

4 - Représentation et visualisation des données

L'apprentissage automatique consiste à adapter des modèles aux données. C'est pourquoi nous commençons par montrer comment les données peuvent être représentées afin d'être comprises par l'ordinateur.

Au début de ce chapitre, nous avons cité la définition de l'apprentissage automatique donnée par Tom Mitchell : "Problème d'apprentissage bien posé : on dit qu'un programme informatique apprend de l'expérience E en ce qui concerne une certaine tâche T et une certaine mesure de performance P, si sa performance sur T, telle que mesurée par P, s'améliore avec l'expérience E." Les données constituent la "matière première" de l'apprentissage automatique. Il apprend à partir des données. Dans la définition de Mitchell, les "données" sont cachées derrière les termes "expérience E" et "mesure de performance P". Comme mentionné précédemment, nous avons besoin de données étiquetées pour apprendre et tester notre algorithme.

Cependant, il est recommandé de vous familiariser avec vos données avant de commencer à entraîner votre classificateur.

Numpy offre des structures de données idéales pour représenter vos données et Matplotlib offre de grandes possibilités pour visualiser vos données.

Dans ce qui suit, nous voulons montrer comment le faire en utilisant les données du module sklearn.

L'ensemble de données Iris, le "Hello World" de l'apprentissage automatique

Quel est le premier programme que vous avez vu ? Je parie que c'était un programme qui lançait "Hello World" dans un langage de programmation. Il est fort probable que j'aie raison. Presque chaque livre d'introduction ou tutoriel sur la programmation commence par un tel programme. C'est une tradition qui remonte au livre de 1968 "The C Programming Language" de Brian Kernighan et Dennis Ritchie !

La probabilité que le premier jeu de données que vous verrez dans un cours d'introduction à l'apprentissage automatique soit le "jeu de données Iris" est tout aussi élevée. Le jeu de données Iris contient les mesures de 150 fleurs d'iris de 3 espèces différentes :

- Iris-Setosa,
- Iris-Versicolor, et
- Iris-Virginica.



Iris Setosa



Iris Versicolor



Iris Virginica

Le jeu de données Iris est souvent utilisé pour sa simplicité. Ce jeu de données est contenu dans scikit-learn, mais avant d'approfondir le jeu de données Iris, nous allons examiner les autres jeux de données disponibles dans scikit-learn.

Chargement des données sur les iris avec Scikit-learn

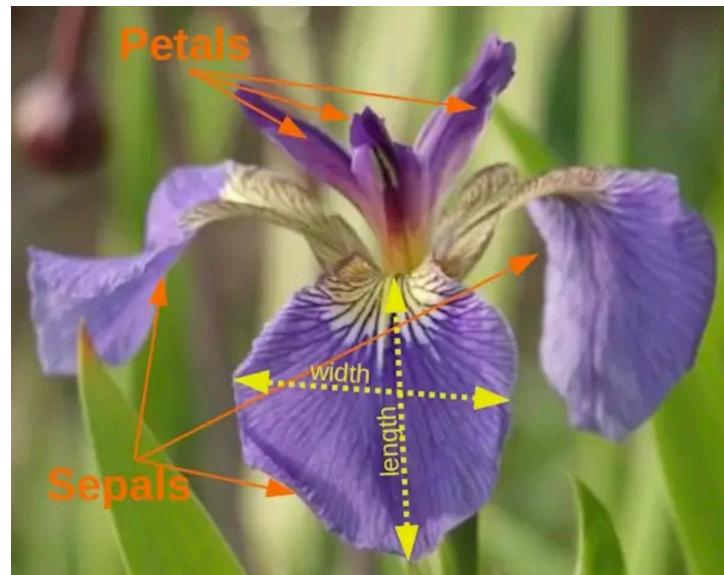
Par exemple, Scikit-learn dispose d'un ensemble très simple de données sur ces espèces d'iris. Les données se composent des éléments suivants :

Caractéristiques de l'ensemble de données sur les iris :

- longueur du sépale en cm
- largeur du sépale en cm
- Longueur des pétales en cm
- largeur des pétales en cm

Classes cibles à prédire :

- Iris Setosa
- Iris Versicolour
- Iris Virginica



Caractéristiques du jeu de données

scikit-learn intègre une copie du fichier CSV de l'iris ainsi qu'une fonction d'aide pour le charger dans des tableaux numpy :

Entrée [1]: `from sklearn.datasets import load_iris
iris = load_iris()`

Pour obtenir la liste de tous les datasets disponibles dans `sklearn` rendez-vous [ici](#) (https://scikit-learn.org/stable/datasets/toy_dataset.html).

Le jeu de données résultant est un objet [`Bunch`](#) (<https://scikit-learn.org/stable/modules/generated/sklearn.utils.Bunch.html>) :

Entrée [2]: `type(iris)`

Out[2]: `sklearn.utils.Bunch`

Vous pouvez voir ce qui est disponible pour ce type de données en utilisant la méthode `keys()` :

Entrée [5]: `iris.keys()`

Out[5]: `dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])`

Un objet `Bunch` est similaire à un dictionnaire, mais il permet en plus d'accéder aux clés comme si il s'agissait d'attributs :

Entrée [6]: *## Les deux appels sont équivalents*

```
print(iris["target_names"])
print(iris.target_names)
```

```
['setosa' 'versicolor' 'virginica']
['setosa' 'versicolor' 'virginica']
```

Les caractéristiques de chaque échantillon de fleur sont stockées dans l'attribut `data` de l'ensemble de données :

Entrée [7]: `n_samples, n_features = iris.data.shape`

```
print('Number of samples:', n_samples)
print('Number of features:', n_features)
# the sepal length, sepal width, petal length and petal width of the
print(iris.data[0])
```

```
Number of samples: 150
Number of features: 4
[5.1 3.5 1.4 0.2]
```

Les caractéristiques de chaque fleur sont stockées dans l'attribut `data` de l'ensemble de données. Jetons un coup d'œil à certains des échantillons :

```
Entrée [8]: # Flowers with the indices 12, 26, 89, and 114  
iris.data[[12, 26, 89, 114]]
```

```
Out[8]: array([[4.8, 3., 1.4, 0.1],  
               [5., 3.4, 1.6, 0.4],  
               [5.5, 2.5, 4., 1.3],  
               [5.8, 2.8, 5.1, 2.4]])
```

Les informations sur la classe de chaque échantillon, c'est-à-dire les étiquettes, sont stockées dans l'attribut "target" de l'ensemble de données :

```
Entrée [9]: print(iris.data.shape)
              print(iris.target.shape)

(150, 4)
(150.)
```

```
Entrée [10]: print(iris.target)
```

| |
|-----------------------------------------------------------------------------------------------------------------------------------------------|
| [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2] |
|-----------------------------------------------------------------------------------------------------------------------------------------------|

```
Entrée [11]: import numpy as np  
np.bincount(iris.target)  
  
Out[11]: array([50, 50, 50])
```

À l'aide de la fonction `bincount` de NumPy (ci-dessus), nous pouvons constater que les classes de cet ensemble de données sont distribuées de manière égale - il y a 50 fleurs de chaque espèce, avec :

- classe 0 : Iris Setosa
 - classe 1 : Iris Versicolor
 - classe 2 : Iris Virginica

Ces noms de classe sont stockés dans le dernier attribut, à savoir `targetNames` :

```
Entrée [12]: print(iris.target_names)
```

Outre la forme des données, nous pouvons également vérifier la forme des étiquettes, c'est-à-dire `target.shape` :

Chaque échantillon de fleur est une ligne dans le tableau de données, et les colonnes (caractéristiques) représentent les mesures de la fleur en centimètres. Par exemple, nous pouvons représenter cet ensemble de données sur les iris, composé de 150 échantillons et de 4 caractéristiques, par un tableau ou une matrice à 2 dimensions $\mathbb{R}^{150 \times 4}$ dans le format suivant :

$$\begin{pmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & x_4^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & x_4^{(2)} \\ \vdots & \vdots & \vdots & \vdots \\ x_1^{(150)} & x_2^{(150)} & x_3^{(150)} & x_4^{(150)} \end{pmatrix}$$

L'exposant désigne la ième ligne, et l'indice la jième caractéristique, respectivement.

En général, nous avons n rangées et k colonnes :

$$\begin{bmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & \dots & x_k^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & \dots & x_k^{(2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_1^{(150)} & x_2^{(150)} & x_3^{(150)} & \dots & x_k^{(150)} \end{bmatrix}$$

Entrée [13]: `print(iris.data.shape)`
`print(iris.target.shape)`

(150, 4)
(150,)

`bincount` de NumPy compte le nombre d'occurrences de chaque valeur dans un tableau d'entiers non-négatifs. Nous pouvons l'utiliser pour vérifier la distribution des classes dans l'ensemble de données :

Entrée [14]: `print(iris.target_names)`

['setosa' 'versicolor' 'virginica']

Visualisation des caractéristiques de l'ensemble de données Iris

Les données d'iris sont quadridimensionnelles, mais nous pouvons visualiser une ou deux des dimensions à la fois en utilisant un simple histogramme ou un nuage de points.

```
Entrée [15]: from sklearn.datasets import load_iris
iris = load_iris()
print(iris.data[iris.target==1][:5])

print(iris.data[iris.target==1, 0][:5])
```

```
[[7.  3.2 4.7 1.4]
 [6.4 3.2 4.5 1.5]
 [6.9 3.1 4.9 1.5]
 [5.5 2.3 4.  1.3]
 [6.5 2.8 4.6 1.5]]
[7.  6.4 6.9 5.5 6.5]
```

Histogrammes des caractéristiques

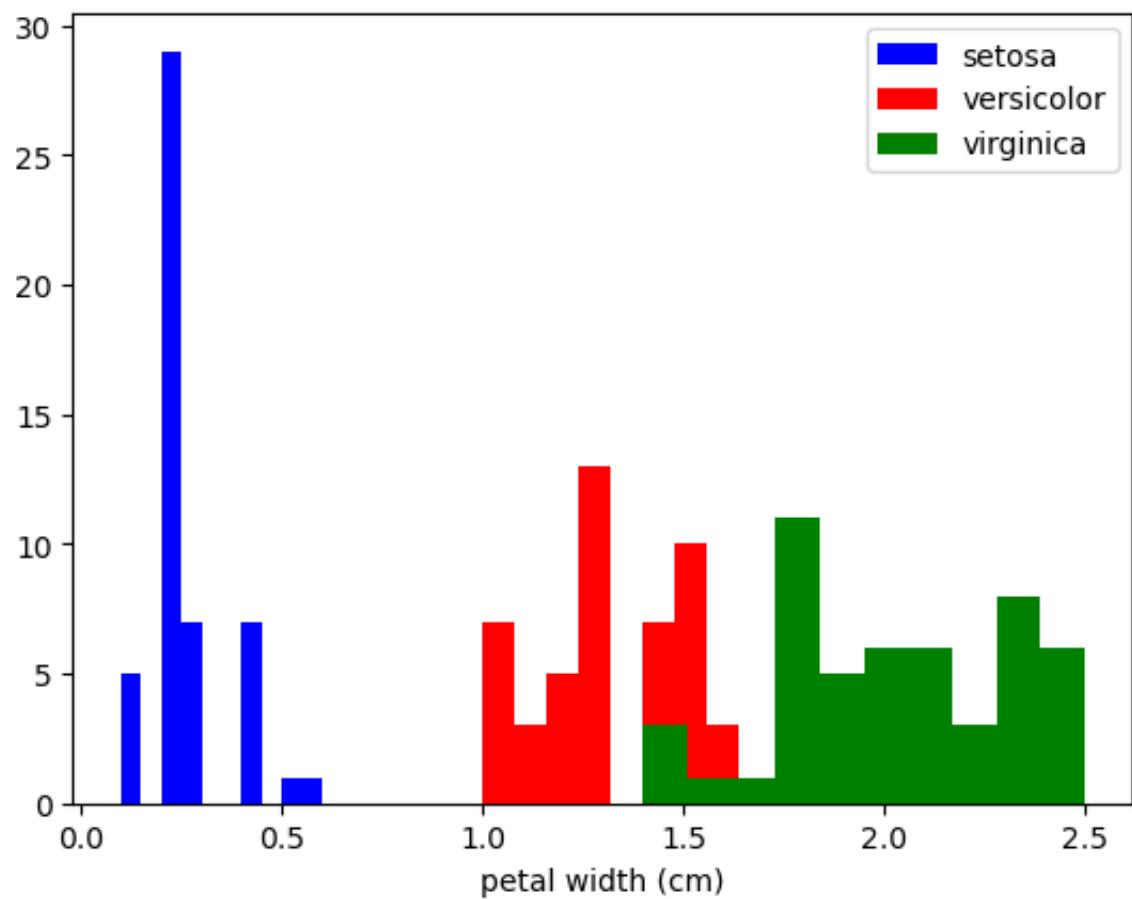
Entrée [17]:

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

fig, ax = plt.subplots()
x_index = 3
colors = ['blue', 'red', 'green']

for label, color in zip(range(len(iris.target_names)), colors):
    ax.hist(iris.data[iris.target==label, x_index],
            label=iris.target_names[label],
            color=color)

ax.set_xlabel(iris.feature_names[x_index])
ax.legend(loc='upper right')
plt.show()
```



Exercice

Regarder les histogrammes des autres caractéristiques, c'est-à-dire la longueur des pétales, la largeur des sépales et la longueur des sépales.

Entrée [18]: *## Compléter*

Diagramme de dispersion avec deux caractéristiques

Le diagramme d'aspect montre deux caractéristiques dans un diagramme :

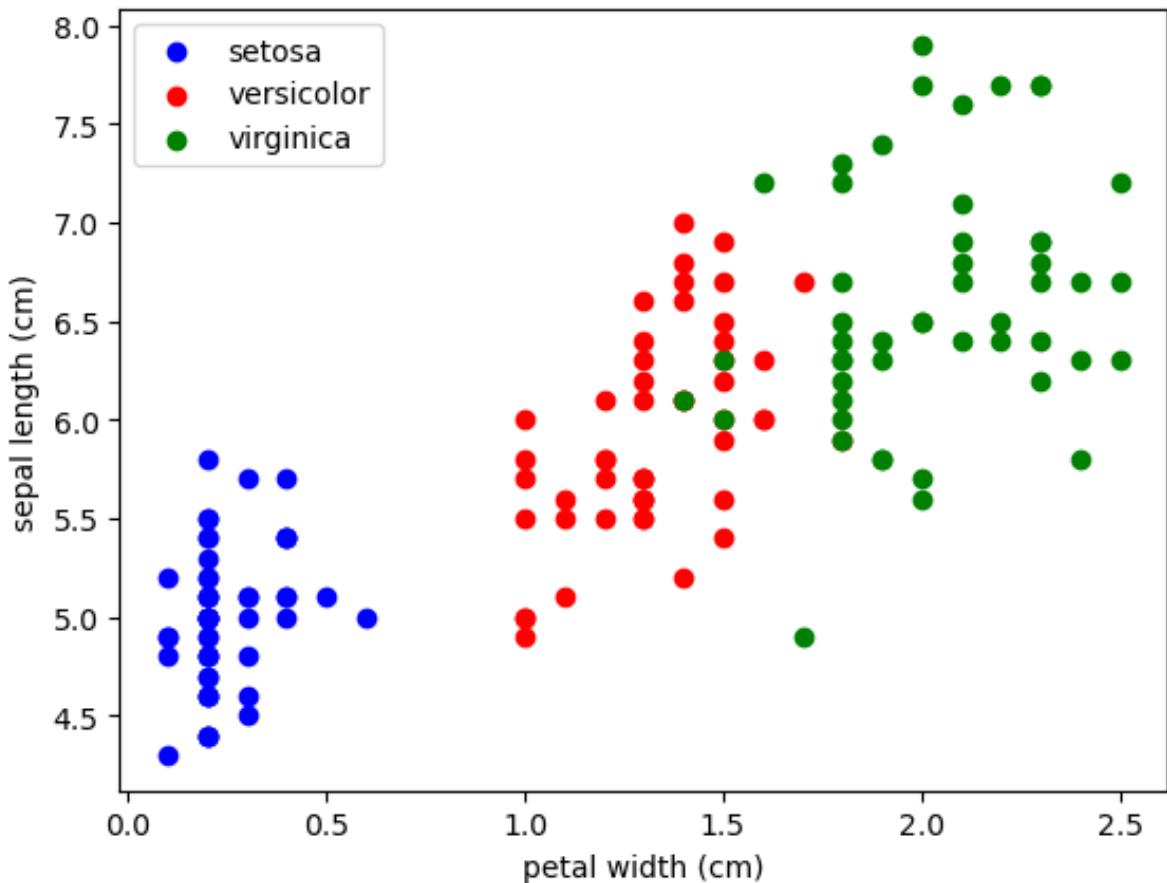
```
Entrée [19]: import matplotlib.pyplot as plt
fig, ax = plt.subplots()

x_index = 3
y_index = 0

colors = ['blue', 'red', 'green']

for label, color in zip(range(len(iris.target_names)), colors):
    ax.scatter(iris.data[iris.target==label, x_index],
               iris.data[iris.target==label, y_index],
               label=iris.target_names[label],
               c=color)

ax.set_xlabel(iris.feature_names[x_index])
ax.set_ylabel(iris.feature_names[y_index])
ax.legend(loc='upper left')
plt.show()
```



Exercice

Changez `x_index` et `y_index` dans le script ci-dessus.

Changez `x_index` et `y_index` dans le script ci-dessus et trouvez une combinaison de deux paramètres qui séparent au maximum les trois classes.

Entrée [20]: `## Completer`

Généralisation

Nous allons maintenant examiner toutes les combinaisons de caractéristiques dans un diagramme combiné :

Entrée [21]:

```
import matplotlib.pyplot as plt

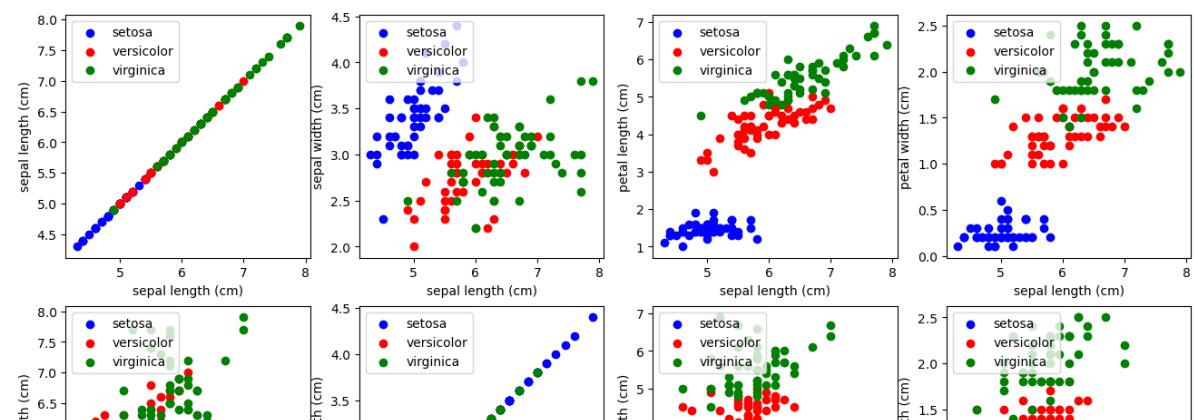
n = len(iris.feature_names)
fig, ax = plt.subplots(n, n, figsize=(16, 16))

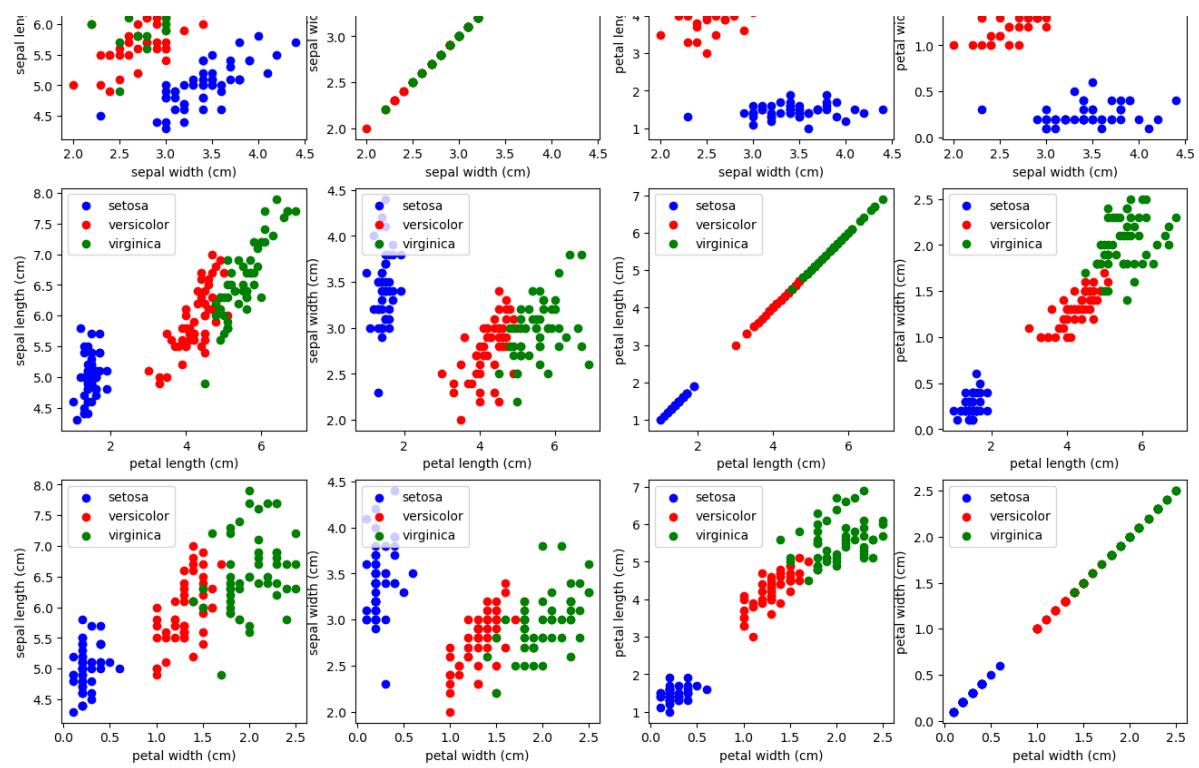
colors = ['blue', 'red', 'green']

for x in range(n):
    for y in range(n):
        xname = iris.feature_names[x]
        yname = iris.feature_names[y]
        for color_ind in range(len(iris.target_names)):
            ax[x, y].scatter(iris.data[iris.target==color_ind, x],
                             iris.data[iris.target==color_ind, y],
                             label=iris.target_names[color_ind],
                             c=colors[color_ind])

        ax[x, y].set_xlabel(xname)
        ax[x, y].set_ylabel(yname)
        ax[x, y].legend(loc='upper left')

plt.show()
```





Matrices Scatterplot

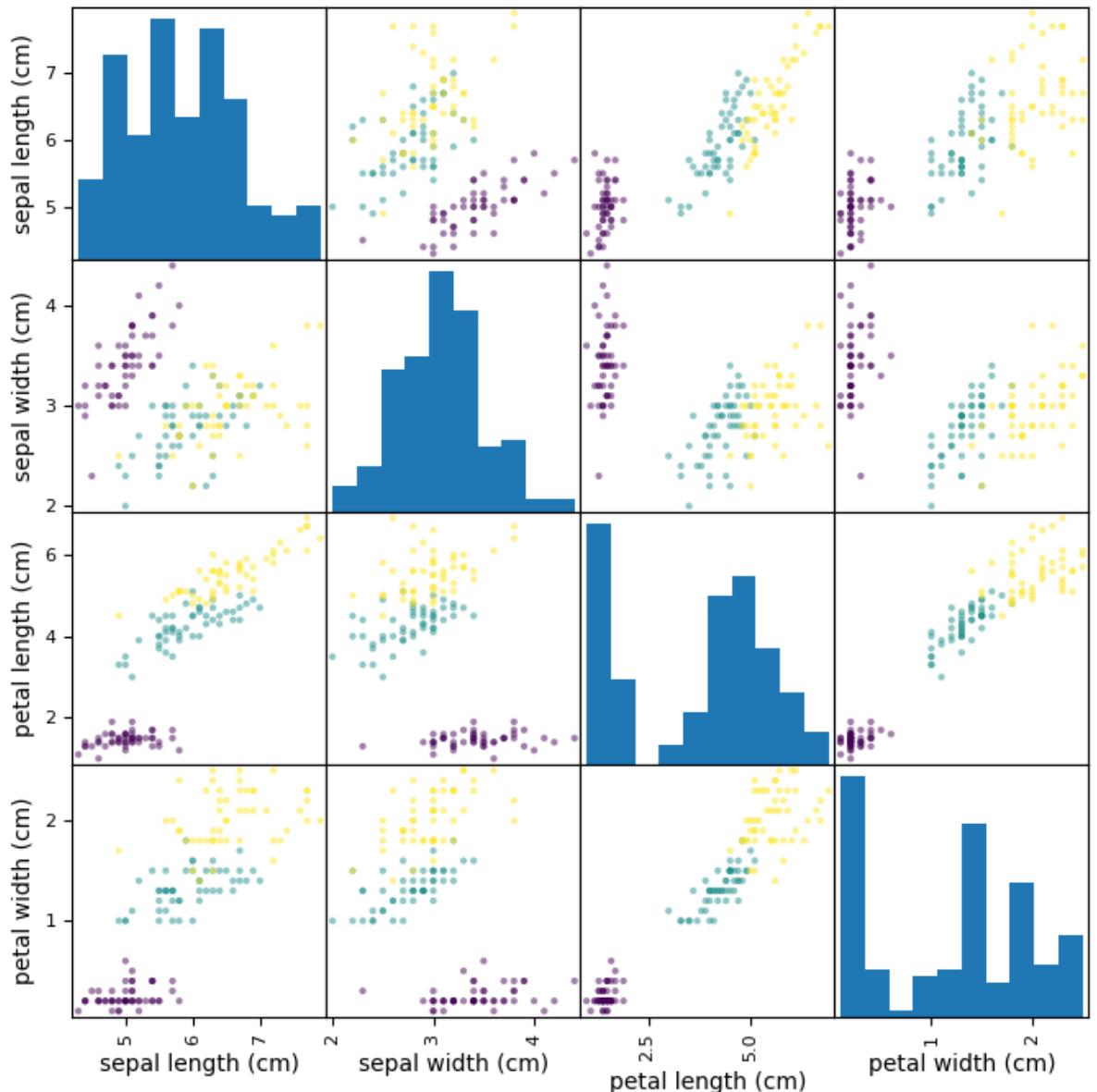
Au lieu de le faire manuellement, nous pouvons également utiliser la matrice [scatterplot](#)

(https://pandas.pydata.org/docs/reference/api/pandas.plotting.scatter_matrix.html) fournie par le module pandas.

Les matrices `scatterplot` affichent des diagrammes de dispersion entre toutes les caractéristiques de l'ensemble de données, ainsi que des histogrammes pour montrer la distribution de chaque caractéristique.

Entrée [22]: `import pandas as pd`

```
iris_df = pd.DataFrame(iris.data, columns=iris.feature_names)
pd.plotting.scatter_matrix(iris_df,
                           c=iris.target,
                           figsize=(8, 8))
);
```



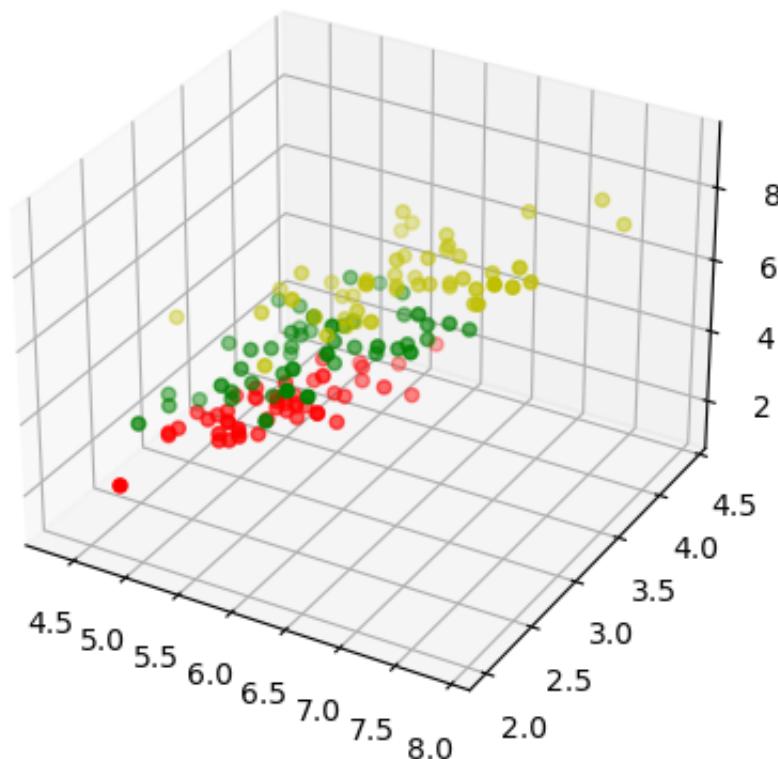
Visualisation tridimensionnelle

```
Entrée [23]: import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from mpl_toolkits.mplot3d import Axes3D
iris = load_iris()
X = []
for iclass in range(3):
    X.append([[], [], []])
    for i in range(len(iris.data)):
        if iris.target[i] == iclass:
            X[iclass][0].append(iris.data[i][0])
            X[iclass][1].append(iris.data[i][1])
            X[iclass][2].append(sum(iris.data[i][2:]))

colours = ("r", "g", "y")
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

for iclass in range(3):
    ax.scatter(X[iclass][0], X[iclass][1], X[iclass][2], c=colours[iclass])

plt.show()
```



```
Entrée [ ]:
```

