Introduction à Python et au Machine Learning

Cours 3 - la librairie Matplotlib pour la visualisation des données

Olivier Goudet

LERIA, Université d'Angers

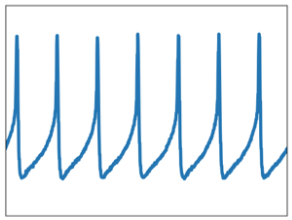
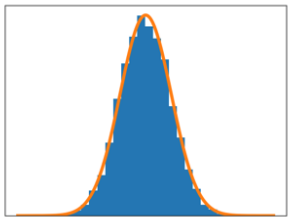
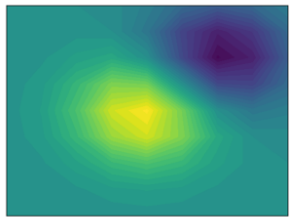
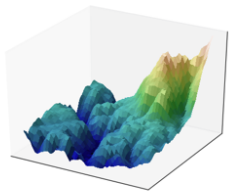
11 février 2025

Section 1

Introduction

La librairie Matplotlib

- Matplotlib est une librairie Python qui permet de visualiser des données.
- Il est possible de réaliser un grand nombre de types d'affichages différents : courbes 2D, courbes 3D, histogrammes, nuages de points, etc...
- Le site de la librairie est ici : `https://matplotlib.org/`.

Utilisation de Matplotlib

- Installation de la librairie :
  - Avec pip : pip install matplotlib
  - Avec Anaconda : conda install matplotlib
- Importation du module pyplot de la librairie Matplotlib et raccourci classiquement utilisé :

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

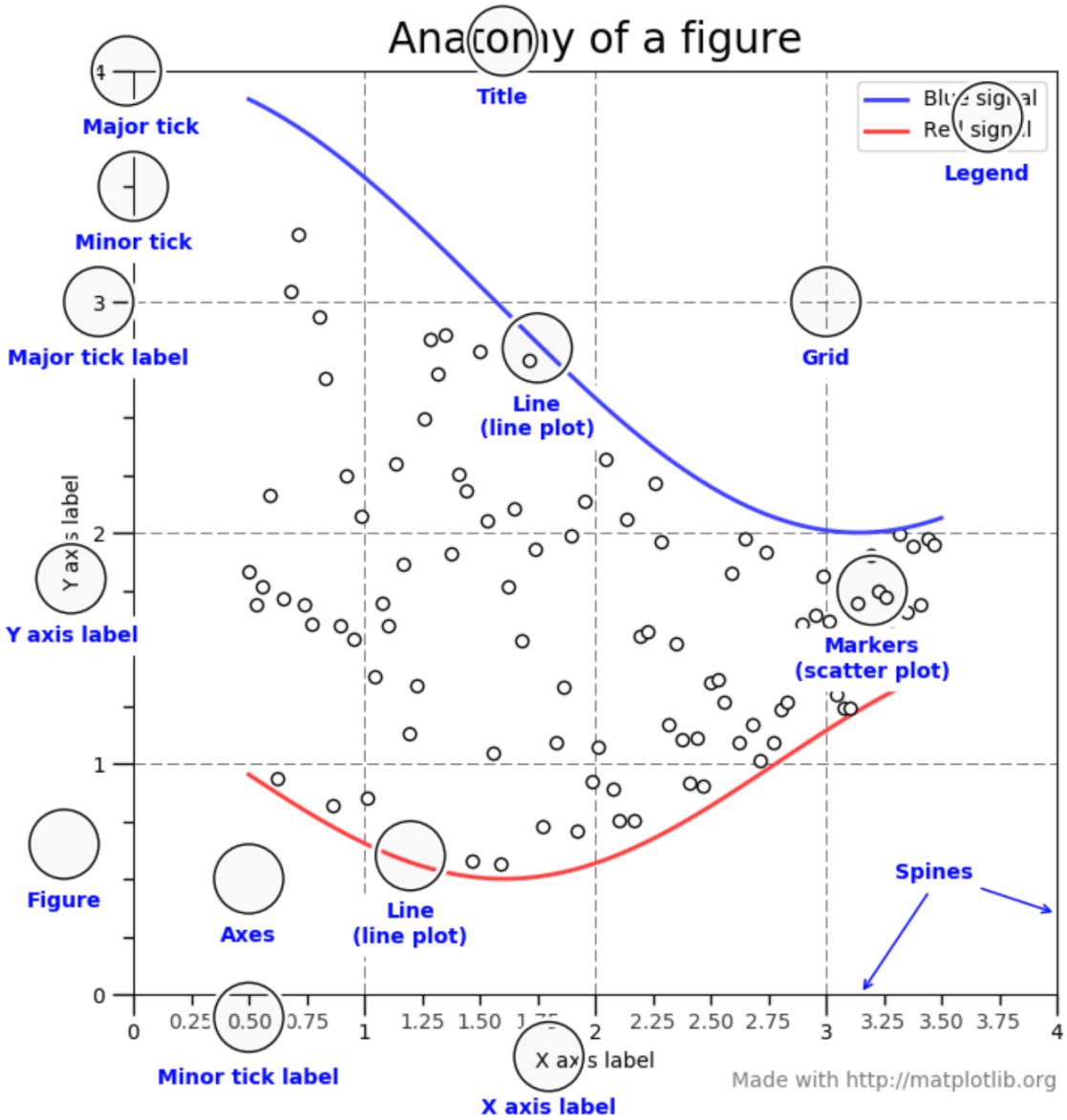
- Vérifier la version installée :

```
print(matplotlib.__version__)
```

Section 2

Concepts généraux

Anatomie d'une figure Matplotlib



Exemple de base

- Les données fournies à Matplotlib peuvent être des listes, des numpy arrays ou bien des colonnes de Dataframe.
- Création de vecteurs de valeurs avec Numpy :

```
import numpy as np
x = np.linspace(0, 5, 11)
y = x ** 2
print(x)
print(y)
```

Affiche :

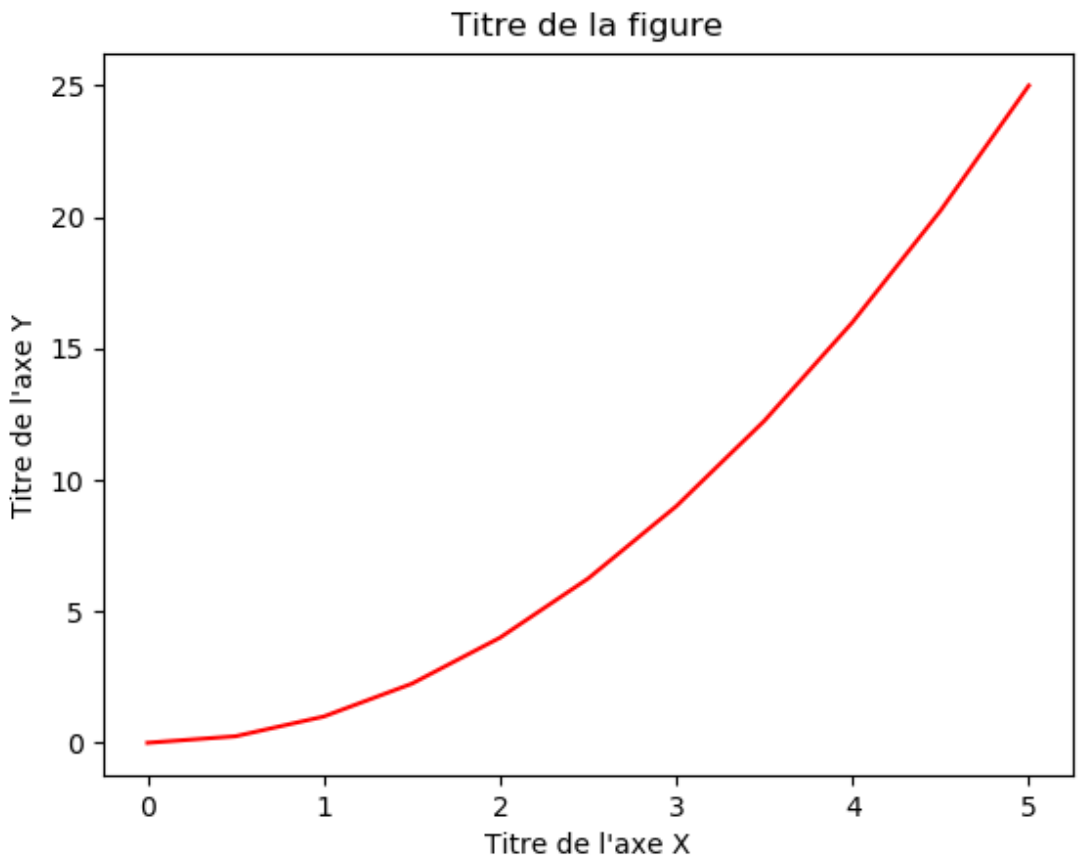
```
array([ 0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5, 5. ])
array([ 0. , 0.25, 1. , 2.25, 4. , 6.25, 9. , 12.25, 16. , 20.25, 25. ])
```

Exemple de base

- On peut ensuite créer une courbe avec les données de x et y de la façon suivante :

```
import matplotlib.pyplot as plt

plt.plot(x, y, "r") # "r" est la couleur rouge
plt.xlabel("Titre de l'axe X")
plt.ylabel("Titre de l'axe Y")
plt.title("Titre de la figure")
plt.show()
```

Figure obtenue

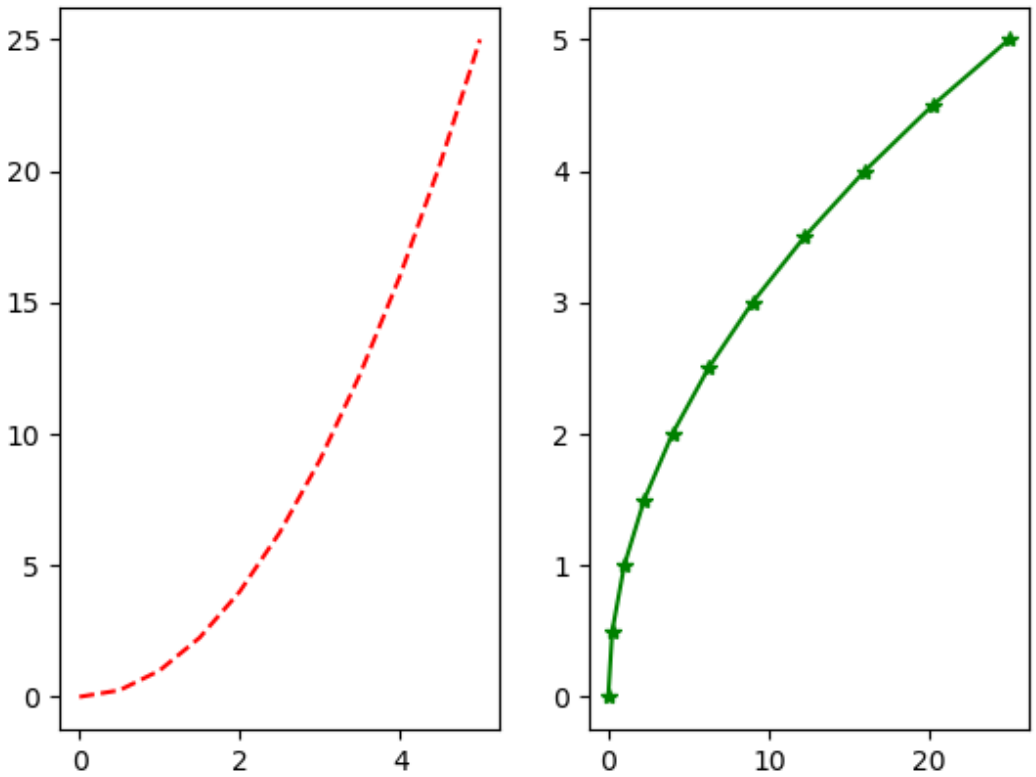


Création de figures multiples sur le même canevas

- On peut créer plusieurs sous-graphiques dans le même affichage :

```
# plt.subplot(nRows, nCols, number_plot)
plt.subplot(1,2,1)
plt.plot(x, y, "r--")
plt.subplot(1,2,2)
plt.plot(y, x, "g*-");

plt.show()
```

Figure obtenue



Section 3

Approche orientée objet avec Matplotlib

Approche orientée objet avec Matplotlib

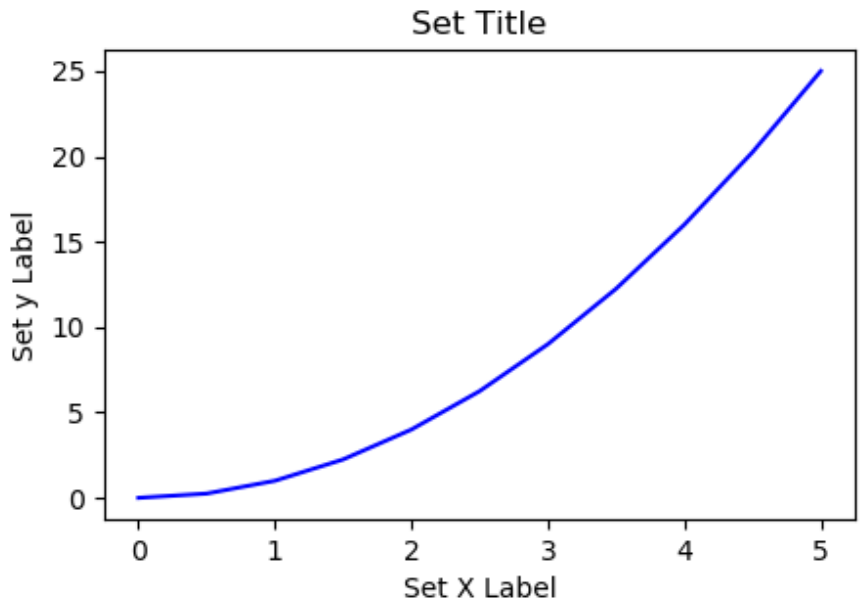
- Avec Matplotlib, on peut instancier des objets de type "figure" et appeler des méthodes dessus ou bien modifier leurs attributs.
- Cette approche orientée objet n'est pas obligatoire avec Matplotlib mais elle est très pratique pour faire des affichages plus évolués qui nécessitent notamment de créer plusieurs figures au sein d'un même canevas.

Introduction - approche orientée objet avec Matplotlib

- Création d'une instance de figure, et ajout d'un système d'axe sur cette figure.

```
# Create Figure (empty canvas)
fig = plt.figure()

# Add set of axes to figure
axes = fig.add_axes([0.3, 0.1, 0.7, 0.5]) # left, bottom, width, height (range 0 to 1)

# Plot on that set of axes
axes.plot(x, y, "b")
axes.set_xlabel("Set X Label")
axes.set_ylabel("Set y Label")
axes.set_title("Set Title")

plt.show()
```

Figure obtenue

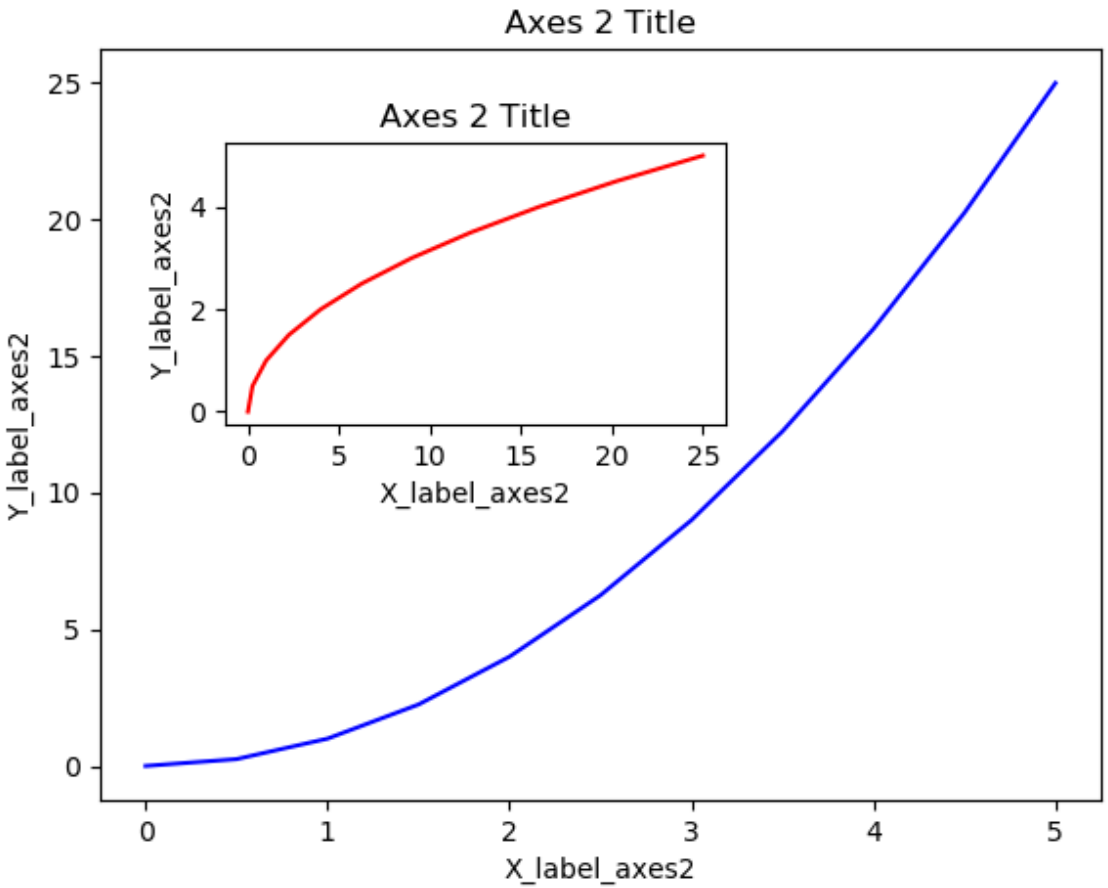


Introduction - approche orientée objet avec Matplotlib

- Ajouts de différents systèmes d'axes au sein d'un même canevas :

```
# Creates blank canvas
fig = plt.figure()

axes1 = fig.add_axes([0.1, 0.1, 0.8, 0.8]) # main axes
axes2 = fig.add_axes([0.2, 0.5, 0.4, 0.3]) # inset axes

# Larger Figure Axes 1
axes1.plot(x, y, "b")
axes1.set_xlabel("X_label_axes2")
axes1.set_ylabel("Y_label_axes2")
axes1.set_title("Axes 2 Title")

# Insert Figure Axes 2
axes2.plot(y, x, "r")
axes2.set_xlabel("X_label_axes2")
axes2.set_ylabel("Y_label_axes2")
axes2.set_title("Axes 2 Title");
```

Figure obtenue

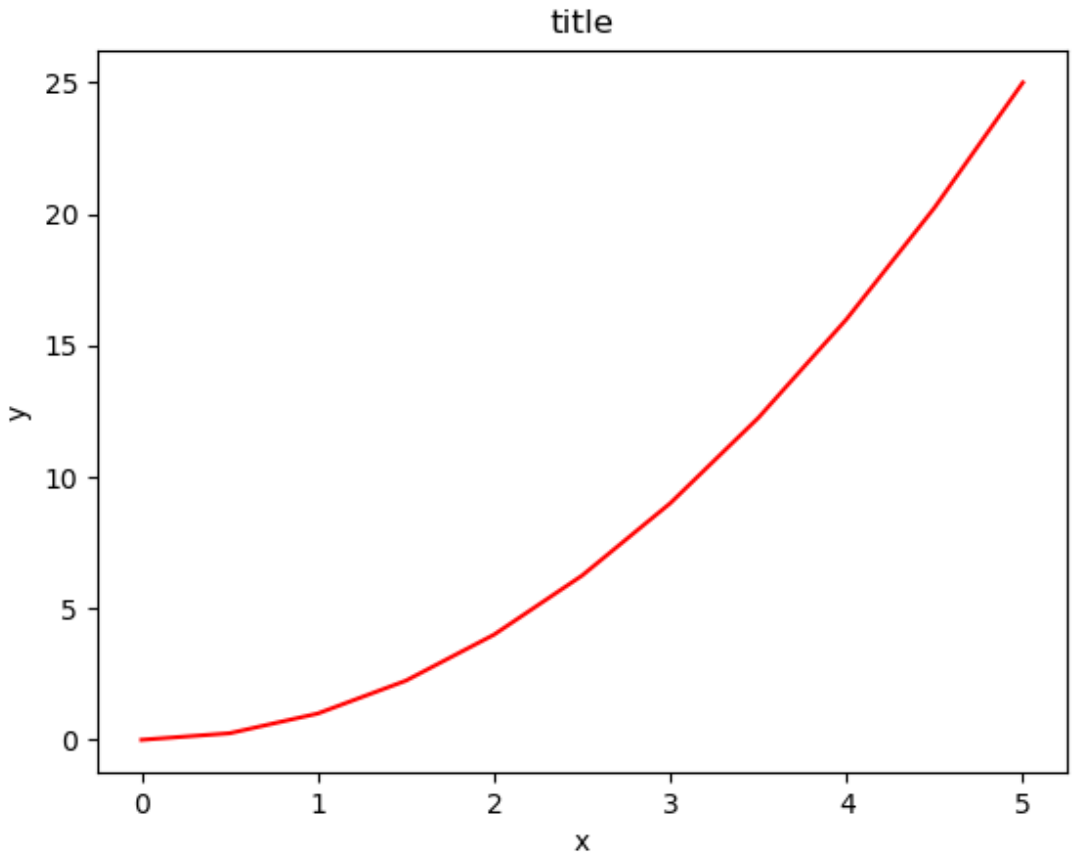


Gestion automatique des axes avec la méthode *subplots()*

- Avec l'appel à *plt.subplots()*, on récupère un tuple qui contient directement la figure ainsi qu'un système d'axes par défaut qu'on peut ensuite utiliser pour tracer des éléments graphiques :

```
fig, axes = plt.subplots()

axes.plot(x, y, "r")
axes.set_xlabel("x")
axes.set_ylabel("y")
axes.set_title("title");

plt.show()
```

Figure obtenue

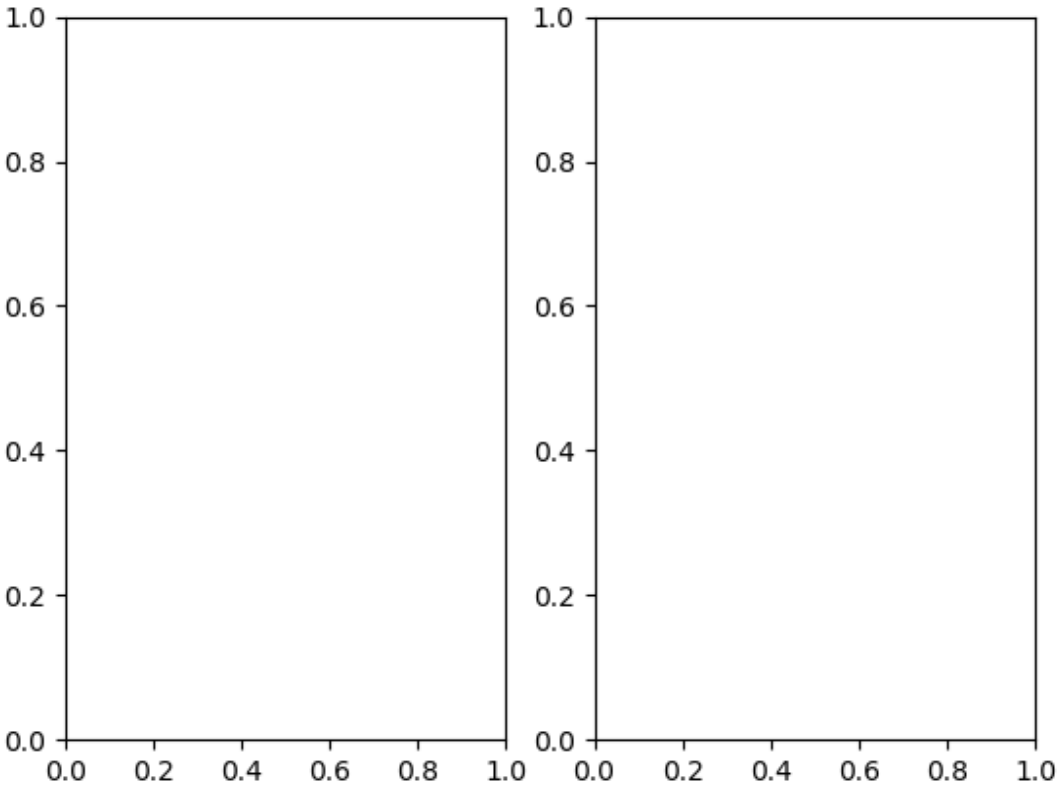


Gestion automatiques des axes avec la méthode *subplots()*

- L'intérêt de *plt.subplots()* va être de pouvoir créer automatiquement deux sous-figures avec leurs systèmes d'axes qu'on peut ensuite manipuler. Par exemple ici avec deux sous-graphiques placés sur deux colonnes :

```
fig, axes = plt.subplots(nrows=1, ncols=2)

plt.show()
```

Figure obtenue

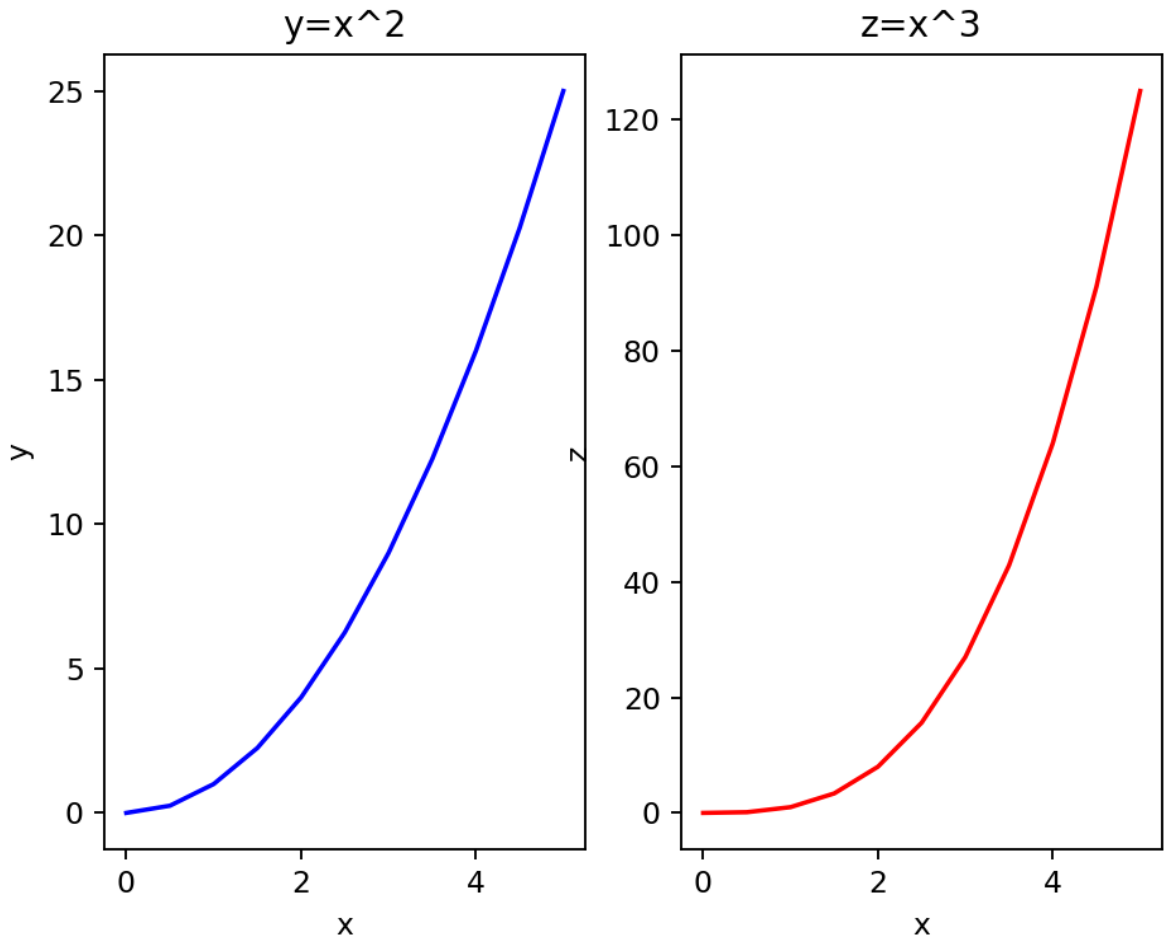


Gestion automatiques des axes avec la méthode *subplots()*

- *axes* correspond ici à un tableau de systèmes d'axes qu'on peut ensuite parcourir de façon à réaliser différents affichages :

```
x = np.linspace(0, 5, 11)
y = x ** 2
z = x ** 3

fig, axes = plt.subplots(nrows=1, ncols=2)

axes[0].plot(x, y, "b")
axes[0].set_xlabel("x")
axes[0].set_ylabel("y")
axes[0].set_title("y=x^2")

axes[1].plot(x, z, "r")
axes[1].set_xlabel("x")
axes[1].set_ylabel("z")
axes[1].set_title("z=x^3")

plt.show()
```

Figure obtenue

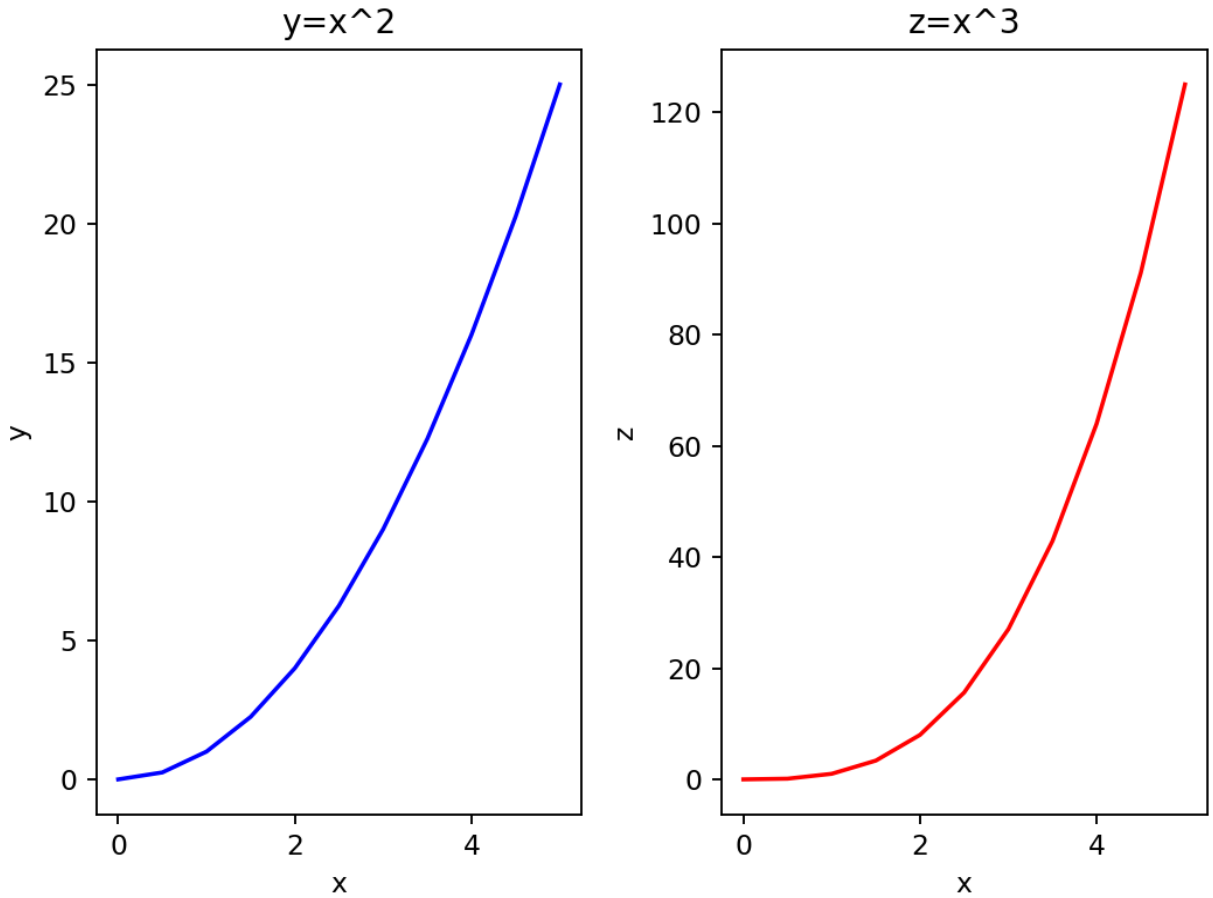


Gestion automatiques des axes avec la méthode *subplots()*

- Un problème courant avec matplotlib est la superposition des sous-figures. On peut les ajuster automatiquement avec la méthode *tight_layout()* :

```
fig, axes = plt.subplots(nrows=1, ncols=2)

axes[0].plot(x, y, "b")
axes[0].set_xlabel("x")
axes[0].set_ylabel("y")
axes[0].set_title("y=x^2")

axes[1].plot(x, z, "r")
axes[1].set_xlabel("x")
axes[1].set_ylabel("z")
axes[1].set_title("z=x^3")

plt.tight_layout()
plt.show()
```

Figure obtenue

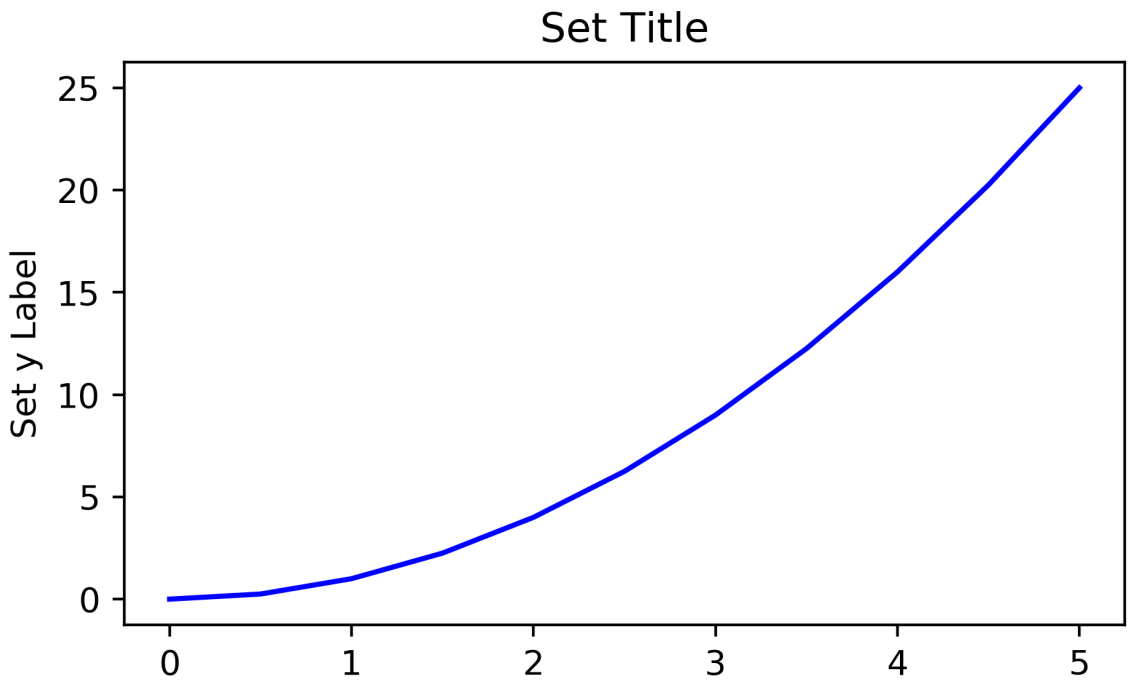


Gestion de la taille de la figure et de la qualité de l'image

- Avec Matplotlib, on peut gérer la taille de l'image ainsi que la qualité de l'image avec deux arguments :
  - figsize est un tuple qui précise la largeur et la hauteur de l'image en pouces ("inches").
  - dpi (*dots-per-inch*) est le nombre de pixels par pouce.
- Exemple :

```
fig = plt.figure(figsize=(8,4), dpi=400)

axes = fig.add_axes([0.1, 0.1, 0.8, 0.8])

axes.plot(x, y, "b")
axes.set_xlabel("Set X Label")
axes.set_ylabel("Set y Label")
axes.set_title("Set Title")

plt.show()
```

Figure obtenue



Sauvegarde de l'image sur le disque

- Au lieu d'afficher l'image avec la fonction *show()*, on peut effectuer une sauvegarde sur le disque avec la méthode *savefig()*.
- Différents formats sont disponibles comme PNG, JPG, PDF, etc...
- On doit préciser le nom du fichier de sauvegarde mais on peut aussi préciser la qualité de l'image au moment de la sauvegarde. Par exemple :

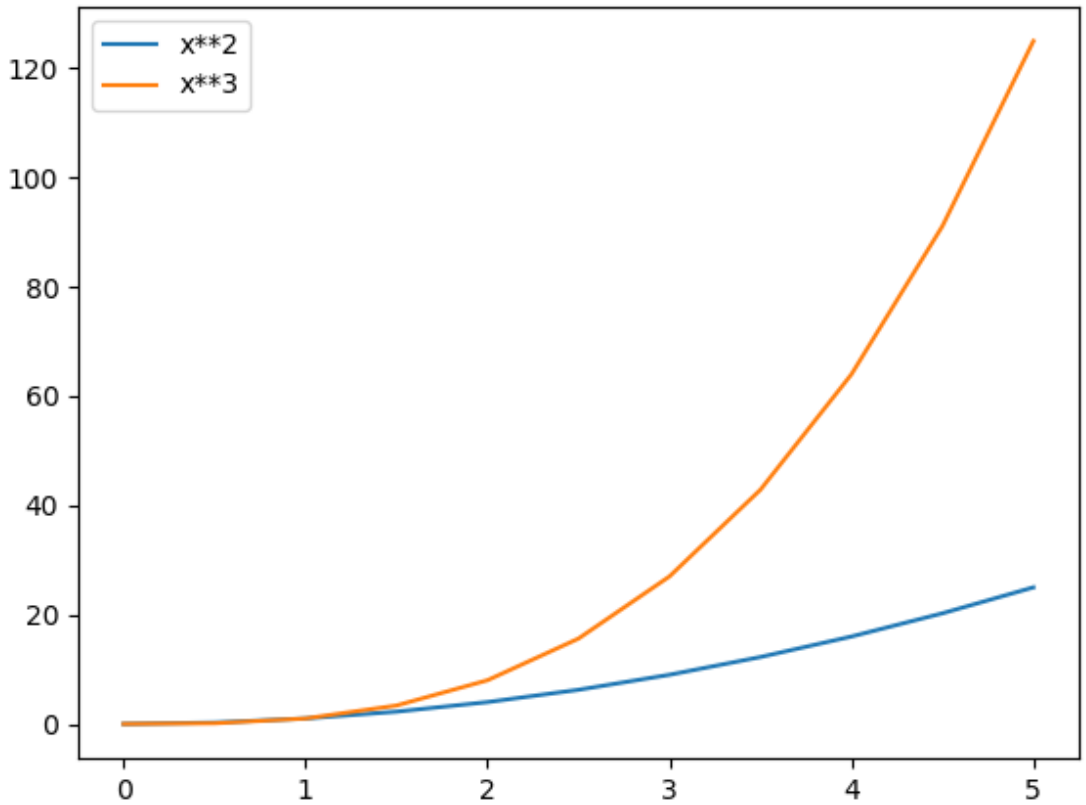
```
plt.savefig("filename.png", dpi=200)
```

Ajout d'une légende à un graphique

- Au moment le graphique, on peut préciser des noms aux différentes courbes avec l'argument *label*.
- Ce sont ces labels qui seront utilisés au moment de l'affichage de la légende du graphique.

```
fig = plt.figure()

ax = fig.add_axes([0.1,0.1,0.8,0.8])

ax.plot(x, x**2, label="x**2")
ax.plot(x, x**3, label="x**3")
ax.legend()

plt.show()
```

Figure obtenue

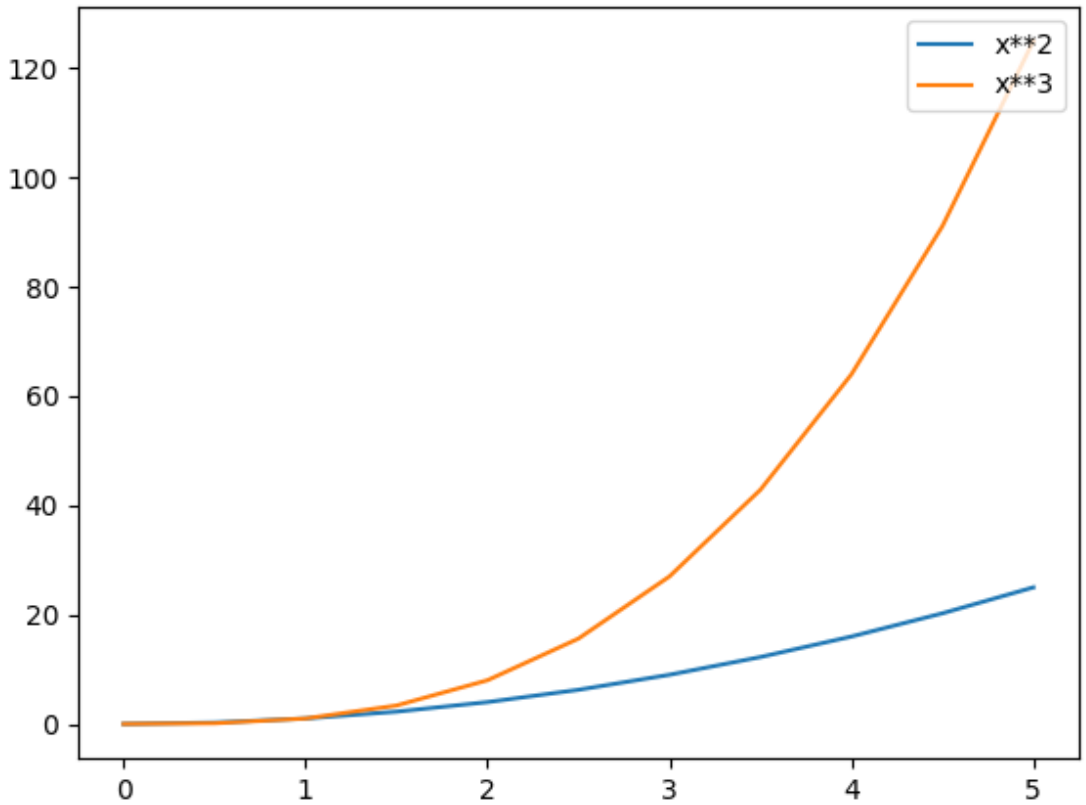


Placement de la légende

- La légende peut ensuite être positionnée avec l'argument *loc*.
  - En haut à droite : loc=1
  - En haut à gauche : loc=2
  - En bas à droite : loc=3
  - En bas à gauche : loc=3
  - On laisse matplotlib décider : loc=0

```
fig = plt.figure()

ax = fig.add_axes([0.1,0.1,0.8,0.8])

ax.plot(x, x**2, label="x**2")
ax.plot(x, x**3, label="x**3")
ax.legend(loc=1)

plt.show()
```

Figure obtenue



Section 4

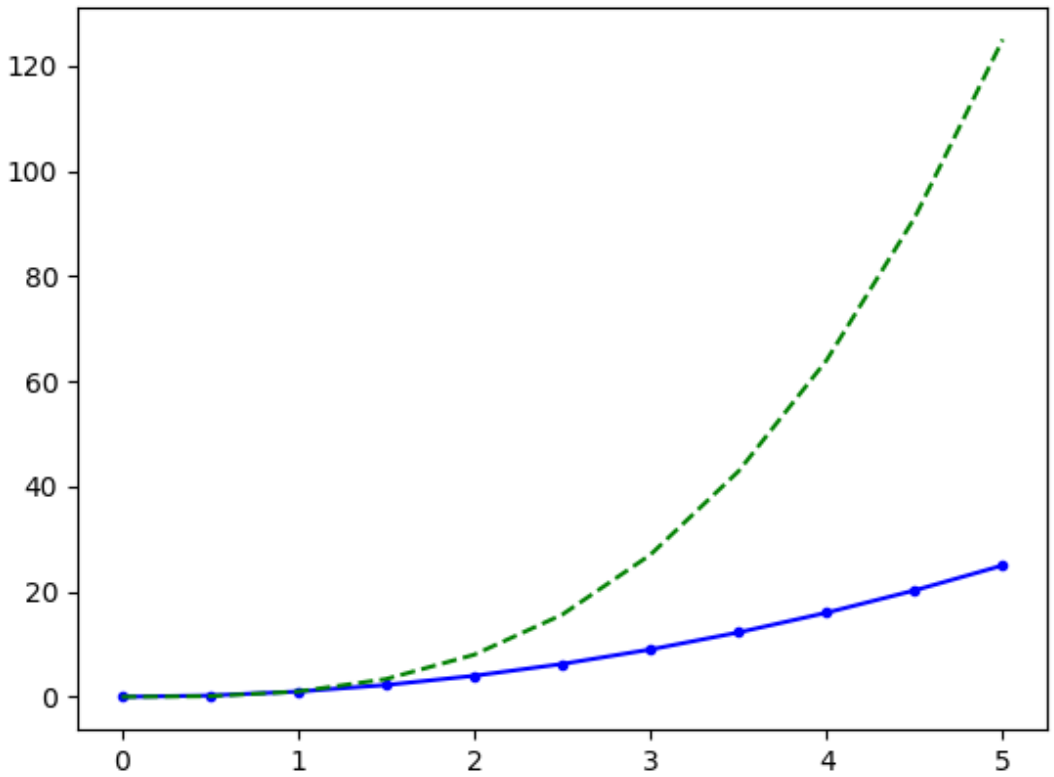
Gestion des couleurs et des types de lignes

Couleurs avec Matplotlib

- On peut gérer les couleurs avec des lettres. Par exemple "b" correspond à bleu, "g" à vert, etc....
- La gestion du type de ligne peut même se faire en même temps que couleur avec par exemple l'option "b.-" pour l'affichage d'une ligne bleu avec des points, ou "g–" pour l'affichage d'une ligne verte avec des tirets.

```
fig, ax = plt.subplots()
ax.plot(x, x**2, "b.-")
ax.plot(x, x**3, "g--")

plt.show()
```

Figure obtenue

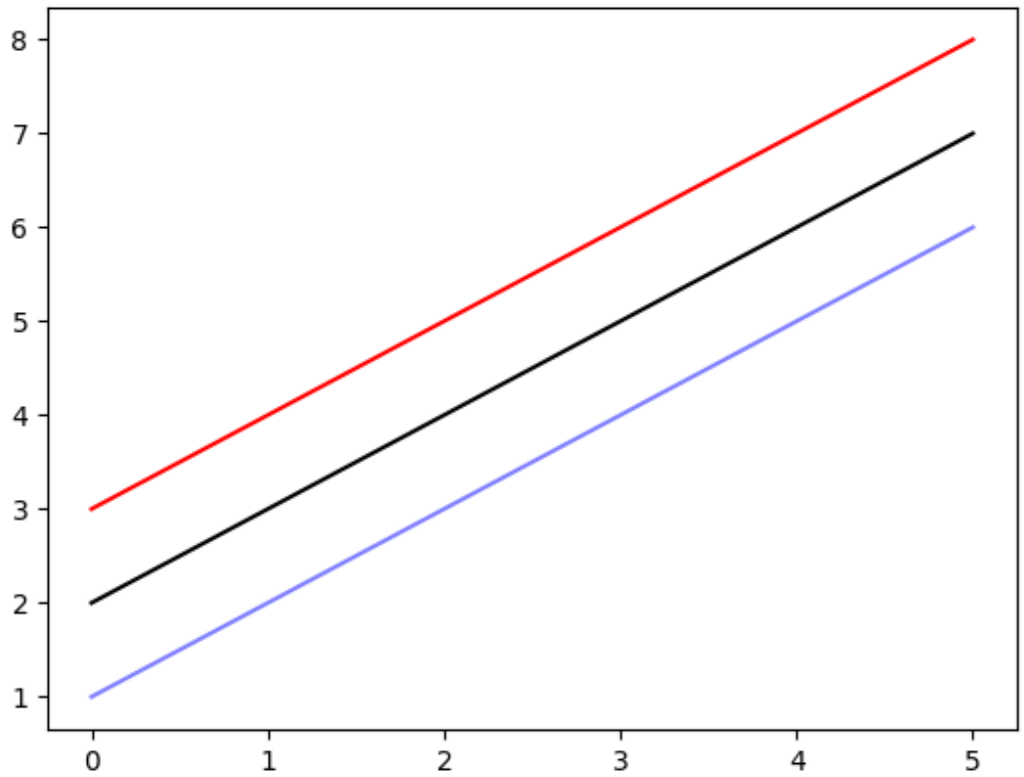


Couleurs avec Matplotlib

- On peut aussi préciser des couleurs avec leurs noms ou bien des codes RGB (Red, Green, Blue) avec des valeurs entre 0 et 1.
- Un autre paramètre *alpha* permet de gérer la transparence de la couleur.

```
fig, ax = plt.subplots()

ax.plot(x, x+1, color="blue", alpha=0.5) # half-transparant
ax.plot(x, x+2, color=(0,0,0)) # RGB black code
ax.plot(x, x+3, color=(1,0,0)) # RGB red code

plt.show()
```

Figure obtenue

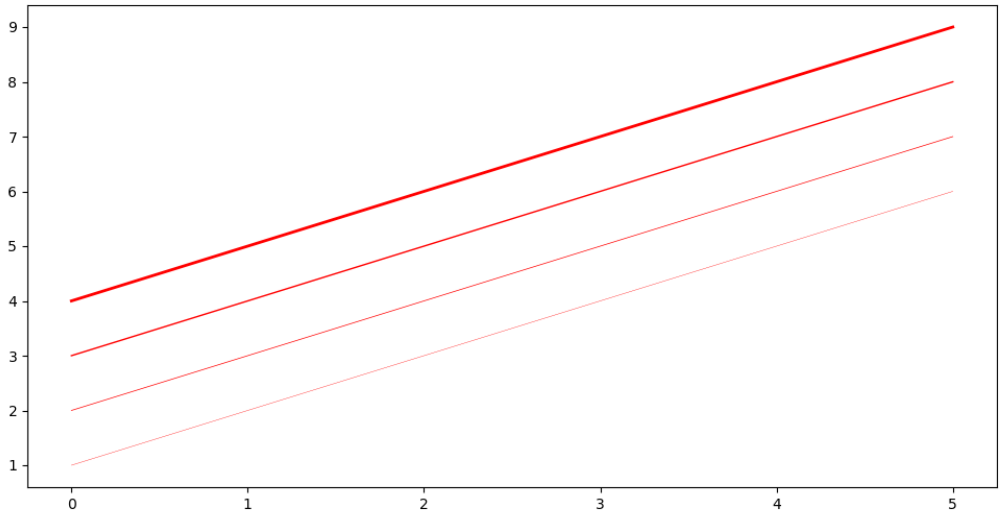


Epaisseur des lignes

- On peut gérer l'épaisseur des lignes avec l'attribut *linewidth* :

```
fig, ax = plt.subplots(figsize=(12,6))

ax.plot(x, x+1, color="red", linewidth=0.25)
ax.plot(x, x+2, color="red", linewidth=0.50)
ax.plot(x, x+3, color="red", linewidth=1.00)
ax.plot(x, x+4, color="red", linewidth=2.00)

plt.show()
```

Figure obtenue

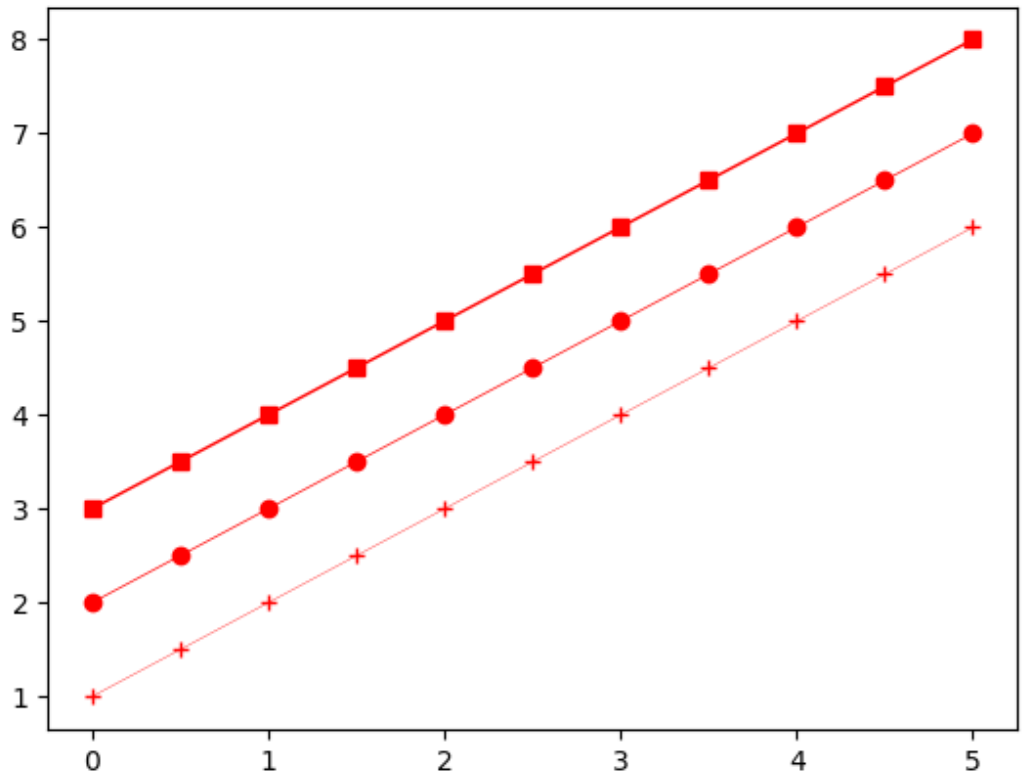


Ajout de marqueurs sur les lignes

- On peut ajouter des marqueurs sur les lignes avec l'attribut *marker* :

```
fig, ax = plt.subplots()

ax.plot(x, x+1, color="red", linewidth=0.25, marker="+")
ax.plot(x, x+2, color="red", linewidth=0.50, marker="o")
ax.plot(x, x+3, color="red", linewidth=1.00, marker="s")


plt.show()
```

Figure obtenue



Section 5

Affichage de graphiques spéciaux

Affichage de graphiques spéciaux

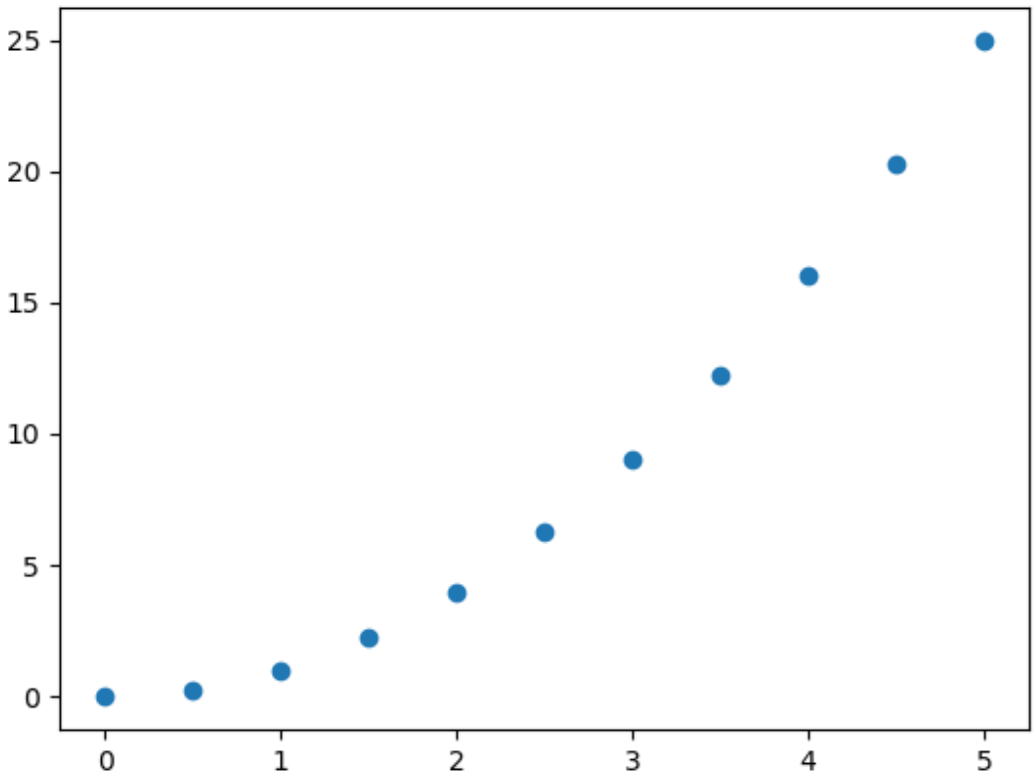
- Avec Matplotlib, il est possible de créer un grand nombre de type de graphiques différents :
  - Histogrammes
  - Nuages de points
  - Cartes thermiques
  - Courbes en 3D
- On va en voir quelques exemples dans ce cours.
- De nombreux autres exemples sont disponibles sur le site `https://matplotlib.org/`.

Affichage d'un nuage de points

- On a vu des affichage sous forme de courbe mais on peut aussi afficher sous forme d'un nuage de points toutes nos points (x, y):

```
plt.scatter(x,y)
plt.show()
```

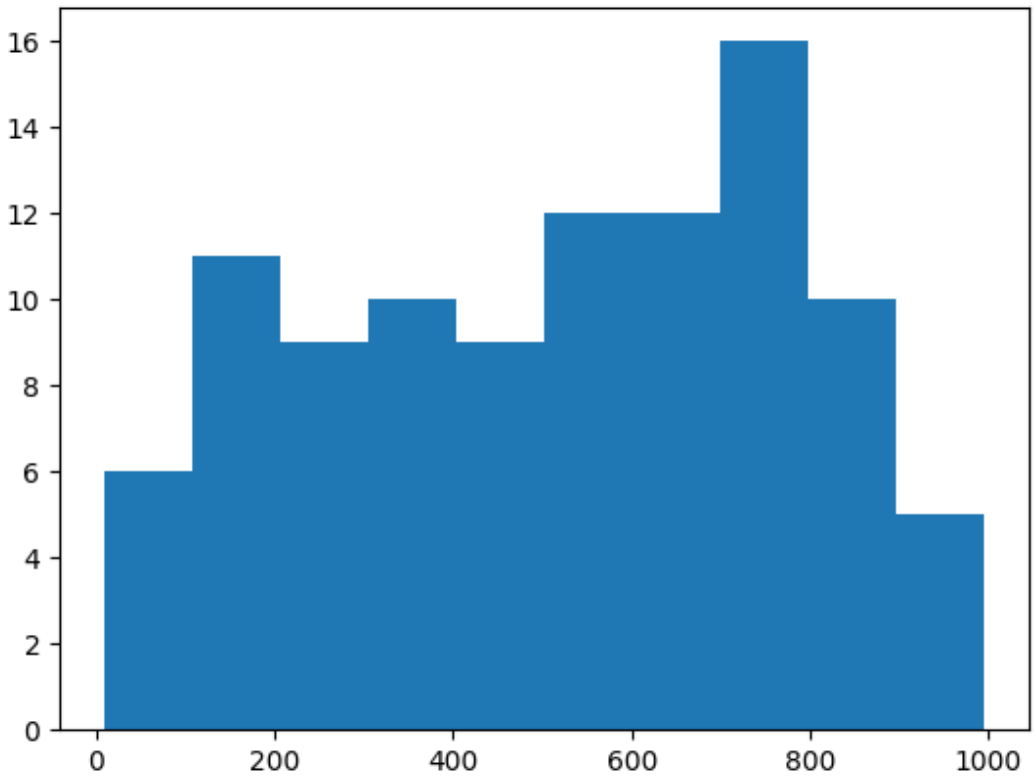
Figure obtenue



Affichage d'un histogramme

```
from random import sample
data = sample(range(1, 1000), 100)
plt.hist(data)
```

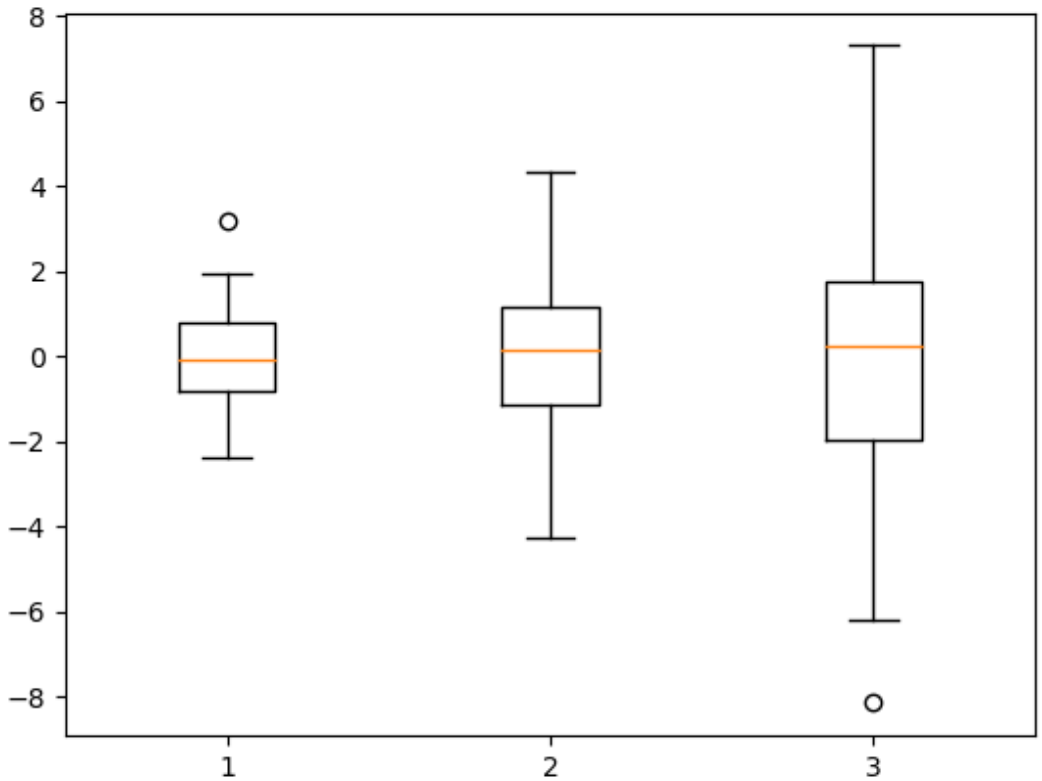
Figure obtenue



Affichage d'une boîte à moustaches

```
data = [np.random.normal(0, std, 100) for std in range(1, 4)]
plt.boxplot(data);

plt.show()
```

Figure obtenue



Affichage d'une carte thermique (*heatmap*) - 1er exemple

```
import numpy as np

x = np.random.randn(25)
x = np.reshape(x,(5,5))
print(x)
```

Affiche:

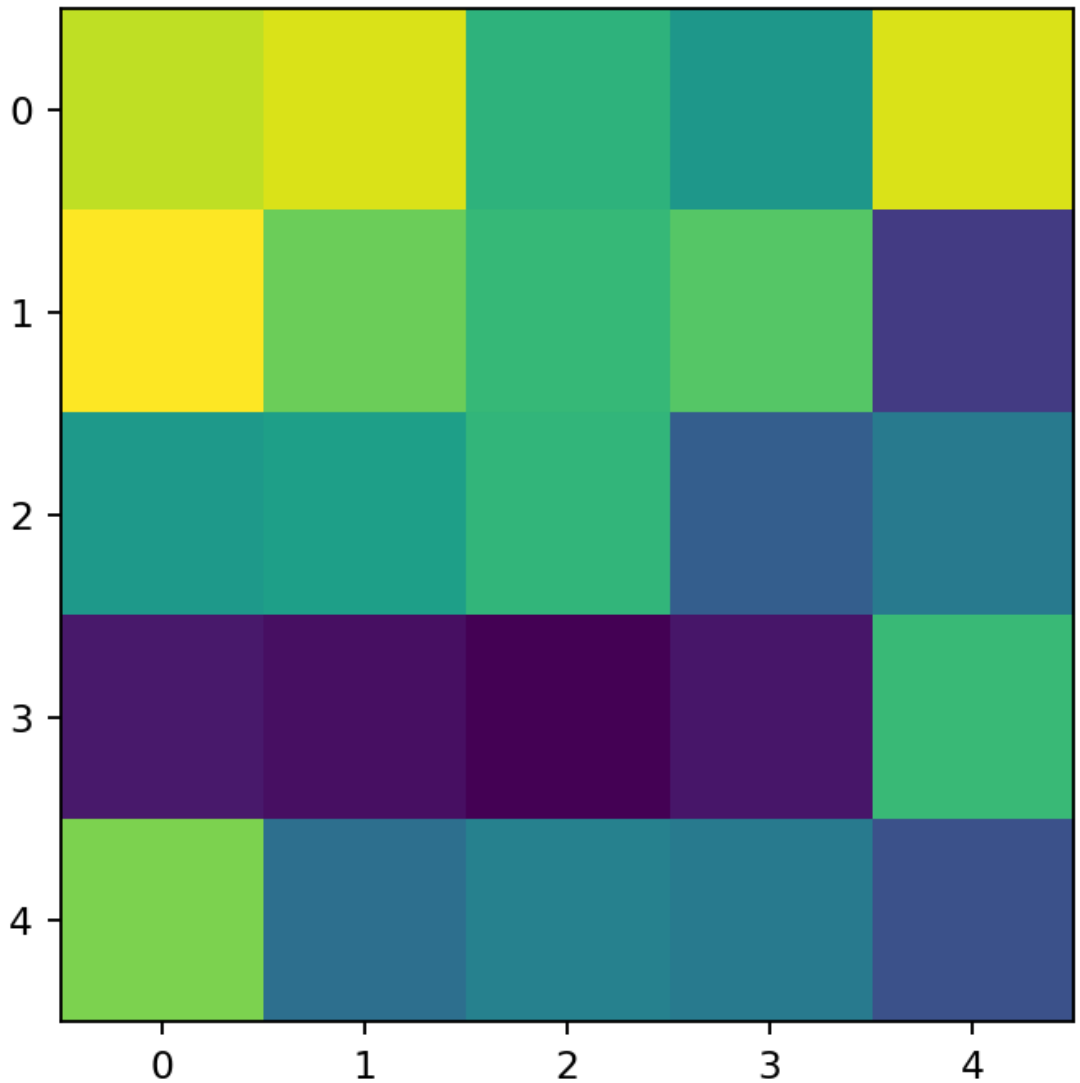
```
[[ 1.02022172 1.1263998 0.25196631 −0.09037885 1.12789931]
 [ 1.29865057 0.63968242 0.32835301 0.51919703 −1.13119984]
 [−0.06300587 0.02026477 0.29224104 −0.76392672 −0.44578138]
 [−1.4405842 −1.52432869 −1.64628573 −1.46894404 0.34495002]
 [ 0.72321632 −0.57355305 −0.35562998 −0.4401031 −0.91266008]]
```

Affichage d'une carte thermique (*heatmap*) - 1er exemple

Affichage de cette matrice :

```
import matplotlib.pyplot as plt

plt.imshow(x)
plt.show()
```

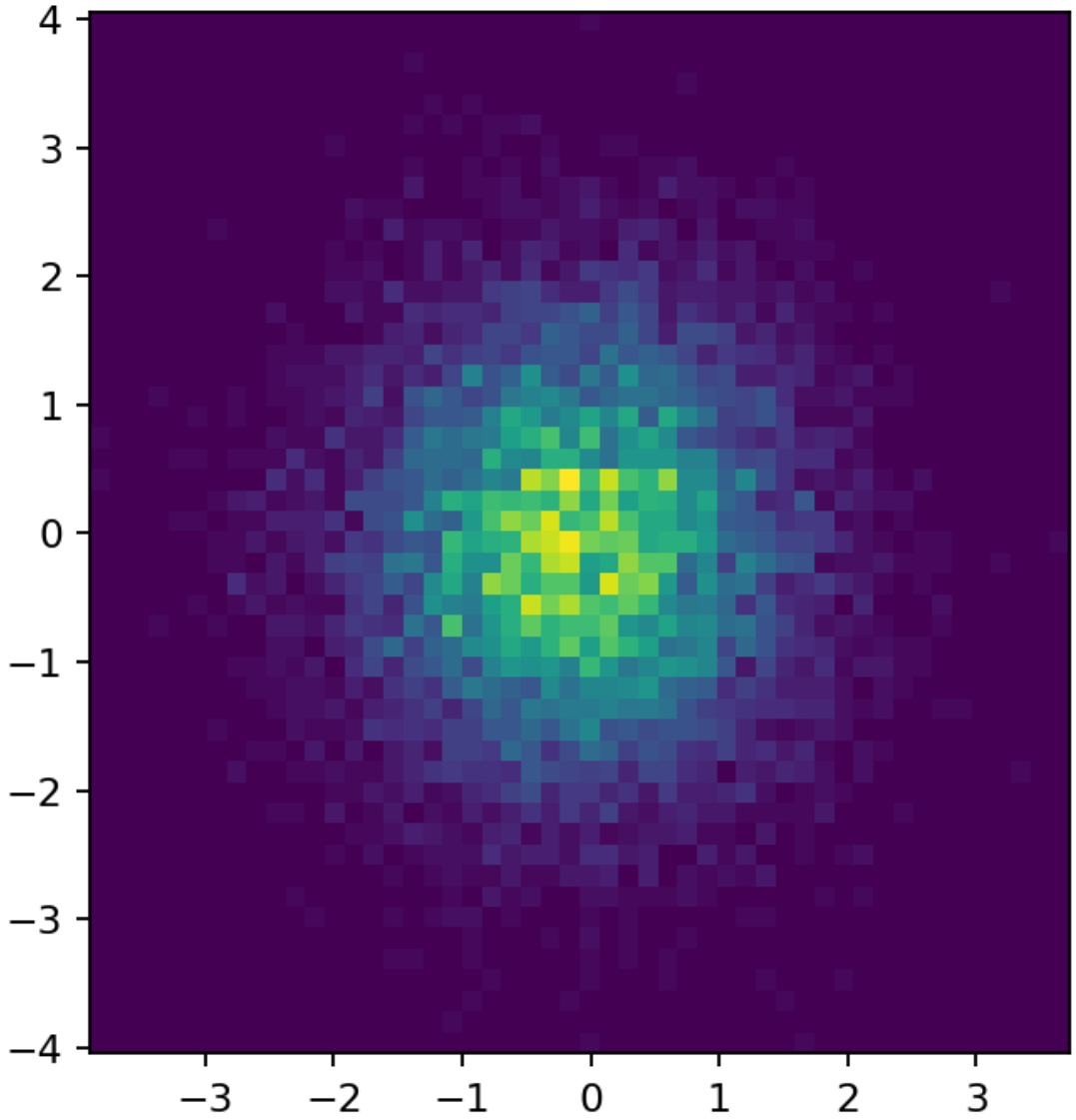
Figure obtenue - 1er exemple



Affichage d'une carte thermique (*heatmap*)

```
import numpy as np
import numpy.random
import matplotlib.pyplot as plt

x = np.random.randn(10000)
y = np.random.randn(10000)

heatmap, xedges, yedges = np.histogram2d(x, y, bins=50)
extent = [xedges[0], xedges[-1], yedges[0], yedges[-1]]

plt.imshow(heatmap, extent=extent)
plt.show()
```

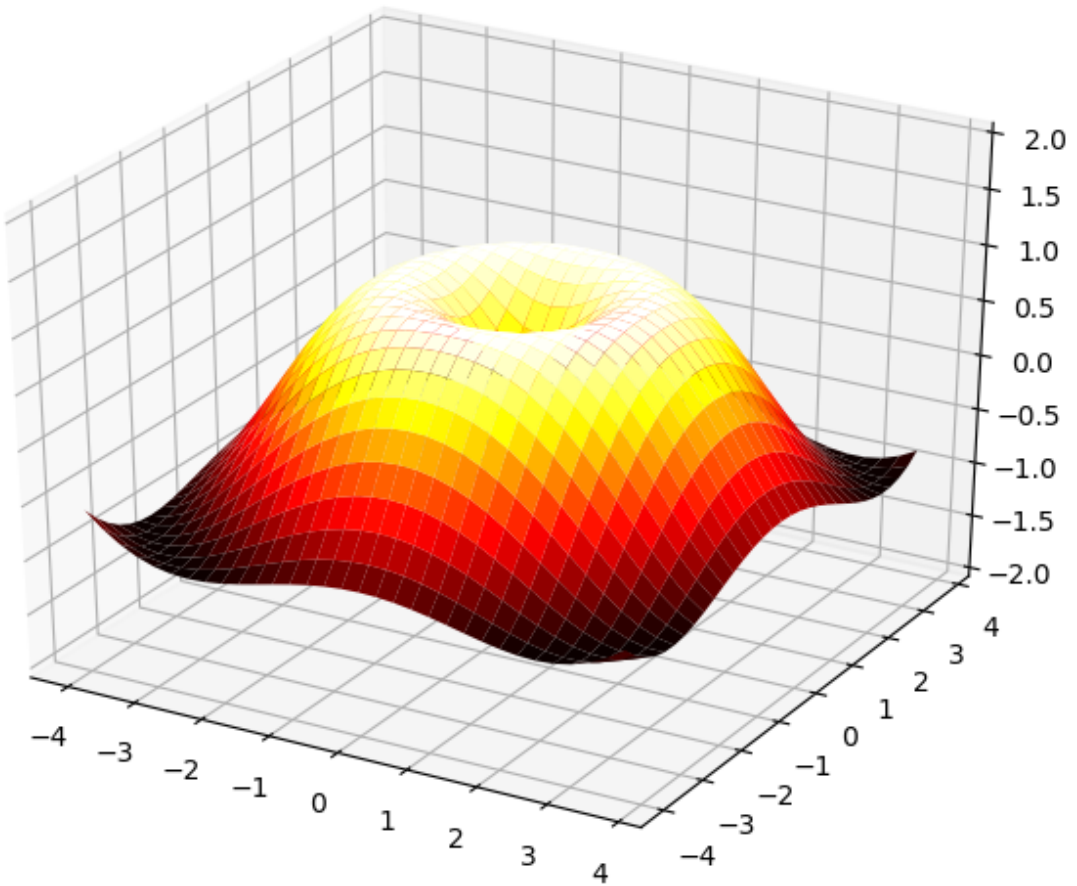
Figure obtenue



Affichage d'une courbe en 3D

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure()
ax = fig.add_subplot(projection='3d')

X = np.arange(-4, 4, 0.25)
Y = np.arange(-4, 4, 0.25)
X, Y = np.meshgrid(X, Y)

R = np.sqrt(X**2 + Y**2)
Z = np.sin(R)

ax.plot_surface(X, Y, Z, cmap=plt.cm.hot)
ax.set_zlim(-2,2)

plt.show()
```

Figure obtenue



Section 6

Exemple d'affichage d'un dataset pour la classification

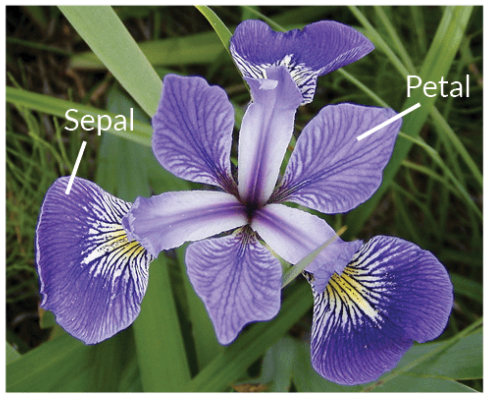
Chargement du dataset Iris et des librairies

- On peut charger directement des datasets étiquetés avec la librairie scikit-learn.

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
from sklearn import datasets

iris = datasets.load_iris()
```

- Ici *iris* est un dictionnaire qui contient différents éléments :
  - data : numpy array de taille (150,4) avec les valeurs de 4 attributs de 150 iris (fleurs).
  - target : numpy array de taille (150,) avec les différentes classes de ces iris. Il y a trois classes numérotées 0, 1 et 2.
  - feature_names : la liste des noms des attributs.
  - target_names : la listes des noms des différentes classes.

Trois classes d'iris dans ce dataset



Iris Versicolor

Iris Setosa

Iris Virginica

Création d'un dataframe pour manipuler plus facilement les données

```
X = iris.data
Y = iris.target
Y = Y.reshape(Y.shape[0],1)
list_col = iris.feature_names + ["target"]
df = pd.DataFrame(data = np.hstack((X,Y)), columns = list_col)
print(df)
```

Affichage du Dataframe

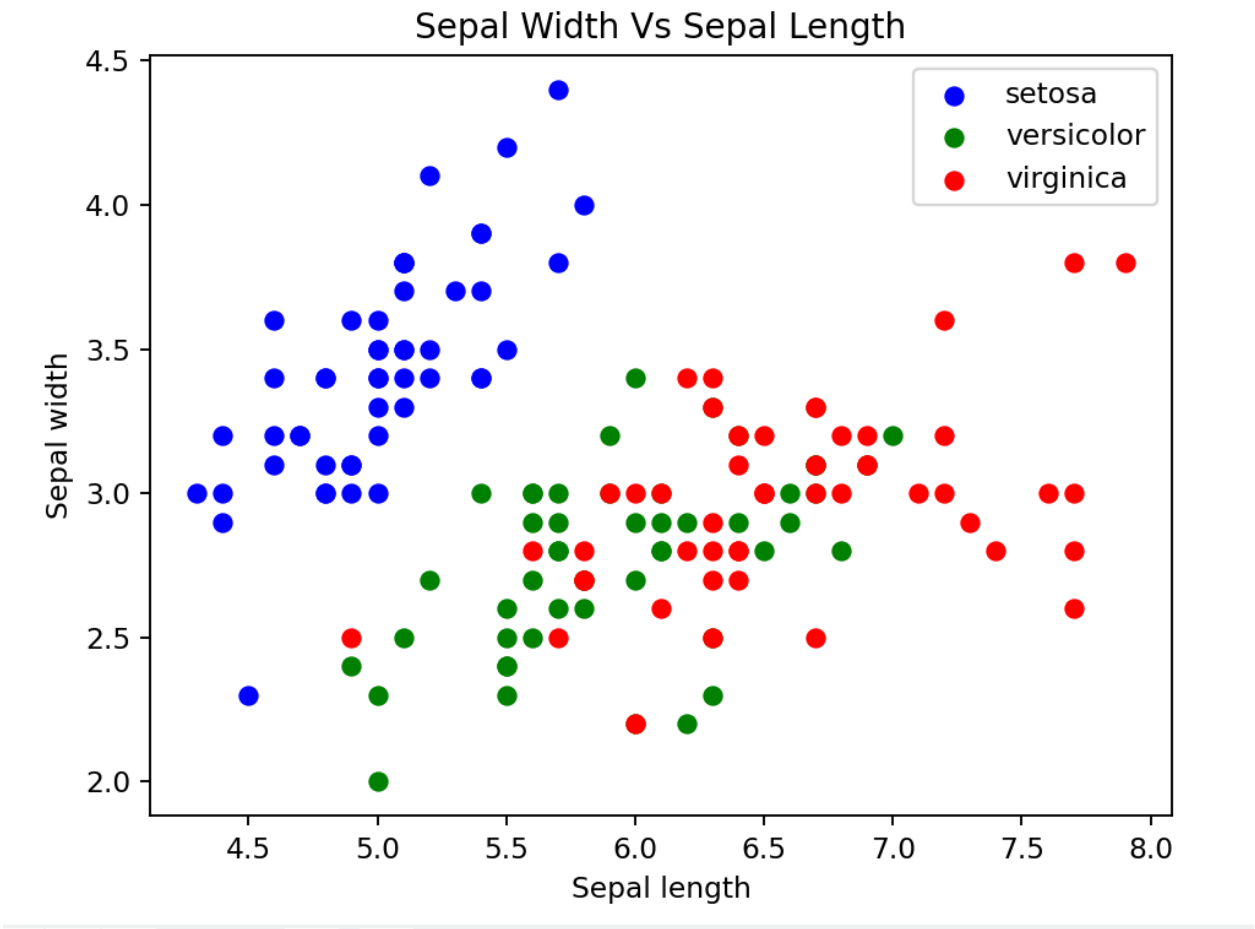
	sepal length (cm)	sepal width (cm)	...	petal width (cm)	target
0	5.1	3.5	...	0.2	0.0
1	4.9	3.0	...	0.2	0.0
2	4.7	3.2	...	0.2	0.0
3	4.6	3.1	...	0.2	0.0
4	5.0	3.6	...	0.2	0.0
..	...	...	...	...	...
145	6.7	3.0	...	2.3	2.0
146	6.3	2.5	...	1.9	2.0
147	6.5	3.0	...	2.0	2.0
148	6.2	3.4	...	2.3	2.0
149	5.9	3.0	...	1.8	2.0

Affichage des données en 2D

- Affichage de trois nuages de points (un par classe).
- Affichage suivant les deux premières dimensions.

```
list_color = ["blue","green","red"]

for i in range(3):
    df_subclass=df.loc[df["target"] ==i]
    plt.scatter(df_subclass["sepal length (cm)"],df_subclass["sepal width (cm)"],
        color=list_color[i], label=iris.target_names[i])

plt.xlabel("Sepal length")
plt.ylabel("Sepal width")
plt.title("Sepal Width Vs Sepal Length")
plt.legend()
plt.show()
```

Figure 2D obtenue



Affichage des données en 3D

- Affichage de trois nuages de points (un par classe).
- Affichage suivant les trois premières dimensions.

```
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
for i in range(3):
    df_subclass=df.loc[df["target"] ==i]
    ax.scatter(df_subclass["sepal length (cm)"],df_subclass["sepal width (cm)"],
        df_subclass["petal length (cm)"],color=list_color[i],label=iris.target_names
        [i])

ax.set_xlabel("Sepal length")
ax.set_ylabel("Sepal width")
ax.set_zlabel("Petal length")
plt.legend()
plt.title("Iris 3D plot")

plt.show()
```

Figure 3D obtenue