جامعة مولاي إسماعيل
UNIVERSITÉ MOULAY ISMAÏL

كلية العلوم والتقنيات
FACULTÉ DES SCIENCES ET TECHNIQUES

Département Informatique

MASTER EN SCIENCES ET TECHNIQUES

SIDI

SYSTÈMES D'INFORMATION DÉCISIONNELS ET IMAGERIE

FACULTÉ DES SCIENCES ET TECHNIQUES ERRACHIDIA

Introduction au module scikit-learn
TP1 – Machine Learning

ZEKKOURI Hassan

Vendredi 29 Novembre 2019

Responsable du module :

Prof. OUANAN Mohamed

REMERCIEMENT

Nous tenons à vous remercier monsieur Mohamed OUANAN pour votre formation et vos services.

Nous sommes également reconnaissant de nous avoir donner l'occasion de s'ouvrir sur un nouvel aspect de technologie qui est le Machine Learning et y mettre en œuvre nos compétences dont nous avons obtenu au cours du module.

Hassan ZEKKOURI

PLAN

Contents

Introduction au module scikit-learn *1*
REMERCIEMENT *2*
INTRODUCTION 16
I. Scikit-learn 17
II. Jeux de données en Scikit-Learn 17
III. TP1 17
IV. Jupyter notebook | TP1 18
A. B. Importation des libraires et manipulation d'un jeu de données

jupyter Machine Learning - TP1 Dernière Sauvegarde : il y a quelques secondes (modifié)

Logout

File Edit View Insert Cell Kernel Widgets Help Fiable Python 3

Exécuter Markdown

A. Importation des libraires

Pour Importer des modules python on utilise le mot clé import

Entrée [5]:
```
# A. Importation des libraires
import sklearn as skl
import numpy as np
import matplotlib.pyplot as plt
```

B. Manipulation d'un jeu de données

Parmi les jeux de données disponible dans le module scikit-learn on trouve: iris qu'on va importer ici!

1. Importation d'un jeu de données

Entrée [6]:
```
from sklearn.datasets import load_iris
iris = load_iris()
```

```
Alors la variable iris contient maintenant notre jeu de données.
```

Variable de jeu de données

Les variables des jeux de données comprennent un certain nombres champs parmi (tous ne sont pas toujours définis) : data, target, target names, feature names, DESCR:

- **.data** est un tableau de dimensions (n,m). Chacune des n lignes correspond à une donnée, chacune des m colonnes à un attribut.
- **.target** stocke les classes (étiquettes/label) de chaque instance (dans le cas supervisé) : c'est un vecteur de taille n dont la première valeur est la classe de la donnée de la première ligne de la matrice **.data**, la deuxième valeur est la classe de la deuxième ligne, etc.
- **.target_names** contient le nom des classes.
- **.target_names** contient le nom des classes.
- **.feature_names** contient le nom des attributs.
- **.DESCR** est un texte décrivant le jeu de données.

2. Afficher les données, noms des variables, le nom des classes

Entrée [7]:
```
# Afficher .data
print(iris.data)
```

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
 [5.1 3.8 1.5 0.3]
 [5.4 3.4 1.7 0.2]
```

Entrée [8]:
```
# Affichage des classes de chaque instance
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 0 0 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 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 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 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

Entrée [9]:
```
# Affichage de noms des classes
print(iris.target_names)
```

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

Entrée [10]:
```
# Affichage de noms des attributs/variables
print(iris.feature_names)
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

3. Nom des classes de chaque donnée

On peut répondre au question par un simple code python:

Entrée [11]:
```
print("Numéro de Donnée | Nom de sa classe")
for i in range(len(iris.data)):
    print(i ,"\t\t  ", iris.target_names[iris.target[i]])
```

```
Numéro de Donnée | Nom de sa classe
0                 setosa
1                 setosa
2                 setosa
3                 setosa
4                 setosa
5                 setosa
6                 setosa
7                 setosa
8                 setosa
9                 setosa
10                setosa
11                setosa
12                setosa
13                setosa
14                setosa
15                setosa
16                setosa
17                setosa
18                setosa
19                setosa
```

18

Les methodes et attributs

Entrée [12]:
```
## les methodes: .data par exemple
### Calculer la moyenne de chaque variable(axe = 0, colonne)
print(iris.data.mean(0))
```

```
[5.84333333 3.05733333 3.758      1.19933333]
```

Calculer la moyenne de chaque donnée(axe = 1, ligne)

Entrée [13]:
```
print(iris.data.mean(1))
```

```
[2.55  2.375 2.35  2.35  2.55  2.85  2.425 2.525 2.225 2.4   2.7   2.5
 2.325 2.125 2.8   3.    2.75  2.575 2.875 2.675 2.675 2.675 2.35  2.65
 2.575 2.45  2.6   2.6   2.55  2.425 2.425 2.675 2.725 2.825 2.425 2.4
 2.625 2.5   2.225 2.55  2.525 2.1   2.275 2.675 2.8   2.375 2.675 2.35
 2.675 2.475 4.075 3.9   4.1   3.275 3.85  3.575 3.975 2.9   3.85  3.3
 2.875 3.65  3.3   3.775 3.35  3.9   3.65  3.4   3.6   3.275 3.925 3.55
 3.8   3.7   3.725 3.85  3.95  4.1   3.725 3.2   3.2   3.15  3.4   3.85
 3.6   3.875 4.    3.575 3.5   3.325 3.425 3.775 3.4   2.9   3.45  3.525
 3.525 3.675 2.925 3.475 4.525 3.875 4.525 4.15  4.375 4.825 3.4   4.575
 4.2   4.85  4.2   4.075 4.35  3.8   4.025 4.3   4.2   5.1   4.875 3.675
 4.525 3.825 4.8   3.925 4.45  4.55  3.9   3.95  4.225 4.4   4.55  5.025
 4.25  3.925 3.925 4.775 4.425 4.2   3.9   4.375 4.45  4.35  3.875 4.55
 4.55  4.3   3.925 4.175 4.325 3.95 ]
```

Calculer le nombre d'objets dans target

Entrée [14]:
```
## les attributs: .target par exemple
### Calculer le nombre d'objets dans target
print(iris.target.size)
```

```
150
```

La fonction `.len()` retourne la longueur de l'objet passé en paramètre. ¶

Entrée [15]:
```
print(len(iris.data)) # >>> affiche 150 données comme lenght
```
```
150
```

La fonction `help()` affiche de l'aide à propos la fonction/methode/module passé en paramètre.

Entrée [16]:
```
help(len)
```
```
Help on built-in function len in module builtins:

len(obj, /)
    Return the number of items in a container.
```

L'instruction `iris.target_names[2]` affiche le nom de la troixième (dernière) classe.

Entrée [17]:
```
print(iris.target_names[2])
```
```
virginica
```

L'instruction `iris.target_names[-1]` affiche le nom de la dernière classe.

Entrée [18]:
```
print(iris.target_names[-1])
```
```
virginica
```

L'instruction `iris.target_names[len(iris.target_names)]` affiche erreur: index out of range.

Entrée [15]:
```
print(iris.target_names[len(iris.target_names)])
```
```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-15-1e1c5c3b5543> in <module>
----> 1 print(iris.target_names[len(iris.target_names)])

IndexError: index 3 is out of bounds for axis 0 with size 3
```

L'instruction `iris.data[0][1]` affiche la valeur de la deuxième variable pour la première donnée.

Entrée [61]:
```
print(iris.data[0][1])
```
```
3.5
```

L'instruction `iris.data[:,1]` affiche la valeur de la deuxième variable pour tous les données.

Entrée [62]:
```
print(iris.data[:,1])
```
```
[3.5 3.  3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 3.7 3.4 3.  3.  4.  4.4 3.9 3.5
 3.8 3.8 3.4 3.7 3.6 3.3 3.4 3.  3.4 3.5 3.4 3.2 3.1 3.4 4.1 4.2 3.1 3.2
 3.5 3.6 3.  3.4 3.5 2.3 3.2 3.5 3.8 3.  3.8 3.2 3.7 3.3 3.2 3.2 3.1 2.3
 2.8 2.8 3.3 2.4 2.9 2.7 2.  3.  2.2 2.9 2.9 3.1 3.  2.7 2.2 2.5 3.2 2.8
 2.5 2.8 2.9 3.  2.8 3.  2.9 2.6 2.4 2.4 2.7 2.7 3.  3.4 3.1 2.3 3.  2.5
 2.6 3.  2.6 2.3 2.7 3.  2.9 2.9 2.5 2.8 3.3 2.7 3.  2.9 3.  3.  2.5 2.9
 2.5 3.6 3.2 2.7 3.  2.5 2.8 3.2 3.  3.8 2.6 2.2 3.2 2.8 2.8 2.7 3.3 3.2
 2.8 3.  2.8 3.  2.8 3.8 2.8 2.8 2.6 3.  3.4 3.1 3.  3.1 3.1 3.1 2.7 3.2
 3.3 3.  2.5 3.  3.4 3. ]
```

4. Afficher: moyenne, ecart-type, min et le max pour chaque variable => axe = 0

Entrée [63]:
```
print("Les variables:")
print(iris.feature_names)
```

```
Les variables:
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

L'instruction `iris.data.mean(0)` retourne la moyenne.

Entrée [64]:
```
print(iris.data.mean(0))
```

```
[5.84333333 3.05733333 3.758      1.19933333]
```

L'instruction `iris.data.std(0)` retourne l'ecart-type.

Entrée [65]:
```
print(iris.data.std(0))
```

```
[0.82530129 0.43441097 1.75940407 0.75969263]
```

L'instruction `iris.data.min(0)` retourne le minimum.

Entrée [66]:
```
print(iris.data.min(0))
```

```
[4.3 2.  1.  0.1]
```

L'instruction `iris.data.maxn(0)` retourne le maximum.

Entrée [67]:
```
print(iris.data.max(0))
```

```
[7.9 4.4 6.9 2.5]
```

..... 20

C. Téléchargement et Importation de données *21*

5. Afficher: nombre de données, nombre de variables et le nombre de classes

Entrée [68]:
```
print("Le nombre de données est:", iris.data.shape[0])
print("Le nombre de variables est:", iris.data.shape[1])
print("Le nombre de classes est:", iris.target_names.size)
```

```
Le nombre de données est: 150
Le nombre de variables est: 4
Le nombre de classes est: 3
```

C. Téléchargement et importation de données

Entrée [69]:
```
from sklearn.datasets import fetch_mldata
mnist = fetch_mldata('MNIST original') # on peut utiliser les données télécharger aussi
```

```
C:\Users\elitebook\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:85: DeprecationWarning: Function fetch_mldata is deprecated; fetch_mldata was deprecated in version 0.20 and will be removed in version 0.22. Please use fetch_openml.
  warnings.warn(msg, category=DeprecationWarning)
C:\Users\elitebook\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:85: DeprecationWarning: Function mldata_filename is deprecated; mldata_filename was deprecated in version 0.20 and will be removed in version 0.22. Please use fetch_openml.
  warnings.warn(msg, category=DeprecationWarning)

---------------------------------------------------------------------------
TimeoutError                              Traceback (most recent call last)
~\Anaconda3\lib\urllib\request.py in do_open(self, http_class, req, **http_conn_args)
   1316                 h.request(req.get_method(), req.selector, req.data, headers,
-> 1317                           encode_chunked=req.has_header('Transfer-encoding'))
   1318             except OSError as err: # timeout error

~\Anaconda3\lib\http\client.py in request(self, method, url, body, headers, encode_chunked)
   1243         """Send a complete request to the server."""
-> 1244         self._send_request(method, url, body, headers, encode_chunked)
   1245

~\Anaconda3\lib\http\client.py in _send_request(self, method, url, body, headers, encode_chunked)
   1289             body = _encode(body, 'body')
```

On remarque que Python demande d'utiliser .fetch_openml qui a remplacer .fetch_mldata

..... 22

Entrée [70]:
```
# Importation d'un jeu de données
from sklearn.datasets import fetch_openml
mnist = fetch_openml('MNIST original') # donne invalidURL, on charge les données téléchargées ou changer en 'mnist_784'
```

```
---------------------------------------------------------------------------
InvalidURL                                Traceback (most recent call last)
~\Anaconda3\lib\site-packages\sklearn\datasets\openml.py in wrapper()
     46             try:
---> 47                 return f()
     48             except HTTPError:

~\Anaconda3\lib\site-packages\sklearn\datasets\openml.py in _load_json()
    150     def _load_json():
--> 151         with closing(_open_openml_url(url, data_home)) as response:
    152             return json.loads(response.read().decode("utf-8"))

~\Anaconda3\lib\site-packages\sklearn\datasets\openml.py in _open_openml_url(openml_path, data_home)
     99         try:
--> 100             with closing(urlopen(req)) as fsrc:
    101                 if is_gzip(fsrc):

~\Anaconda3\lib\urllib\request.py in urlopen(url, data, timeout, cafile, capath, cadefault, context)
    221         opener = _opener
--> 222     return opener.open(url, data, timeout)
```

Pour résoude ce pronlème on passe la valeur 'mnist_784'

Entrée [71]:
```
# Importation d'un jeu de données
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784')
```

Afficher la matrice de données

Entrée [72]:
```
print("Data:\n", mnist.data)
```

```
Data:
 [[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

Afficher le nombres de données et variables

Entrée [73]:
```
donnee, variable = mnist.data.shape # ou np.shape(mnist.data)
print("Le nombre de donnees:", donnee)
print("Le nombre de variables:", variable)
```

```
Le nombre de donnees: 70000
Le nombre de variables: 784
```

Afficher le nombres de données et variables

Entrée [*]:
```
print("Numéro de Donnée | Numéro de sa classe")
for i in range(len(mnist.data)):
    print(i ,"\t\t  ", mnist.target[i])
```

```
Numéro de Donnée | Numéro de sa classe
0                5
1                0
2                4
3                1
4                9
5                2
6                1
7                3
8                1
9                4
10               3
```

Entrée [75]:
```
print("Les variables:")
print(mnist.feature_names)
```

```
Les variables:
['pixel1', 'pixel2', 'pixel3', 'pixel4', 'pixel5', 'pixel6', 'pixel7', 'pixel8', 'pixel9', 'pixel10', 'pixel11', 'pixel12', 'pi
xel13', 'pixel14', 'pixel15', 'pixel16', 'pixel17', 'pixel18', 'pixel19', 'pixel20', 'pixel21', 'pixel22', 'pixel23', 'pixel2
4', 'pixel25', 'pixel26', 'pixel27', 'pixel28', 'pixel29', 'pixel30', 'pixel31', 'pixel32', 'pixel33', 'pixel34', 'pixel35', 'p
ixel36', 'pixel37', 'pixel38', 'pixel39', 'pixel40', 'pixel41', 'pixel42', 'pixel43', 'pixel44', 'pixel45', 'pixel46', 'pixel4
7', 'pixel48', 'pixel49', 'pixel50', 'pixel51', 'pixel52', 'pixel53', 'pixel54', 'pixel55', 'pixel56', 'pixel57', 'pixel58', 'p
```

Afficher l'écart-type des variables

Entrée [77]:
```
print(mnist.data.std(0)) # 784 variables
```

```
[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 4.40061249e-01 1.26020885e+00 8.16397430e-01 3.40165596e-02
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
```

Afficher le min des variables

Entrée [78]:
```
print(mnist.data.min(0)) # 784 variables
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

Afficher le max des variables

Entrée [79]:
```
print(mnist.data.max(0)) # 784 variables
```

```
[  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0. 116. 254.
 216.   9.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.  16.  47. 236. 254. 255. 254. 255. 255. 255. 255.
 255. 255. 255. 255. 255. 255. 244. 255. 184. 197.   0.   0.   0.   0.
   0.   0.  64.  29. 134. 192. 254. 255. 255. 255. 255. 255. 255. 255.
 255. 255. 255. 255. 255. 255. 255. 255. 255. 255. 255. 165.   0.   0.
   0.   0. 141. 101. 139. 255. 255. 255. 255. 255. 255. 255. 255. 255.
 255. 255. 255. 255. 255. 255. 255. 255. 255. 255. 255. 234. 163.   0.
```

Nombre de classe avec la fonction `numpy.unique()`

Entrée [80]:
```
print("Les classes: ", np.unique(mnist.target))
print("Le nombre des classes: ", len(np.unique(mnist.target)))
```

```
Les classes:  ['0' '1' '2' '3' '4' '5' '6' '7' '8' '9']
Le nombre des classes:  10
```

.... 22
C. *Génération de données et affichage* 24

D. Génération de données et affichage

1. La fonction `help()`

Entrée [81]:
```
from sklearn.datasets import make_blobs
help(make_blobs)
```

```
Help on function make_blobs in module sklearn.datasets.samples_generator:

make_blobs(n_samples=100, n_features=2, centers=None, cluster_std=1.0, center_box=(-10.0, 10.0), shuffle=True, random_state=
None)
    Generate isotropic Gaussian blobs for clustering.

    Read more in the :ref:`User Guide <sample_generators>`.

    Parameters
    ----------
    n_samples : int or array-like, optional (default=100)
        If int, it is the total number of points equally divided among
        clusters.
        If array-like, each element of the sequence indicates
        the number of samples per cluster.

    n_features : int, optional (default=2)
        The number of features for each sample.

    centers : int or array of shape [n_centers, n_features], optional
```

On peut compredre que la methode génére:

- un jeu de données X avec des classes y.

Les paramètres:

- **centers** est résponsable à la déterminations de nombre de classes.
- **n_samples** est pour les données
- **n_features** est pour les variables

2. Génération de données

On génère 1000 données de deux variables réparties en 4 groupes

Entrée [ ]:
```
X, y = make_blobs(n_samples=1000, centers=4, n_features=2)
```

Entrée [87]:
```
print("Jeu de donnee")
X
```

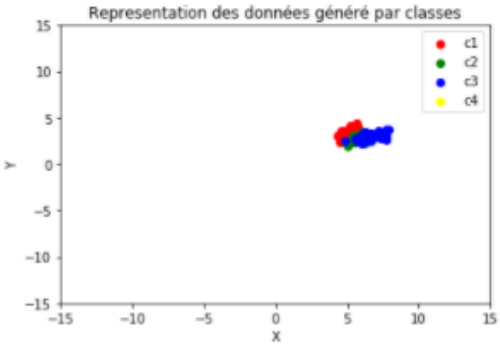
```
Jeu de donnee
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
```

Entrée [86]:
```
y
```

Out[86]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 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, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 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, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

3. Visualisation de données

Entrée [91]:
```
plt.xlim(-15, 15)
plt.ylim(-15, 15)
colors = ["red", "green", "blue", "yellow"]
a = 0
b = 1
classes = ["c1", "c2", "c3", "c4"]
for i in range(4):
    plt.scatter(X[y==i][:,a], X[y==i][:,b], color=colors[i], label=classes[i])
plt.legend()
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Representation des données généré par classes")
plt.show()
```



4. Générer deux jeux de données et les combiner ¶

Entrée [94]:
```
X1, y1 = make_blobs(n_samples=100, centers=2, n_features=2)
X2, y2 = make_blobs(n_samples=500, centers=3, n_features=2)
y = np.hstack((y1, y2))
X = np.vstack((X1, X2))
```

Visualisation de premier jeu de données

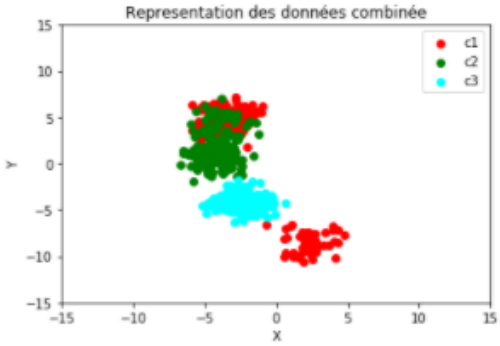
Entrée [95]:
```
plt.xlim(-15, 15)
plt.ylim(-15, 15)
colors = ["red", "green"]
a = 0
b = 1
classes = ["c1", "c2"]
for i in range(2):
    plt.scatter(X1[y1==i][:,a], X1[y1==i][:,b], color=colors[i], label=classes[i])
plt.legend()
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Representation des données généré X1, y1")
plt.show()
```

Visualisation de deuxième jeu de données

Entrée [97]:
```
plt.xlim(-15, 15)
plt.ylim(-15, 15)
colors = ["red", "green", "cyan"]
a = 0
b = 1
classes = ["c1", "c2", "c3"]
for i in range(3):
    plt.scatter(X2[y2==i][:,a], X2[y2==i][:,b], color=colors[i], label=classes[i])
plt.legend()
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Representation des données généré X2, y2")
plt.show()
```

Visualisation de jeu de données combiné

Entrée [99]:
```
plt.xlim(-15, 15)
plt.ylim(-15, 15)
colors = ["red", "green", "cyan"]
a = 0
b = 1
classes = ["c1", "c2", "c3"]
for i in range(3):
    plt.scatter(X[y==i][:,a], X[y==i][:,b], color=colors[i], label=classes[i])
plt.legend()
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Representation des données combinée")
plt.show()
```



D. *Visualisation de données*

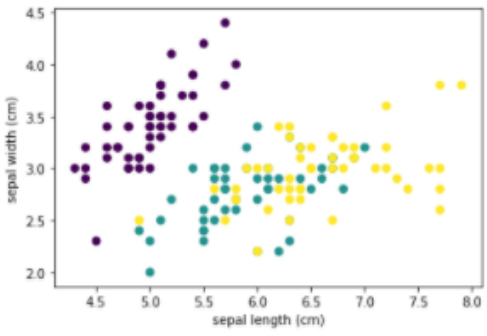
E. Visualiser les données

1. Execution des commandes

Le script suivant trace le graphe des données X en fonction des variable Y.

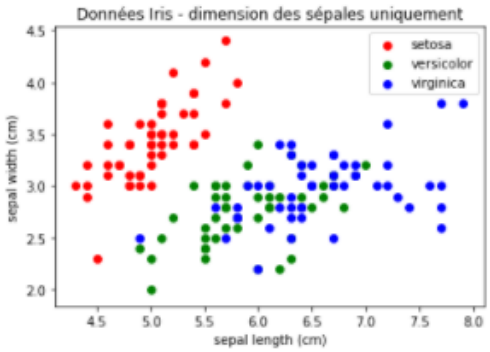
- La methode `pylab.scatter` prend en paramètres X, Y, et c qui désigne les coleurs des classes: Une listes des coleurs ou des codes des coleurs

Entrée [104]:
```
import pylab as pl #permet de remplacer le nom "pylab" par "pl"
X = iris.data
Y = iris.target
a = 0
b= 1
#pl.scatter(X[:, a], X[:, b],c=Y) # les fonctions d'une librairie doivent être préfixées par
# son nom
#pl.show()  affiche la figure
#help(pl.scatter) si on a besoin de savoir le fonctionement de .scatter()
pl.xlabel(iris.feature_names[a])
pl.ylabel(iris.feature_names[b])
pl.scatter(X[:, a], X[:, b],c=Y)
pl.show()
```



2. Autre méthode: on peut écrire le script `TP1prog1.py` contenant le programme suivant

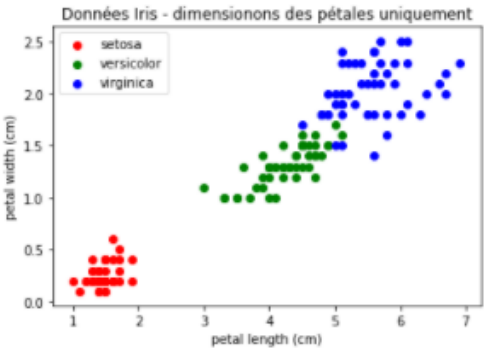
Entrée [105]:
```
import pylab as pl
iris=load_iris()
X=iris.data
y=iris.target
a = 0
b = 1
colors = ["red", "green", "blue"]
for i in range(3):
    plt.scatter(X[y==i][:,a], X[y==i][:,b], color=colors[i], label=iris.target_names[i])
plt.legend()
plt.xlabel(iris.feature_names[a])
plt.ylabel(iris.feature_names[b])
plt.title("Données Iris - dimension des sépales uniquement")
plt.show()
```



On remarque qu'on fixant des couleurs avec une liste et en spécifiant un autre paramètre `label` qui recoit les noms des classes, on peut ajouter une légende pour noter la désignation de chaque couleur.

3. On change les variables et on teste

Entrée [108]:
```
import pylab as pl
iris=load_iris()
X=iris.data
y=iris.target
a = 2 # troisiemes variable
b = 3 # 4ieme variable
colors = ["red", "green", "blue"]
for i in range(3):
    plt.scatter(X[y==i][:,a], X[y==i][:,b], color=colors[i], label=iris.target_names[i])
plt.legend()
plt.xlabel(iris.feature_names[a])
plt.ylabel(iris.feature_names[b])
plt.title("Données Iris - dimensionons des pétales uniquement") # changer le titre
plt.show()
```



On peut maintenant regarder les 4 autres combinations qui restes:

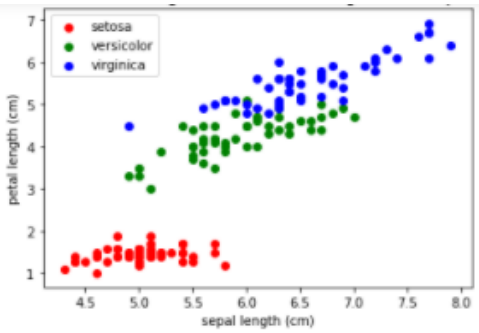
- a = 0, b = 2
- a = 0, b = 3
- a = 1, b = 2
- a = 1, b = 3

On déja tester 2 combination:

- a = 0, b = 1
- a = 2, b = 3

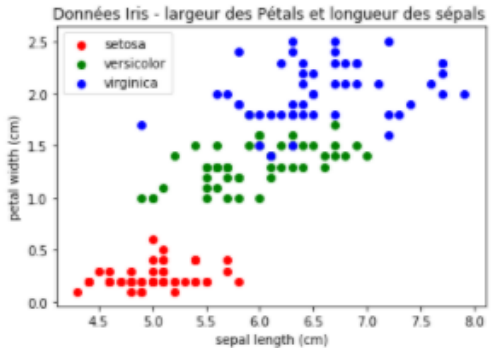
Pour `a = 0, b = 2`

Entrée [111]:
```
import pylab as pl
iris=load_iris()
X=iris.data
y=iris.target
a = 0
b = 2
colors = ["red", "green", "blue"]
for i in range(3):
    plt.scatter(X[y==i][:,a], X[y==i][:,b], color=colors[i], label=iris.target_names[i])
plt.legend()
plt.xlabel(iris.feature_names[a])
plt.ylabel(iris.feature_names[b])
plt.title("Données Iris - longueur des Pétals et longueur des sépals ") # changer le titre
plt.show()
```



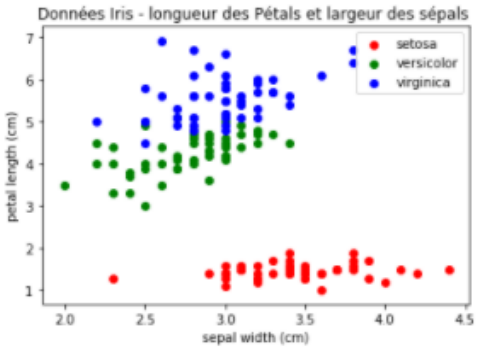
Pour `a = 0, b = 3`

Entrée [113]:
```
import pylab as pl
iris=load_iris()
X=iris.data
y=iris.target
a = 0
b = 3
colors = ["red", "green", "blue"]
for i in range(3):
    plt.scatter(X[y==i][:,a], X[y==i][:,b], color=colors[i], label=iris.target_names[i])
plt.legend()
plt.xlabel(iris.feature_names[a])
plt.ylabel(iris.feature_names[b])
plt.title("Données Iris - largeur des Pétals et longueur des sépals ") # changer le titre
plt.show()
```



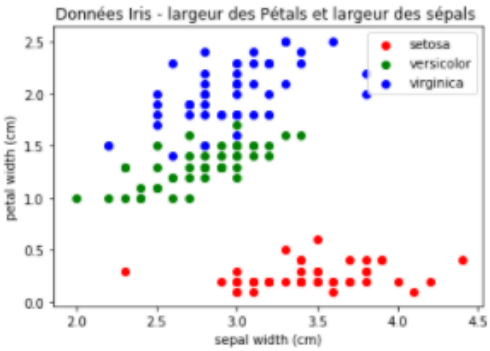
Pour `a = 1, b = 2`

Entrée [115]:
```
import pylab as pl
iris=load_iris()
X=iris.data
y=iris.target
a = 1
b = 2
colors = ["red", "green", "blue"]
for i in range(3):
    plt.scatter(X[y==i][:,a], X[y==i][:,b], color=colors[i], label=iris.target_names[i])
plt.legend()
plt.xlabel(iris.feature_names[a])
plt.ylabel(iris.feature_names[b])
plt.title("Données Iris - longueur des Pétals et largeur des sépals ") # changer le titre
plt.show()
```



Pour `a = 1, b = 3`

Entrée [129]:
```
import pylab as pl
iris=load_iris()
X=iris.data
y=iris.target
a = 1
b = 3
colors = ["red", "green", "blue"]
for i in range(3):
    plt.scatter(X[y==i][:,a], X[y==i][:,b], color=colors[i], label=iris.target_names[i])
plt.legend()
plt.xlabel(iris.feature_names[a])
plt.ylabel(iris.feature_names[b])
plt.title("Données Iris - largeur des Pétals et largeur des sépals ") # changer le titre
plt.show()
```



On peut regrouper les figures pour faire la comparaison! On utilise `pylab.subplot()` ou autre methode!

26

FIN..... 30

INTRODUCTION

➢ Objectif :

Dans ce TP nous allons découvrir le module scikit-learn qui spécialisé en Machine Learning et on va essayer de découvrir et tester ses fonctionnalités !

I. Scikit-learn

Scikit-learn est un logiciel écrit en Python, qui nécessite l'installation préalable du langage Python et des librairies NumPy1 et SciPy2 (pour le calcul scientifique), dans des versions qui doivent vérifier certaines contraintes de compatibilité. Le plus simple est d'installer une distribution de Python complète, comme Anaconda3, qui comprend la plupart des librairies courantes développées en Python, dont les trois citées plus haut. Le site officiel du logiciel Scikit-learn est :

http://scikit-learn.org/stable/index.html

La documentation en ligne est complète, et devra être consultée chaque fois que nécessaire :

http://scikit-learn.org/stable/documentation.html

Des tutoriaux sont disponibles à l'adresse suivante :

http://scikit-learn.org/stable/tutorial/index.html

II. Jeux de données en Scikit-Learn

Un certain nombre de jeux de données sont disponibles dans scikit-learn. Il est également possible de générer des données artificielles ou de récupérer des données externes.

Documentation relative au chargement de jeux de données :

http://scikit-learn.org/stable/datasets/

Les jeux de données disponibles dans scikit-learn sont : iris, boston, diabetes, digits, linnerud, sample images, 20newsgroups. Chacun de ces jeux de données se récupère à l'aide de la commande load_nom-jeu qu'il faut dans un premier temps charger. Par exemple, pour récupérer le jeu iris :

1. Importation d'un jeu de données

```
Entrée [6]: from sklearn.datasets import load_iris
            iris = load_iris()
```

Alors la variable iris contient maintenant notre jeu de données.

III. TP1

Pour bien documenter le travail au fur et à mesure nous avons choisie de travailler avec l'utile jupyter-notebook (IDE).

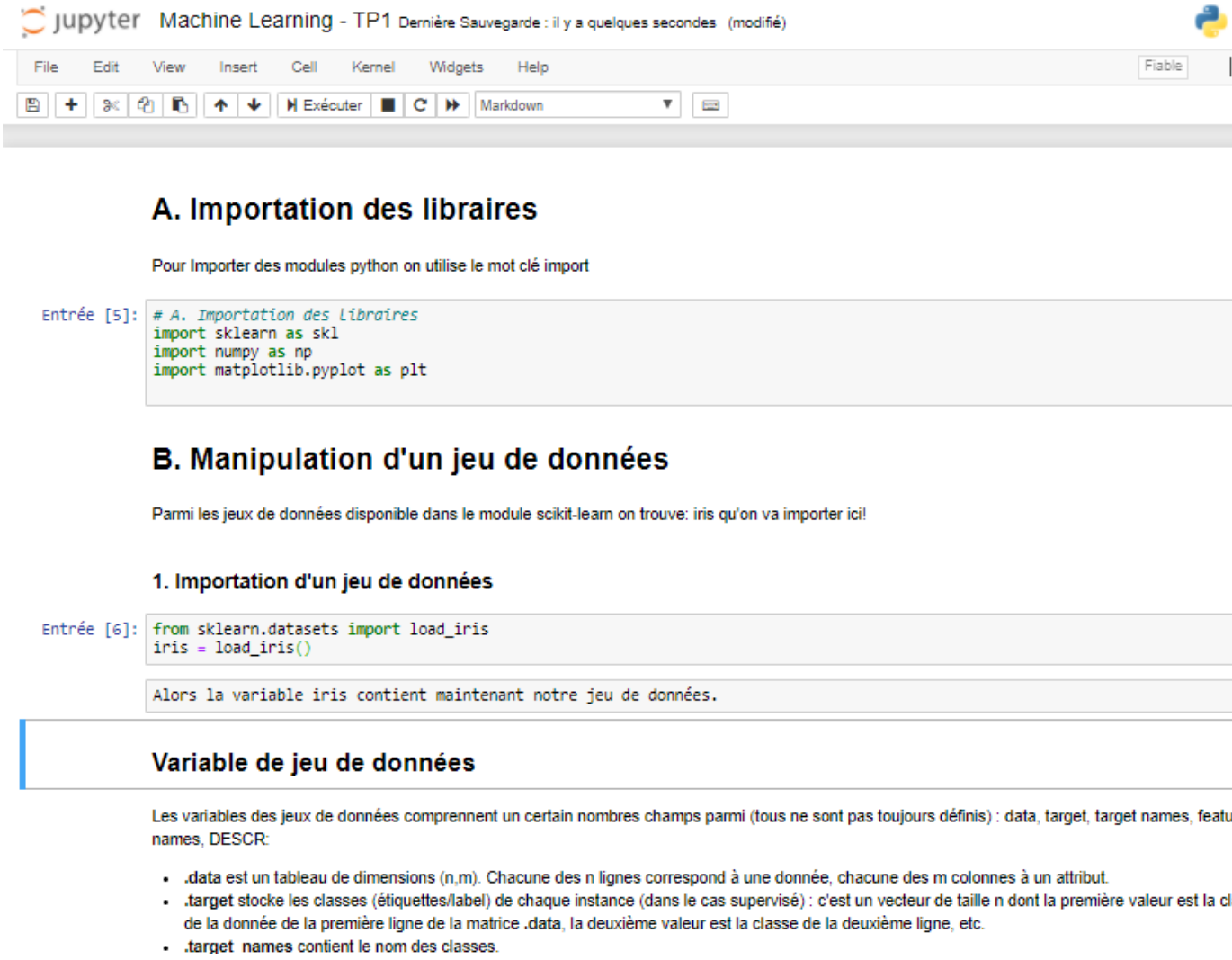
Tout le travail réalisé peut être consulté en ligne sur :

https://notebooks.azure.com/DgrinderHZ/projects/machine-learning/tree/Master%20MST%20SIDI%20-%20FST%20Errachidia

il suffit de l'ouvrir sur Azur Microsoft ou la télécharger pour Anaconda Jupyter.

IV. Jupyter notebook | TP1

A. B. Importation des libraires et manipulation d'un jeu de données



- **.target_names** contient le nom des classes.
- **.feature_names** contient le nom des attributs.
- **.DESCR** est un texte décrivant le jeu de données.

2. Afficher les données, noms des variables, le nom des classes

Entrée [7]:
```
# Afficher .data
print(iris.data)
```

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
 [5.1 3.8 1.5 0.3]
 [5.4 3.4 1.7 0.2]
```

Entrée [8]:
```
# Affichage des classes de chaque instance
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 0 0 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 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 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 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

Entrée [9]:
```
# Affichage de noms des classes
print(iris.target_names)
```

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

Entrée [10]:
```
# Affichage de noms des attributs/variables
print(iris.feature_names)
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

3. Nom des classes de chaque donnée

On peut répondre au question par un simple code python:

Entrée [11]:
```
print("Numéro de Donnée | Nom de sa classe")
for i in range(len(iris.data)):
    print(i ,"\t\t  ", iris.target_names[iris.target[i]])
```

```
Numéro de Donnée | Nom de sa classe
0                 setosa
1                 setosa
2                 setosa
3                 setosa
4                 setosa
5                 setosa
6                 setosa
7                 setosa
8                 setosa
9                 setosa
10                setosa
11                setosa
12                setosa
13                setosa
14                setosa
15                setosa
16                setosa
17                setosa
18                setosa
19                setosa
```

Les methodes et attributs

Entrée [12]:
```
## les methodes: .data par exemple
### Calculer la moyenne de chaque variable(axe = 0, colonne)
print(iris.data.mean(0))
```

```
[5.84333333 3.05733333 3.758      1.19933333]
```

Calculer la moyenne de chaque donnée(axe = 1, ligne)

Entrée [13]:
```
print(iris.data.mean(1))
```

```
[2.55  2.375 2.35  2.35  2.55  2.85  2.425 2.525 2.225 2.4   2.7   2.5
 2.325 2.125 2.8   3.    2.75  2.575 2.875 2.675 2.675 2.675 2.35  2.65
 2.575 2.45  2.6   2.6   2.55  2.425 2.425 2.675 2.725 2.825 2.425 2.4
 2.625 2.5   2.225 2.55  2.525 2.1   2.275 2.675 2.8   2.375 2.675 2.35
 2.675 2.475 4.075 3.9   4.1   3.275 3.85  3.575 3.975 2.9   3.85  3.3
 2.875 3.65  3.3   3.775 3.35  3.9   3.65  3.4   3.6   3.275 3.925 3.55
 3.8   3.7   3.725 3.85  3.95  4.1   3.725 3.2   3.2   3.15  3.4   3.85
 3.6   3.875 4.    3.575 3.5   3.325 3.425 3.775 3.4   2.9   3.45  3.525
 3.525 3.675 2.925 3.475 4.525 3.875 4.525 4.15  4.375 4.825 3.4   4.575
 4.2   4.85  4.2   4.075 4.35  3.8   4.025 4.3   4.2   5.1   4.875 3.675
 4.525 3.825 4.8   3.925 4.45  4.55  3.9   3.95  4.225 4.4   4.55  5.025
 4.25  3.925 3.925 4.775 4.425 4.2   3.9   4.375 4.45  4.35  3.875 4.55
 4.55  4.3   3.925 4.175 4.325 3.95 ]
```

Calculer le nombre d'objets dans target

Entrée [14]:
```
## les attributs: .target par exemple
### Calculer le nombre d'objets dans target
print(iris.target.size)
```

```
150
```

La fonction `.len()` retourne la longueur de l'objet passé en paramètre. ¶

Entrée [15]:
```
print(len(iris.data)) # >>> affiche 150 données comme lenght
```

```
150
```

La fonction `help()` affiche de l'aide à propos la fonction/methode/module passé en paramètre.

Entrée [16]:
```
help(len)
```

```
Help on built-in function len in module builtins:

len(obj, /)
    Return the number of items in a container.
```

L'instruction `iris.target_names[2]` affiche le nom de la troixième (dernière) classe.

Entrée [17]:
```
print(iris.target_names[2])
```

```
virginica
```

L'instruction `iris.target_names[-1]` affiche le nom de la dernière classe.

Entrée [18]:
```
print(iris.target_names[-1])
```

```
virginica
```

L'instruction `iris.target_names[len(iris.target_names)]` *affiche erreur: index out of range.*

Entrée [15]:
```
print(iris.target_names[len(iris.target_names)])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-15-1e1c5c3b5543> in <module>
----> 1 print(iris.target_names[len(iris.target_names)])

IndexError: index 3 is out of bounds for axis 0 with size 3
```

L'instruction `iris.data[0][1]` *affiche la valeur de la deuxième variable pour la première donnée.*

Entrée [61]:
```
print(iris.data[0][1])
```

```
3.5
```

L'instruction `iris.data[:,1]` *affiche la valeur de la deuxième variable pour tous les données.*

Entrée [62]:
```
print(iris.data[:,1])
```

```
[3.5 3.  3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 3.7 3.4 3.  3.  4.  4.4 3.9 3.5
 3.8 3.8 3.4 3.7 3.6 3.3 3.4 3.  3.4 3.5 3.4 3.2 3.1 3.4 4.1 4.2 3.1 3.2
 3.5 3.6 3.  3.4 3.5 2.3 3.2 3.5 3.8 3.  3.8 3.2 3.7 3.3 3.2 3.2 3.1 2.3
 2.8 2.8 3.3 2.4 2.9 2.7 2.  3.  2.2 2.9 2.9 3.1 3.  2.7 2.2 2.5 3.2 2.8
 2.5 2.8 2.9 3.  2.8 3.  2.9 2.6 2.4 2.4 2.7 2.7 3.  3.4 3.1 2.3 3.  2.5
 2.6 3.  2.6 2.3 2.7 3.  2.9 2.9 2.5 2.8 3.3 2.7 3.  2.9 3.  3.  2.5 2.9
 2.5 3.6 3.2 2.7 3.  2.5 2.8 3.2 3.  3.8 2.6 2.2 3.2 2.8 2.8 2.7 3.3 3.2
 2.8 3.  2.8 3.  2.8 3.8 2.8 2.8 2.6 3.  3.4 3.1 3.  3.1 3.1 3.1 2.7 3.2
 3.3 3.  2.5 3.  3.4 3. ]
```

4. Afficher: moyenne, ecart-type, min et le max pour chaque variable => axe = 0

Entrée [63]:
```
print("Les variables:")
print(iris.feature_names)
```

```
Les variables:
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

L'instruction `iris.data.mean(0)` *retourne la moyenne.*

Entrée [64]:
```
print(iris.data.mean(0))
```

```
[5.84333333 3.05733333 3.758      1.19933333]
```

L'instruction `iris.data.std(0)` *retourne l'ecart-type.*

Entrée [65]:
```
print(iris.data.std(0))
```

```
[0.82530129 0.43441097 1.75940407 0.75969263]
```

L'instruction `iris.data.min(0)` *retourne le minimum.*

Entrée [66]:
```
print(iris.data.min(0))
```

```
[4.3 2.  1.  0.1]
```

L'instruction `iris.data.maxn(0)` *retourne le maximum.*

Entrée [67]:
```
print(iris.data.max(0))
```

```
[7.9 4.4 6.9 2.5]
```

C. Téléchargement et Importation de données

5. Afficher: nombre de données, nombre de variables et le nombre de classes

Entrée [68]:
```
print("Le nombre de données est:", iris.data.shape[0])
print("Le nombre de variables est:", iris.data.shape[1])
print("Le nombre de classes est:", iris.target_names.size)
```

```
Le nombre de données est: 150
Le nombre de variables est: 4
Le nombre de classes est: 3
```

C. Téléchargement et importation de données

Entrée [69]:
```
from sklearn.datasets import fetch_mldata
mnist = fetch_mldata('MNIST original') # on peut utiliser les données télécharger aussi
```

```
C:\Users\elitebook\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:85: DeprecationWarning: Function fetch_mldata is deprecated; fetch_mldata was deprecated in version 0.20 and will be removed in version 0.22. Please use fetch_openml.
  warnings.warn(msg, category=DeprecationWarning)
C:\Users\elitebook\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:85: DeprecationWarning: Function mldata_filename is deprecated; mldata_filename was deprecated in version 0.20 and will be removed in version 0.22. Please use fetch_openml.
  warnings.warn(msg, category=DeprecationWarning)

---------------------------------------------------------------------------
TimeoutError                              Traceback (most recent call last)
~\Anaconda3\lib\urllib\request.py in do_open(self, http_class, req, **http_conn_args)
   1316                 h.request(req.get_method(), req.selector, req.data, headers,
-> 1317                           encode_chunked=req.has_header('Transfer-encoding'))
   1318             except OSError as err: # timeout error

~\Anaconda3\lib\http\client.py in request(self, method, url, body, headers, encode_chunked)
   1243         """Send a complete request to the server."""
-> 1244         self._send_request(method, url, body, headers, encode_chunked)
   1245

~\Anaconda3\lib\http\client.py in _send_request(self, method, url, body, headers, encode_chunked)
   1289             body = _encode(body, 'body')
```

On remarque que Python demande d'utiliser .fetch_openml qui a remplacer .fetch_mldata

Entrée [70]:
```
# Importation d'un jeu de données
from sklearn.datasets import fetch_openml
mnist = fetch_openml('MNIST original') # donne invalidURL, on charge les données téléchargées ou changer en 'mnist_784'
```

```
---------------------------------------------------------------------------
InvalidURL                                Traceback (most recent call last)
~\Anaconda3\lib\site-packages\sklearn\datasets\openml.py in wrapper()
     46             try:
---> 47                 return f()
     48             except HTTPError:

~\Anaconda3\lib\site-packages\sklearn\datasets\openml.py in _load_json()
    150     def _load_json():
--> 151         with closing(_open_openml_url(url, data_home)) as response:
    152             return json.loads(response.read().decode("utf-8"))

~\Anaconda3\lib\site-packages\sklearn\datasets\openml.py in _open_openml_url(openml_path, data_home)
     99         try:
--> 100             with closing(urlopen(req)) as fsrc:
    101                 if is_gzip(fsrc):

~\Anaconda3\lib\urllib\request.py in urlopen(url, data, timeout, cafile, capath, cadefault, context)
    221         opener = _opener
--> 222     return opener.open(url, data, timeout)
```

Pour résoude ce pronlème on passe la valeur 'mnist_784'

Entrée [71]:
```
# Importation d'un jeu de données
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784')
```

Afficher la matrice de données

Entrée [72]:
```
print("Data:\n", mnist.data)
```

```
Data:
 [[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

Afficher le nombres de données et variables

Entrée [73]:
```
donnee, variable = mnist.data.shape # ou np.shape(mnist.data)
print("Le nombre de donnees:", donnee)
print("Le nombre de variables:", variable)
```

```
Le nombre de donnees: 70000
Le nombre de variables: 784
```

Afficher le nombres de données et variables

Entrée [*]:
```
print("Numéro de Donnée | Numéro de sa classe")
for i in range(len(mnist.data)):
    print(i ,"\t\t  ", mnist.target[i])
```

```
Numéro de Donnée | Numéro de sa classe
0                  5
1                  0
2                  4
3                  1
4                  9
5                  2
6                  1
7                  3
8                  1
9                  4
10                 3
```

Entrée [75]:
```
print("Les variables:")
print(mnist.feature_names)
```

```
Les variables:
['pixel1', 'pixel2', 'pixel3', 'pixel4', 'pixel5', 'pixel6', 'pixel7', 'pixel8', 'pixel9', 'pixel10', 'pixel11', 'pixel12', 'pi
xel13', 'pixel14', 'pixel15', 'pixel16', 'pixel17', 'pixel18', 'pixel19', 'pixel20', 'pixel21', 'pixel22', 'pixel23', 'pixel2
4', 'pixel25', 'pixel26', 'pixel27', 'pixel28', 'pixel29', 'pixel30', 'pixel31', 'pixel32', 'pixel33', 'pixel34', 'pixel35', 'p
ixel36', 'pixel37', 'pixel38', 'pixel39', 'pixel40', 'pixel41', 'pixel42', 'pixel43', 'pixel44', 'pixel45', 'pixel46', 'pixel4
```

Afficher l'écart-type des variables

Entrée [77]:
```
print(mnist.data.std(0)) # 784 variables
```

```
[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 4.40061249e-01 1.26020885e+00 8.16397430e-01 3.40165596e-02
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
```

Afficher le min des variables

Entrée [78]:
```
print(mnist.data.min(0)) # 784 variables
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

Afficher le max des variables

Entrée [79]:
```
print(mnist.data.max(0)) # 784 variables
```

```
[  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0. 116. 254.
 216.   9.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.  16.  47. 236. 254. 255. 254. 255. 255. 255. 255. 255.
 255. 255. 255. 255. 255. 255. 244. 255. 184. 197.   0.   0.   0.   0.
   0.   0.  64.  29. 134. 192. 254. 255. 255. 255. 255. 255. 255. 255.
 255. 255. 255. 255. 255. 255. 255. 255. 255. 255. 255. 165.   0.   0.
   0.   0. 141. 101. 139. 255. 255. 255. 255. 255. 255. 255. 255. 255.
 255. 255. 255. 255. 255. 255. 255. 255. 255. 255. 255. 234. 163.   0.
```

Nombre de classe avec la fonction `numpy.unique()`

Entrée [80]:
```
print("Les classes: ", np.unique(mnist.target))
print("Le nombre des classes: ", len(np.unique(mnist.target)))
```

```
Les classes:  ['0' '1' '2' '3' '4' '5' '6' '7' '8' '9']
Le nombre des classes:  10
```

C. Génération de données et affichage

D. Génération de données et affichage

1. La fonction `help()`

Entrée [81]:
```
from sklearn.datasets import make_blobs
help(make_blobs)
```

```
Help on function make_blobs in module sklearn.datasets.samples_generator:

make_blobs(n_samples=100, n_features=2, centers=None, cluster_std=1.0, center_box=(-10.0, 10.0), shuffle=True, random_state=
None)
    Generate isotropic Gaussian blobs for clustering.

    Read more in the :ref:`User Guide <sample_generators>`.

    Parameters
    ----------
    n_samples : int or array-like, optional (default=100)
        If int, it is the total number of points equally divided among
        clusters.
        If array-like, each element of the sequence indicates
        the number of samples per cluster.

    n_features : int, optional (default=2)
        The number of features for each sample.

    centers : int or array of shape [n_centers, n_features], optional
        (default=None)
```

On peut compredre que la methode génére:

- un jeu de données X avec des classes y.

Les paramètres:

- **centers** est résponsable à la déterminations de nombre de classes.
- **n_samples** est pour les données
- **n_features** est pour les variables

2. Génération de données

On génére 1000 données de deux variables réparties en 4 groupes

Entrée [ ]:
```
X, y = make_blobs(n_samples=1000, centers=4, n_features=2)
```

Entrée [87]:
```
print("Jeu de donnee")
X
```

```
Jeu de donnee
```

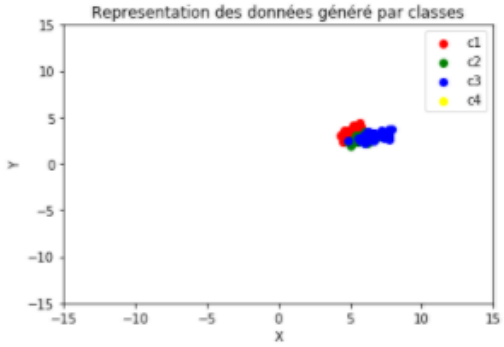
Out[87]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
```

Entrée [86]:
```
y
```

Out[86]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 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, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 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, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

3. Visualisation de données

Entrée [91]:
```
plt.xlim(-15, 15)
plt.ylim(-15, 15)
colors = ["red", "green", "blue", "yellow"]
a = 0
b = 1
classes = ["c1", "c2", "c3", "c4"]
for i in range(4):
    plt.scatter(X[y==i][:,a], X[y==i][:,b], color=colors[i], label=classes[i])
plt.legend()
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Representation des données généré par classes")
plt.show()
```



4. Générer deux jeux de données et les combiner ¶

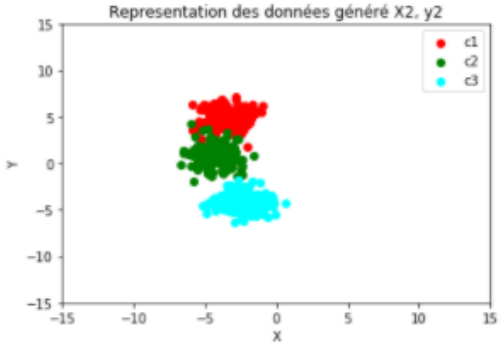
Entrée [94]:
```
X1, y1 = make_blobs(n_samples=100, centers=2, n_features=2)
X2, y2 = make_blobs(n_samples=500, centers=3, n_features=2)
y = np.hstack((y1, y2))
X = np.vstack((X1, X2))
```

Visualisation de premier jeu de données

Entrée [95]:
```
plt.xlim(-15, 15)
plt.ylim(-15, 15)
colors = ["red", "green"]
a = 0
b = 1
classes = ["c1", "c2"]
for i in range(2):
    plt.scatter(X1[y1==i][:,a], X1[y1==i][:,b], color=colors[i], label=classes[i])
plt.legend()
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Representation des données généré X1, y1")
plt.show()
```

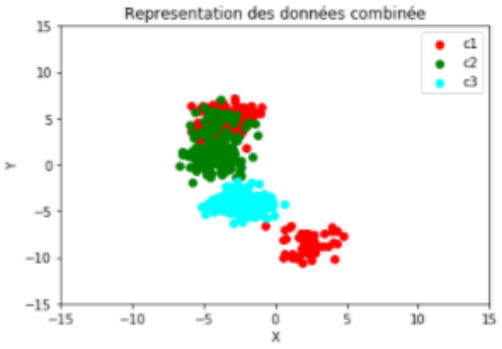
Visualisation de deuxième jeu de données

Entrée [97]:
```
plt.xlim(-15, 15)
plt.ylim(-15, 15)
colors = ["red", "green", "cyan"]
a = 0
b = 1
classes = ["c1", "c2", "c3"]
for i in range(3):
    plt.scatter(X2[y2==i][:,a], X2[y2==i][:,b], color=colors[i], label=classes[i])
plt.legend()
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Representation des données généré X2, y2")
plt.show()
```



Visualisation de jeu de données combiné

Entrée [99]:
```
plt.xlim(-15, 15)
plt.ylim(-15, 15)
colors = ["red", "green", "cyan"]
a = 0
b = 1
classes = ["c1", "c2", "c3"]
for i in range(3):
    plt.scatter(X[y==i][:,a], X[y==i][:,b], color=colors[i], label=classes[i])
plt.legend()
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Representation des données combinée")
plt.show()
```



D. Visualisation de données

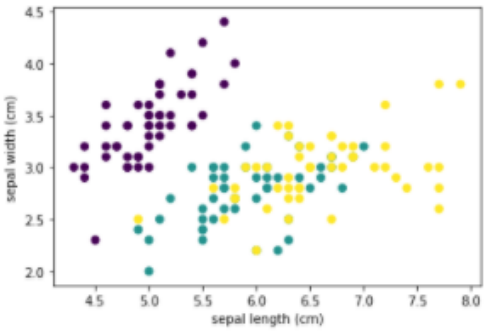
E. Visualiser les données

1. Execution des commandes

Le script suivant trace le graphe des données X en fonction des variable Y.

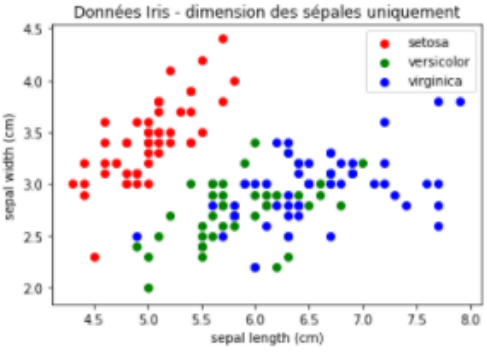
- La methode `pylab.scatter` prend en paramètres X, Y, et c qui désigne les coleurs des classes: Une listes des coleurs ou des codes des coleurs

Entrée [104]:
```
import pylab as pl #permet de remplacer le nom "pylab" par "pl"
X = iris.data
Y = iris.target
a = 0
b= 1
#pl.scatter(X[:, a], X[:, b],c=Y) # les fonctions d'une librairie doivent être préfixées par
# son nom
#pl.show()  affiche la figure
#help(pl.scatter) si on a besoin de savoir le fonctionement de .scatter()
pl.xlabel(iris.feature_names[a])
pl.ylabel(iris.feature_names[b])
pl.scatter(X[:, a], X[:, b],c=Y)
pl.show()
```



2. Autre méthode: on peut écrire le script `TP1prog1.py` contenant le programme suivant

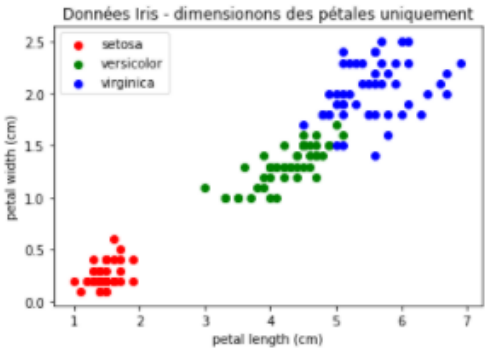
Entrée [105]:
```
import pylab as pl
iris=load_iris()
X=iris.data
y=iris.target
a = 0
b = 1
colors = ["red", "green", "blue"]
for i in range(3):
    plt.scatter(X[y==i][:,a], X[y==i][:,b], color=colors[i], label=iris.target_names[i])
plt.legend()
plt.xlabel(iris.feature_names[a])
plt.ylabel(iris.feature_names[b])
plt.title("Données Iris - dimension des sépales uniquement")
plt.show()
```



On remarque qu'on fixant des couleurs avec une liste et en spécifiant un autre paramètre `label` qui recoit les noms des classes, on peut ajouter une légende pour noter la désignation de chaque couleur.

3. On change les variables et on teste

Entrée [108]:
```
import pylab as pl
iris=load_iris()
X=iris.data
y=iris.target
a = 2 # troisiemes variable
b = 3 # 4ieme variable
colors = ["red", "green", "blue"]
for i in range(3):
    plt.scatter(X[y==i][:,a], X[y==i][:,b], color=colors[i], label=iris.target_names[i])
plt.legend()
plt.xlabel(iris.feature_names[a])
plt.ylabel(iris.feature_names[b])
plt.title("Données Iris - dimensionons des pétales uniquement") # changer le titre
plt.show()
```



On peut maintenant regarder les 4 autres combinations qui restes:

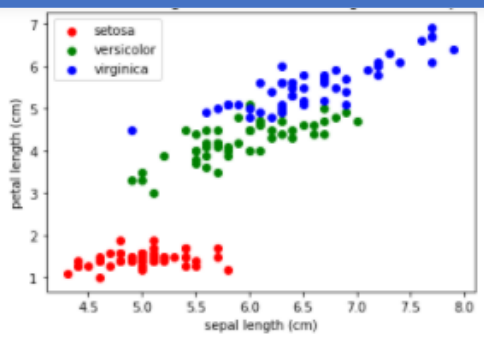
- a = 0, b = 2
- a = 0, b = 3
- a = 1, b = 2
- a = 1, b = 3

On déja tester 2 combination:

- a = 0, b = 1
- a = 2, b = 3

Pour `a = 0, b = 2`

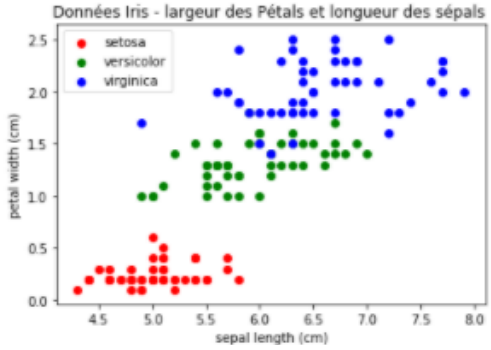
Entrée [111]:
```
import pylab as pl
iris=load_iris()
X=iris.data
y=iris.target
a = 0
b = 2
colors = ["red", "green", "blue"]
for i in range(3):
    plt.scatter(X[y==i][:,a], X[y==i][:,b], color=colors[i], label=iris.target_names[i])
plt.legend()
plt.xlabel(iris.feature_names[a])
plt.ylabel(iris.feature_names[b])
plt.title("Données Iris - longueur des Pétals et longueur des sépals ") # changer le titre
plt.show()
```



Pour `a = 0, b = 3`

Entrée [113]:

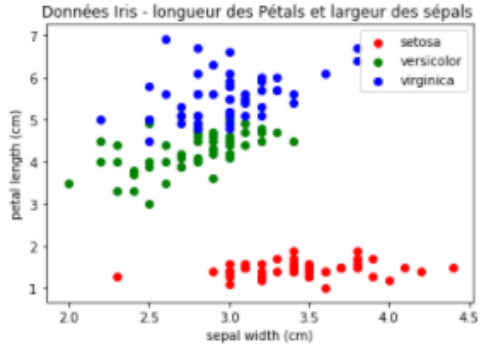
```
import pylab as pl
iris=load_iris()
X=iris.data
y=iris.target
a = 0
b = 3
colors = ["red", "green", "blue"]
for i in range(3):
    plt.scatter(X[y==i][:,a], X[y==i][:,b], color=colors[i], label=iris.target_names[i])
plt.legend()
plt.xlabel(iris.feature_names[a])
plt.ylabel(iris.feature_names[b])
plt.title("Données Iris - largeur des Pétals et longueur des sépals ") # changer le titre
plt.show()
```



Pour `a = 1, b = 2`

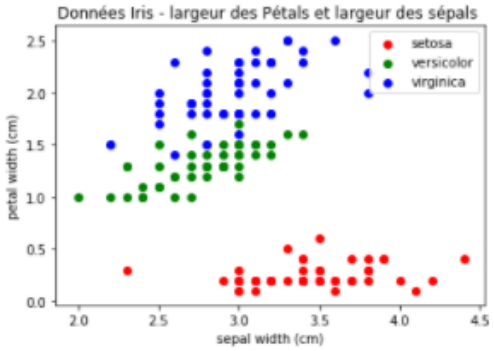
Entrée [115]:

```
import pylab as pl
iris=load_iris()
X=iris.data
y=iris.target
a = 1
b = 2
colors = ["red", "green", "blue"]
for i in range(3):
    plt.scatter(X[y==i][:,a], X[y==i][:,b], color=colors[i], label=iris.target_names[i])
plt.legend()
plt.xlabel(iris.feature_names[a])
plt.ylabel(iris.feature_names[b])
plt.title("Données Iris - longueur des Pétals et largeur des sépals ") # changer le titre
plt.show()
```



Pour `a = 1, b = 3`

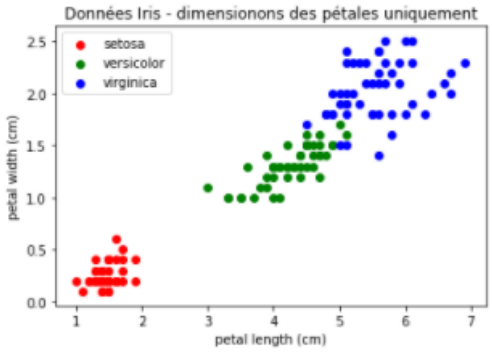
Entrée [129]:
```
import pylab as pl
iris=load_iris()
X=iris.data
y=iris.target
a = 1
b = 3
colors = ["red", "green", "blue"]
for i in range(3):
    plt.scatter(X[y==i][:,a], X[y==i][:,b], color=colors[i], label=iris.target_names[i])
plt.legend()
plt.xlabel(iris.feature_names[a])
plt.ylabel(iris.feature_names[b])
plt.title("Données Iris - largeur des Pétals et largeur des sépals ") # changer le titre
plt.show()
```



On peut regrouper les figures pour faire la comparaison! On utilise `pylab.subplot()` ou autre methode!

On peut alors constater que le couples d'attributs qui semble le mieux à discriminer les 3 classes d'iris et le suivant :

Entrée [130]:
```
import pylab as pl
iris=load_iris()
X=iris.data
y=iris.target
a = 2 # troisiemes variable
b = 3 # 4ieme variable
colors = ["red", "green", "blue"]
for i in range(3):
    plt.scatter(X[y==i][:,a], X[y==i][:,b], color=colors[i], label=iris.target_names[i])
plt.legend()
plt.xlabel(iris.feature_names[a])
plt.ylabel(iris.feature_names[b])
plt.title("Données Iris - dimensionons des pétales uniquement") # changer le titre
plt.show()
```



FIN

Merci