Découvrez les librairies Python pour la Data Science

10 heures Moyenne

Mis à jour le 07/06/2022

Maîtrisez les possibilités offertes par Matplotlib

08:13

Matplotlib a vu le jour pour permettre de générer directement des graphiques à partir de Python. Au fil des années, Matplotlib est devenu une librairie puissante, compatible avec beaucoup de plateformes, et capable de générer des graphiques dans beaucoup de formats différents.

Dans ce chapitre, nous allons nous concentrer sur l'utilisation de Matplotlib comme outil de visualisation dans les notebooks Jupyter.

Pour commencer, mettons en place l'environnement de travail.

python

```python
%matplotlib inline
import matplotlib.pyplot as plt
```

```
plt.style.use('seaborn-whitegrid')
import numpy as np
```

Réaliser des graphiques simples

Commençons par étudier un cas simple, comme tracer la courbe d'une fonction. Nous avons vu un exemple de cette utilisation dans le chapitre 3 de la première partie de ce cours. Ici, nous allons le faire d'une manière moins simple, mais qui nous donne plus de possibilités.

python

```python
fig = plt.figure()
ax = plt.axes()
```

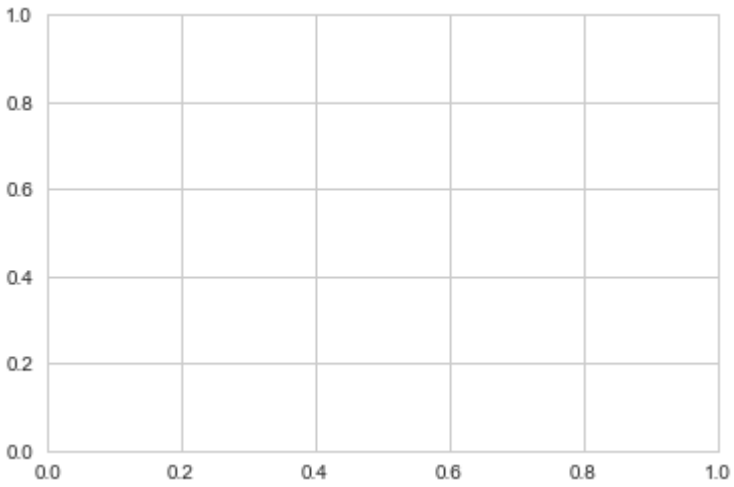
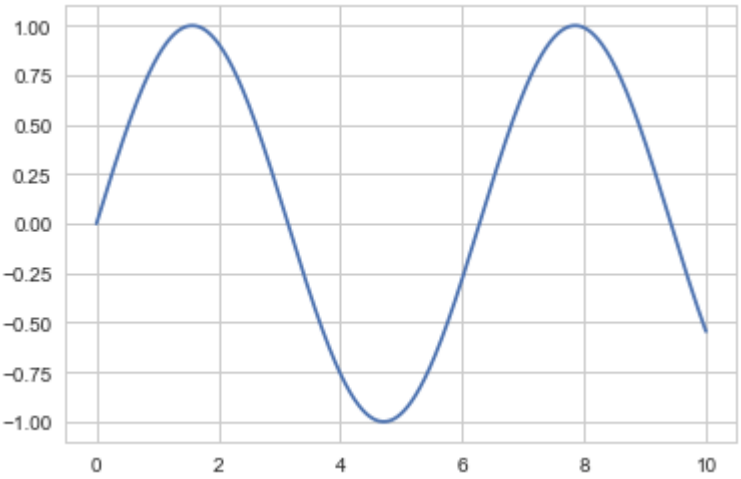


Figure vide

La variable `fig` correspond à un conteneur qui contient tous les objets (axes, labels, données, etc). Les axes correspondent au carré que l'on voit au-dessus, et qui contiendra par la suite les données du graphe.

python

```python
fig = plt.figure()
ax = plt.axes()
x = np.linspace(0, 10, 1000)
ax.plot(x, np.sin(x));
```



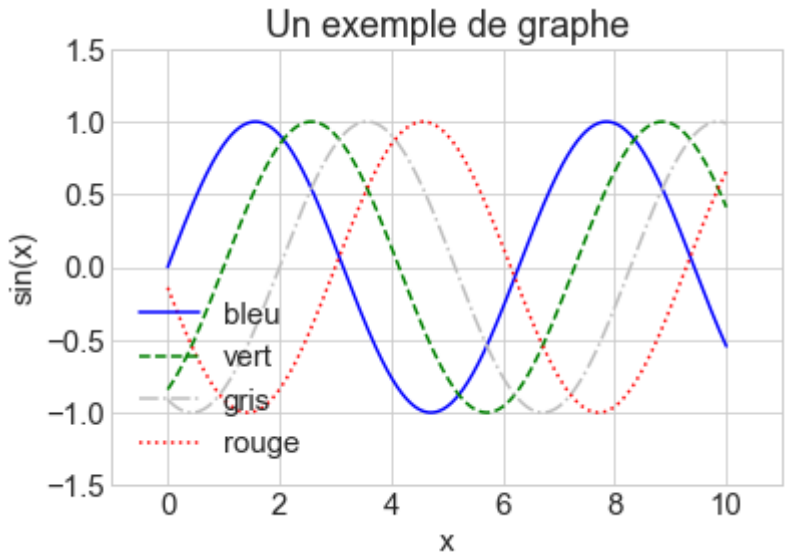
Courbe de sinus

On aurait pu simplement taper `plt.plot(x, np.sin(x))` .

Maintenant, voyons un exemple un peu plus poussé. Pour cet exemple, je vous invite à lire le code et les commentaires, et à observer le résultat sur le graphique généré.

python

```python
# Chanegr la taille de police par défaut
plt.rcParams.update({'font.size': 15})

fig = plt.figure()
ax = plt.axes()
# Couleur spécifiée par son nom, ligne solide
plt.plot(x, np.sin(x - 0), color='blue', linestyle='solid', label='bleu')
# Nom court pour la couleur, ligne avec des traits
plt.plot(x, np.sin(x - 1), color='g', linestyle='dashed', label='vert')
# Valeur de gris entre 0 et 1, des traits et des points
plt.plot(x, np.sin(x - 2), color='0.75', linestyle='dashdot', label='gris')
# Couleur spécifié en RGB, avec des points
plt.plot(x, np.sin(x - 3), color='#FF0000', linestyle='dotted', label='rouge')

# Les limites des axes, essayez aussi les arguments 'tight' et 'equal'
# pour voir leur effet
plt.axis([-1, 11, -1.5, 1.5]);

# Les labels
plt.title("Un exemple de graphe")

# La légende est générée à partir de l'argument label de la fonctio
# plot. L'argument loc spécifie le placement de la légende
plt.legend(loc='lower left');

# Titres des axes
ax = ax.set(xlabel='x', ylabel='sin(x)')
```



Un graphique plus complexe

Visualiser l'incertitude

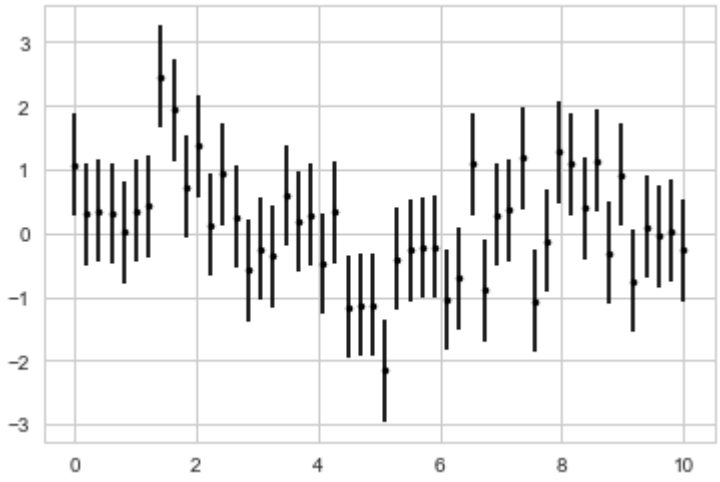
Dans la vie réelle, les données que nous sommes amenés à analyser sont souvent bruitées, c'est-à-dire qu'il existe une part d'incertitude sur leur valeur réelle. Il est extrêmement important d'en tenir compte non seulement lors de l'analyse des données, mais aussi quand on veut les présenter.

Données discrètes

Dans le cas de données discrètes (des points), nous utilisons souvent les barres d'erreur pour représenter, pour chaque point, l'incertitude quant à sa valeur exacte. Souvent la longueur des barres correspond à l'écart type des observations empiriques. C'est chose aisée avec Matplotlib.

python

```python
x = np.linspace(0, 10, 50)
dy = 0.8
y = np.sin(x) + dy * np.random.randn(50)

plt.errorbar(x, y, yerr=dy, fmt='.k');
```

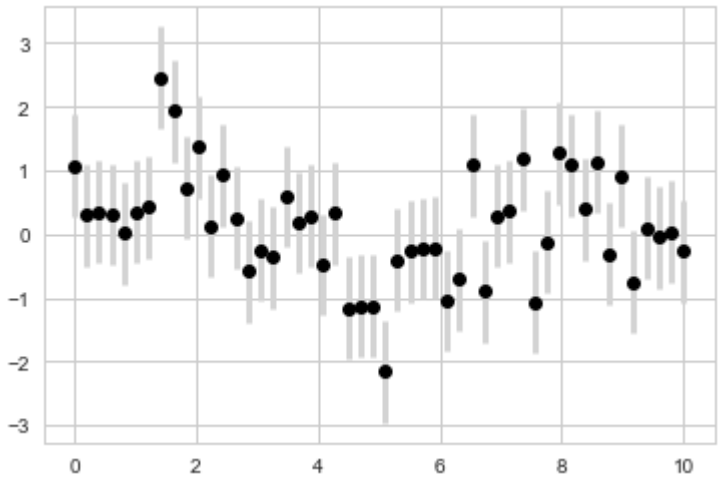


Les barres d'erreur

`Errorbar` prend en argument les abscisses `x`, les coordonnées `y` et les longueurs de chaque barre (une barre par point) `yerr`. Notez l'argument `fmt`. Il permet de choisir, de façon courte, la couleur (ici noir ou blac**k**) et la forme des marqueurs du graphe. `Errorbar` permet aussi de personnaliser encore plus l'apparence du graphe. Je vous invite fortement à consulter sa documentation.

python

```python
plt.errorbar(x, y, yerr=dy, fmt='o', color='black',
             ecolor='lightgray', elinewidth=3, capsize=0);
```



Des barres d'erreur d'aspect différent

Données continues

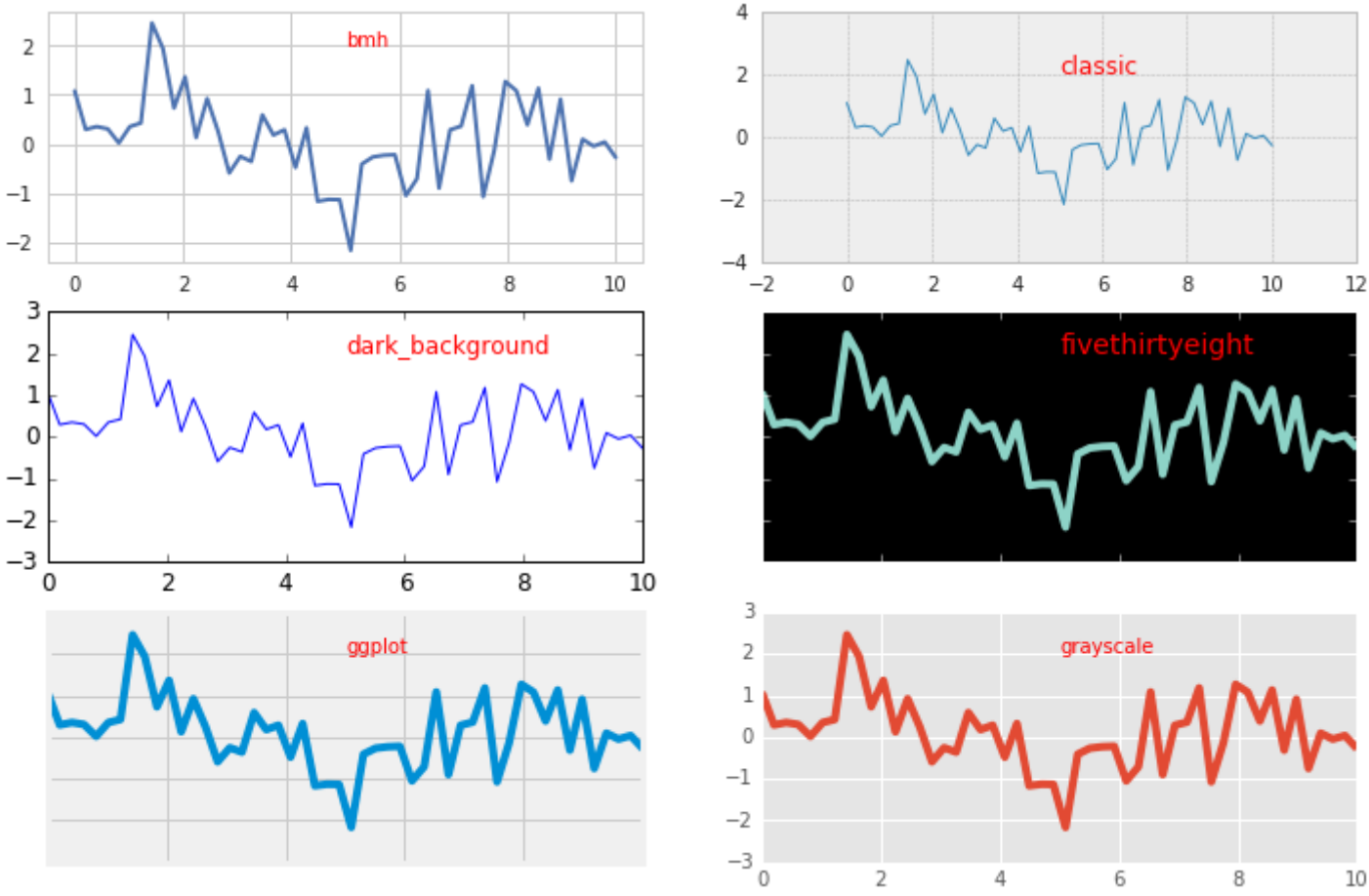
Parfois, comme quand on essaie d'appliquer la régression par processus gaussien, nous avons besoin de représenter une incertitude sur une fonction continue. On peut faire ceci en utilisant la fonction `plot` conjointement avec la fonction `fill_between`. Mais nous allons voir plus tard dans ce chapitre comment le faire plus simplement avec la librairie Seaborn.

Personnalisation et sous-graphes

Matplotlib est très flexible. Quasiment tous les aspects d'une figure peuvent être configurés par l'utilisateur soit pour y ajouter des données, soit pour améliorer l'aspect esthétique. Plutôt que de vous faire une liste des fonctions qui permettent de faire ces actions, j'ai plutôt décidé de vous montrer des exemples. A l'avenir, n'hésitez pas à revenir vers cette partie pour vous remémorer comment réaliser une opération spécifique.

python

```python
print(plt.style.available[:6])

# Notez la taille de la figure
fig = plt.figure(figsize=(12,8))
for i in range(6):
    # On peut ajouter des sous graphes ainsi
    fig.add_subplot(3,2,i+1)
    plt.style.use(plt.style.available[i])
    plt.plot(x, y)

    # Pour ajouter du texte
    plt.text(s=plt.style.available[i], x=5, y=2, color='red')
```



Exemples de stylesheet

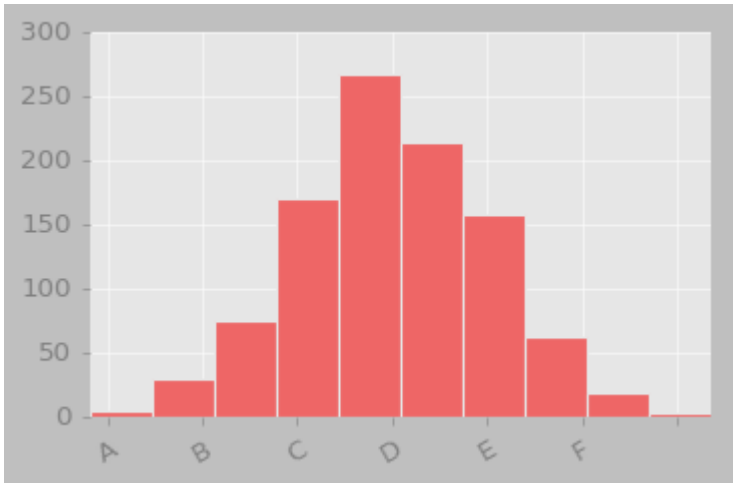
Le premier argument de la fonction add_subplot est le nombre de lignes de notre tableau de graphes (ici 3). Le deuxième est le nombre de colonnes (ici 2). Le troisième est le numéro du graphe, parmi les graphes de ce tableau, que nous voulons dessiner.

Pour des raisons historiques, les sous-graphes sont numérotés à partir de 1, au lieu de 0. Le graphe tout en haut à gauche est donc le graphe numéro 1.

Nous pouvons aussi tout personnaliser à la main.

python

```python
# On peut aussi tout personnaliser à la main
x = np.random.randn(1000)

plt.style.use('classic')
fig=plt.figure(figsize=(5,3))
ax = plt.axes(facecolor='#E6E6E6')

# Afficher les ticks en dessous de l'axe
ax.set_axisbelow(True)

# Cadre en blanc
plt.grid(color='w', linestyle='solid')

# Cacher le cadre
# ax.spines contient les lignes qui entourent la zone où les
# données sont affichées.
for spine in ax.spines.values():
    spine.set_visible(False)

# Cacher les marqueurs en haut et à droite
ax.xaxis.tick_bottom()
ax.yaxis.tick_left()

# Nous pouvons personnaliser les étiquettes des marqueurs
# et leur appliquer une rotation
marqueurs = [-3, -2, -1, 0, 1, 2, 3]
xtick_labels = ['A', 'B', 'C', 'D', 'E', 'F']
plt.xticks(marqueurs, xtick_labels, rotation=30)

# Changer les couleur des marqueurs
ax.tick_params(colors='gray', direction='out')
for tick in ax.get_xticklabels():
    tick.set_color('gray')
for tick in ax.get_yticklabels():
    tick.set_color('gray')

# Changer les couleur des barres
ax.hist(x, edgecolor='#E6E6E6', color='#EE6666');
```



Exemple de graphique personnalisé

Pour aller plus loin

- Je vous recommande ce notebook, qui vous servira de référence pour des bouts de code simples et plus avancées de Matplotlib.

J'ai terminé ce chapitre et je passe au suivant

Plongez en détail dans la librairie NumPy

Réalisez de beaux graphiques avec Seaborn

Le professeur

Ali Neishabouri

Freelance Data Scientist, and teacher at OpenClassrooms


Télécharger dans
l'App Store