Params 1st axis
       #range=[0,1200000],                         # Set range
        autorange=True,
        showgrid=False,
        zeroline=False,
        showline=True,
        autotick=True,
        ticks='',
        showticklabels=True
    ),
    yaxis2=dict(                                   # Params 2nd axis
        overlaying='y',
        autorange=True,
        showgrid=False,
        zeroline=False,
        showline=True,
        autotick=True,
        ticks='',
        side='right'
    ))

fig = go.Figure(data=data, layout=layout)
py.iplot(fig, filename='marker-h-bar')
```

```
High five! You successfully sent some data to your account on plotly. View your
plot in your browser at https://plot.ly/~athean/0 or inside your plot.ly account
where it is named 'marker-h-bar'
```

[39]:
```
<plotly.tools.PlotlyDisplay object>
```

[40]:
```
trace5 = go.Scatter(
    x = ['h', 'h'],
    y = [0,0.09],
    yaxis='y2',                                # Define 2 axis
    showlegend = False,                        # Hiding legend for this trace
    marker = dict(                             # Colors
        color = 'rgba(46, 138, 24, 1)',
    )
)

from plotly import tools
import plotly.plotly as py
import plotly.graph_objs as go


fig = tools.make_subplots(rows=1, cols=2)

# 1st subplot
fig.append_trace(trace1, 1, 1)
fig.append_trace(trace2, 1, 1)
```

```
# 2nd subplot
fig.append_trace(trace3, 1, 2)
fig.append_trace(trace4, 1, 2)
fig.append_trace(trace5, 1, 2)              # Vertical line here


fig['layout'].update(height=600, width=1000, title='Two in One  & Vertical line')
py.iplot(fig, filename='make-subplots')
```

```
This is the format of your plot grid:
[ (1,1) x1,y1 ]  [ (1,2) x2,y2 ]
```

[40]:
```
<plotly.tools.PlotlyDisplay object>
```

7.0.1 Exercice : représenter le nombre de mariages par département avec plotly ou tout autre librairie javascript

Bokeh, altair, ...

[41]:

[42]: