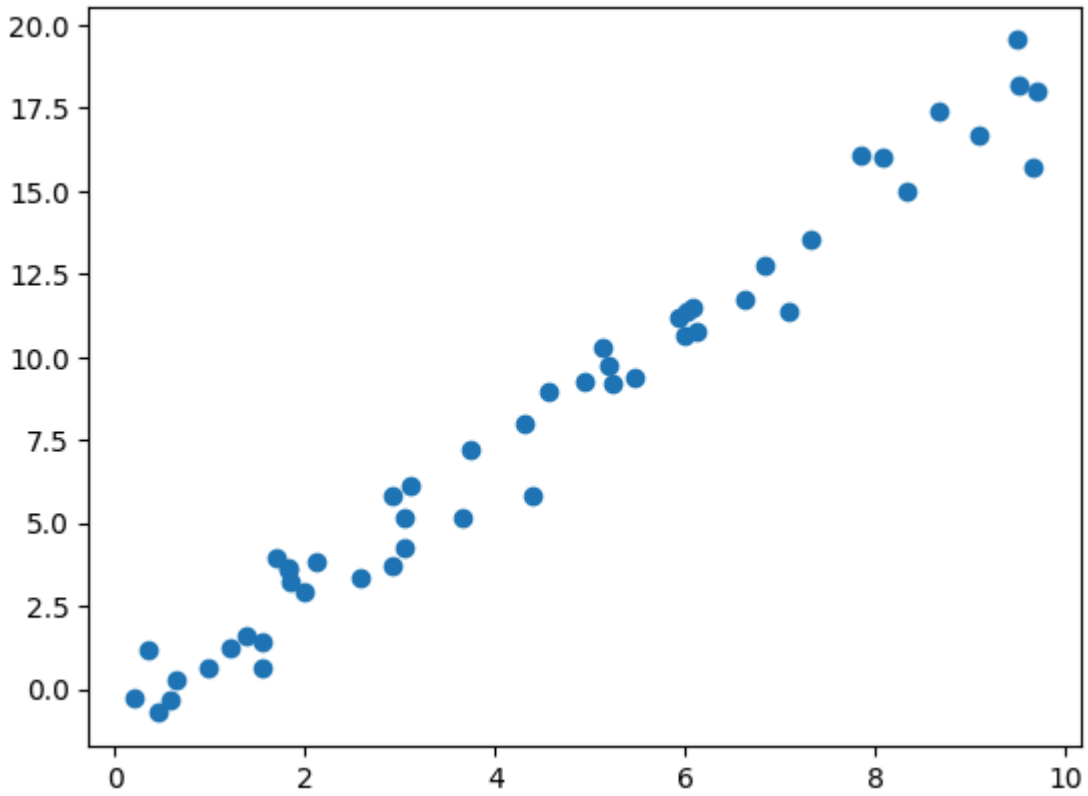
In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
import sklearn as sk
```

I- Régression

In [3]:
```
#un exemple simple en deux dimensions

rng = np.random.RandomState(42) #pour générer les mêmes données

#constituer  un exmple de data
x = 10 * rng.rand(50)
print('la taille de notre échantillon est :',x.shape)

y=2*x-1 + rng.randn(50)
#afficher data y=f(x) [y en fonction de x]  comme un nuage de points
plt.scatter(x, y);
```

```
la taille de notre échantillon est : (50,)
```



Y-a-t-il une relation entre x et y : trouver f tel que y=f(x) ? (Cours 1)

Pour répondre à cette question, nous allons supposer que f est une fonction de la forme $f(x) = a * x + b$ avec a et b sont des réels à déterminer.

Input : (x_i, y_i), pour $i = 1 \dots 50$

on a $y_i = a * x_i + b$, pour $i = 1 \dots 50$

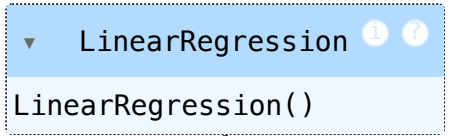
qui forme un système linéaire facile à résoudre (plus de données que d'inconnus)

Formulation (cours semaine 1)

In [4]:
```
# On peut résoudre ce problème de régression linéaire avec sklearn

# on choisit et charge le modèle
from sklearn.linear_model import LinearRegression

X = x[:, np.newaxis]
print('la tailles des entrées est :',X.shape)

models = LinearRegression(fit_intercept=True)
models.fit(X, y)
```

```
la tailles des entrées est : (50, 1)
```

Out[4]:



In [5]:
```
a=models.coef_
print('-'*5,'la solution','-'*5)
print('la valeur trouvée de a est : ', a[0])

b=models.intercept_
print('la valeur trouvée de b est : ', b)
```

```
----- la solution -----
la valeur trouvée de a est :  1.9776566003853104
la valeur trouvée de b est :  -0.9033107255311146
```

Si maintenat on a un nouveau $xnew = 2.5$ qui est différent de tous les x_i observés

On peut trouver son image $ynew$ avec $ynew = a * xnew + b$

In [6]:
```
xnew=np.array([2.50])
ynew = models.predict(xnew.reshape(-1, 1))
print(ynew)
```

```
[4.04083078]
```

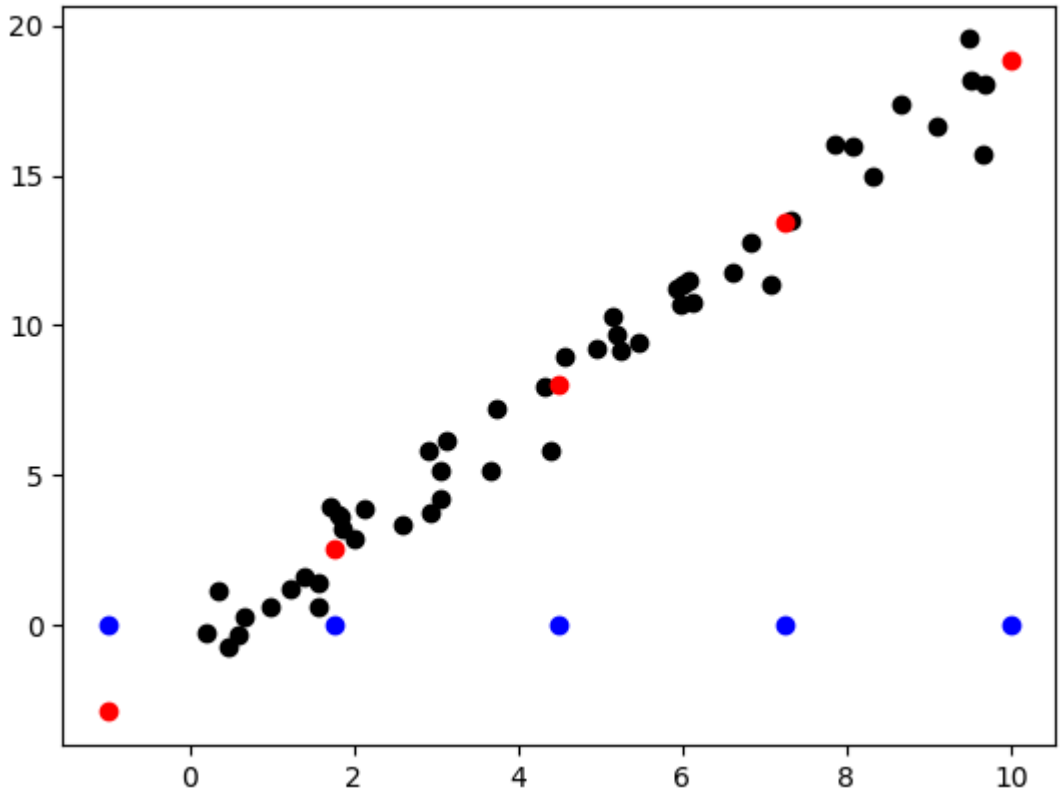
On peut aussi appliquer la même méthode sur xnew comme tableau de valeurs au lieu d'un seul scalaire

In [7]:
```
xnew=np.linspace(-1,10,5)
#s'assurer d'avoir le bon format
xnew=xnew[:, np.newaxis]

ynew = models.predict(xnew)
print(ynew)
```

```
[-2.88096733  2.55758833  7.99614398 13.43469963 18.87325528]
```

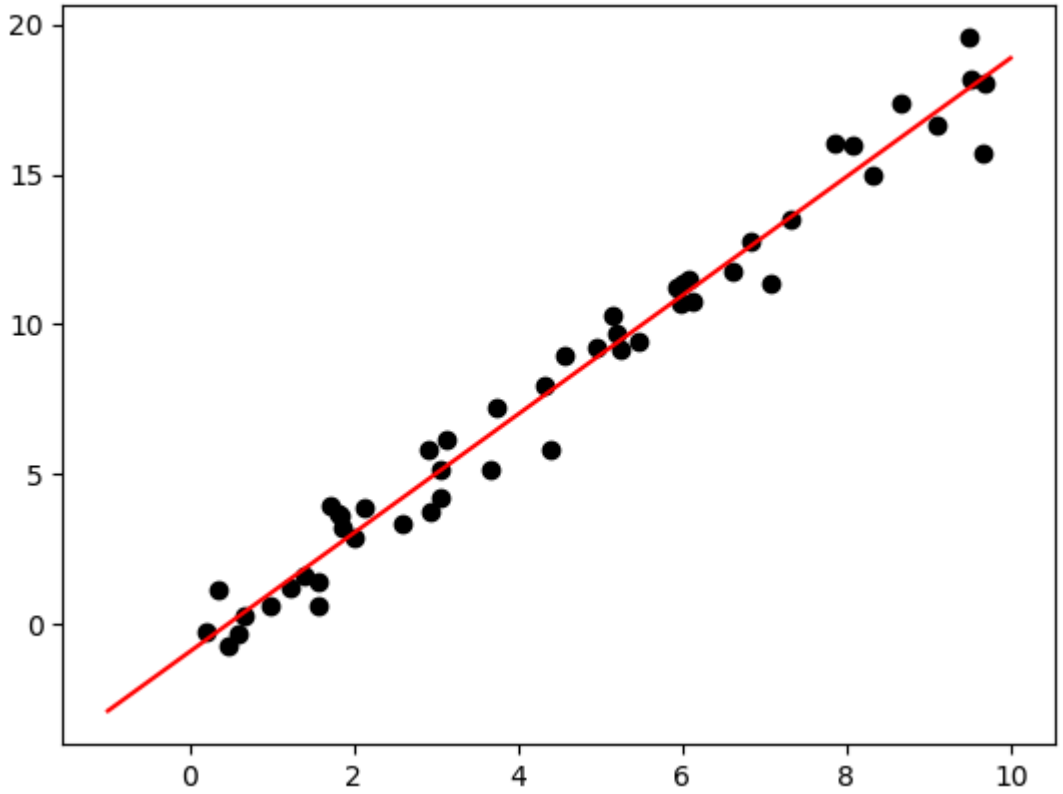
Vérification visuelle

In [8]:
```
plt.scatter(x, y,color='k');# données apprentissage en noir
plt.scatter(xnew, np.zeros(xnew.shape[0]),color='b');# x_i non observés en bleu
plt.scatter(xnew, ynew,color='r');# y_i prédit ave la régression linéaire (x_i,y_i) en rouge
```



In [9]:
```
#on peut aussi afficher la fonction f
plt.scatter(x, y,color='k');
#plt.scatter(xnew, ynew);
plt.plot(xnew, ynew,'r');
#l'erreur est donnée par la somme cumulée des distances
#entre les points en noir et la droite en rouge

ypred=models.predict(X)
print(ypred.shape)
print('Erreur quadratique moyenne : ',np.mean((y-ypred)**2))
```

```
(50,)
Erreur quadratique moyenne :  0.8230711437486875
```

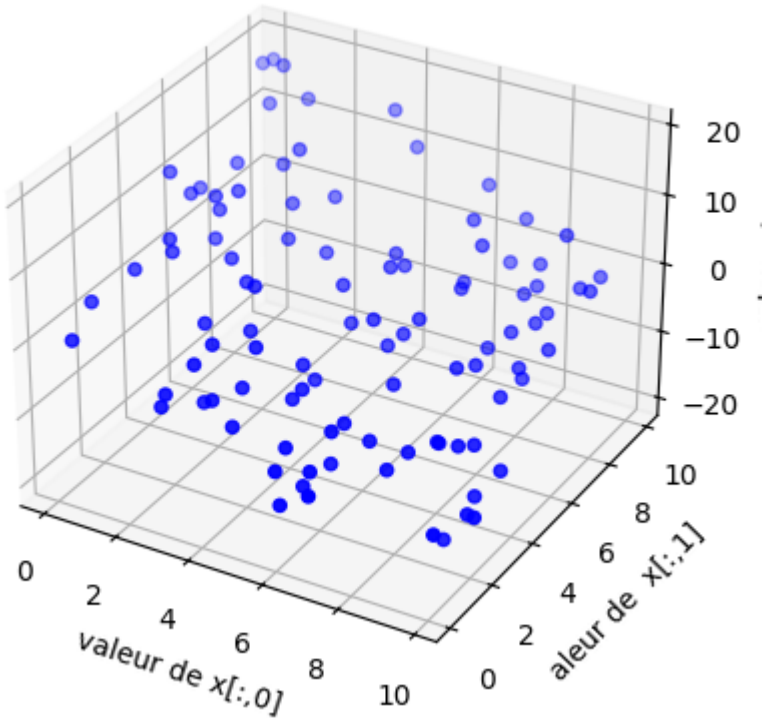


Nous venons de faire notre premier exemple pour le cas simple $x \in \mathbb{R}$ et $y \in \mathbb{R}$

On peut généraliser ce résultat quelque soit la taille de x : $x \in \mathbb{R}^d$, pour toute dimension d

Exemple :

In [10]:
```
from mpl_toolkits.mplot3d import Axes3D
#constituer  un exmple de data
x = np.array(10 * rng.rand(100,2))
y=2*np.inner(np.array([-1,1]), x)+ 2*rng.randn(x.shape[0])

fig=plt.figure()
ax = fig.add_subplot(111, projection='3d')

ax.scatter(x[:,0], x[:,1],y,c='b', marker='o');
ax.set_xlabel('valeur de x[:,0]')
ax.set_ylabel('aleur de  x[:,1]')
ax.set_zlabel('valeur de y ')

plt.show()
```



In [11]:
```
model = LinearRegression(fit_intercept=True)
model.fit(x, y)
```

Out[11]:

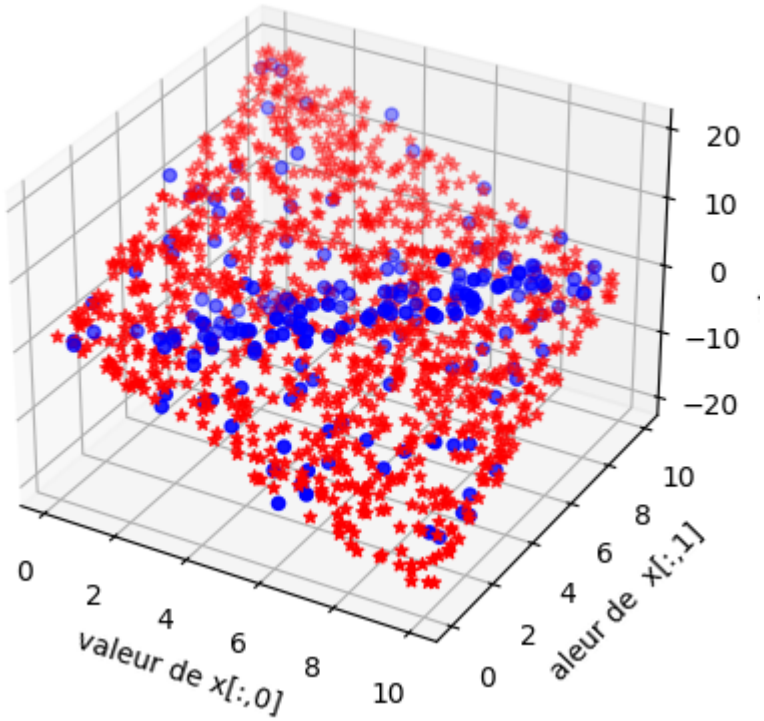
LinearRegression

LinearRegression()

In [12]:
```
xnew = np.array(10 * rng.rand(1000,2))
ynew = model.predict(xnew)
```

In [13]:
```
fig=plt.figure()
ax = fig.add_subplot(111, projection='3d')

ax.scatter(x[:,0], x[:,1],y,c='b', marker='o');
```

```
ax.scatter(x[:,0], x[:,1],-y,c='b', marker='o');
ax.set_xlabel('valeur de x[:,0]')
ax.set_ylabel('aleur de  x[:,1]')
ax.set_zlabel('valeur de y ')

ax.scatter(xnew[:,0], xnew[:,1],ynew,c='r', marker='*');

plt.show()
```



In [14]:
```
# régression avec statsmodels avec d'autres modèles statistiques
import statsmodels.api as sm

x = [[0, 1], [5, 1], [15, 2], [25, 5], [35, 11], [45, 15], [55, 34], [60, 35]]
y = [4, 5, 20, 14, 32, 22, 38, 43]
x, y = np.array(x), np.array(y)
```

In [15]:
```
x = sm.add_constant(x)
print(x, ' \n avec une taille : \n', x.shape)
```

```
[[ 1.  0.  1.]
 [ 1.  5.  1.]
 [ 1. 15.  2.]
 [ 1. 25.  5.]
 [ 1. 35. 11.]
 [ 1. 45. 15.]
 [ 1. 55. 34.]
 [ 1. 60. 35.]]
 avec une taille :
 (8, 3)
```

In [16]:
```
#créer le modèle OLS (ordinary least squares) qui minimse l'erreur quadratique
model = sm.OLS(y, x)
```

In [17]:
```
#avec plus de détails sur le modèle
results = model.fit()
#print(results.summary())
```

In [18]:
```
#utiliser le modèle pour la prédiction
x_new = sm.add_constant(np.arange(10).reshape((-1, 2)))
y_new = results.predict(x_new)
print(y_new)
```

```
[ 5.77760476  7.18179502  8.58598528  9.99017554 11.3943658 ]
```

Bien que nous pouvons nous contenter d'un modèle linéaire dans un premier temps, il est bon à savoir que d'autres modèles existent.

Règle d'or : on choisit selon l'application !!

On retient que :

- Le modèle dans sklearn est facile à comprendre et à implimenter
- Statsmodels offre des outils statistiques plus avancés

Un test de la régression linéaire sur des données réelles

Dans cette partie nous allons découvrir comment on peut utiliser python, numpy, matplotlib et sklearn pour classer des fleurs.

Le dataset (jeu de données) est une base de données des caractéristiques de trois espèces de fleurs d'Iris (Setosa, Versicolour et Virginica).

Description :

- chaque ligne est une observation des caractéristiques d'une fleur d'Iris ;
- caractéristiques : longueur et largeur de (sépales, pétales) ;
- dans cet exemple, le dataset continet 150 observations (50 observations par espèce) ;
- plus d'information https://fr.wikipedia.org/wiki/Iris_de_Fisher .

In [19]:
```
#importer les bibliothèques

#pour l'affichage (si déjà fait pour np, plt)
%matplotlib inline

#des datastes dans sklearn pour les tests
from sklearn import datasets
```

In [20]:
```
#charger la base
iris = datasets.load_iris()
#vérifier le type de la variable iris
print(type(iris))
```

```
#vérifier quel est le type de données
print(type(iris.data))
#vérifier les dimensions
print(iris.data.shape)
```

```
<class 'sklearn.utils._bunch.Bunch'>
<class 'numpy.ndarray'>
(150, 4)
```

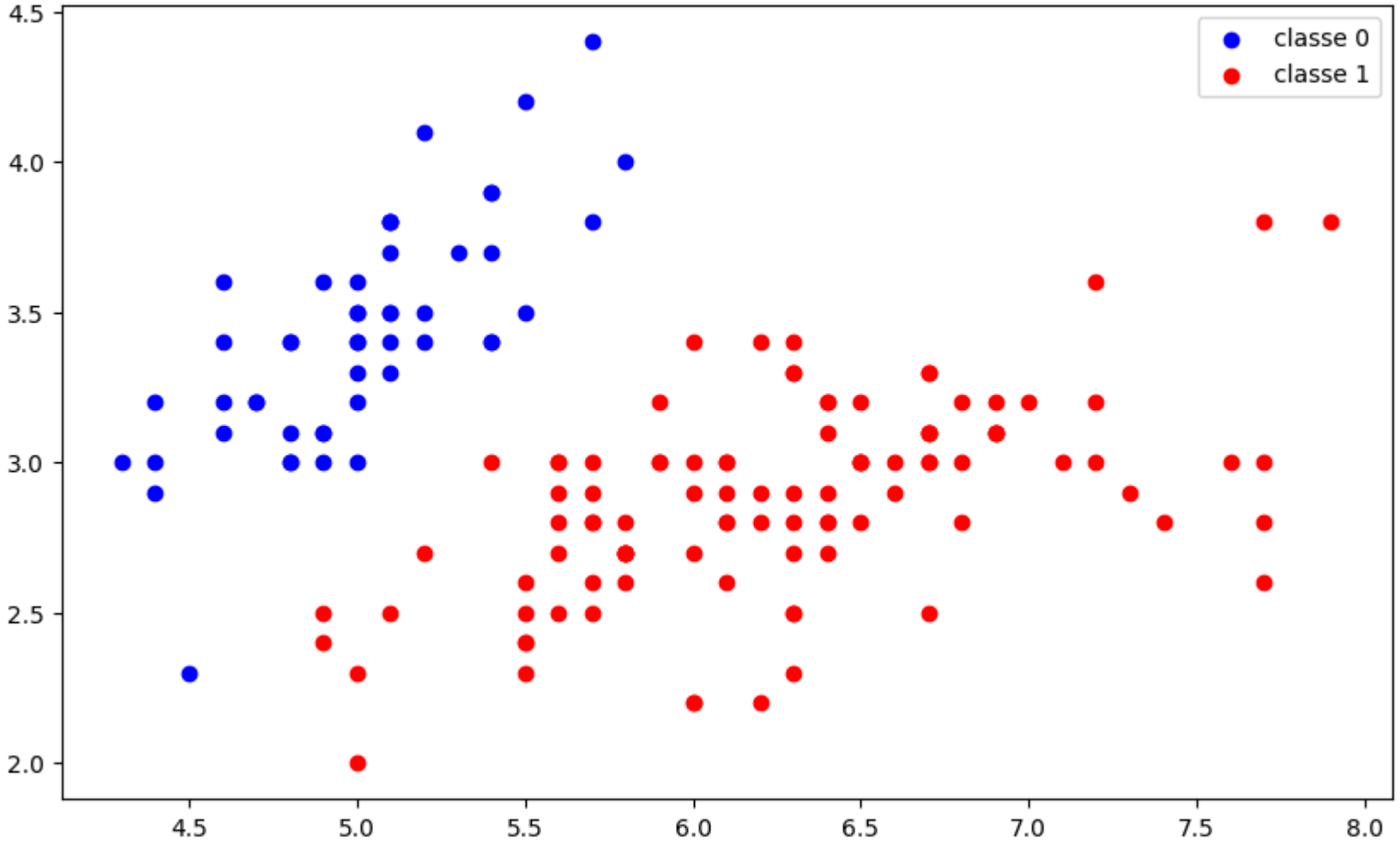
Le choix des variables

In [21]:
```
X = iris.data[:, :2] # Utiliser les deux premières colonnes afin d'avoir un problème de classification binaire.

print(np.unique(iris.target))
#on va garder deux classes seulement pour un test simple
y = (iris.target != 0) * 1 # re-étiquetage des fleurs
print(X.shape)
print(np.unique(y))
```

```
[0 1 2]
(150, 2)
[0 1]
```

Pour mieux comprendre notre data, on va visualiser des exemples

In [22]:
```
#visualisation des données
plt.figure(figsize=(10, 6))
plt.scatter(X[y == 0][:, 0], X[y == 0][:, 1], color='b', label='classe 0')
plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], color='r', label='classe 1')
plt.legend();
```



Une petite vérification visuelle montre que les deux classes peuvtn être séparées par une droite. On dira que elles peuvent être **linéairement** séparées.

Pour la suite on va utiliser une régression logistique pour exploiter cette séparation.

In [23]:
```
#charger le modèle
from sklearn.linear_model import LogisticRegression
```

```
model = LogisticRegression(C=1e20) #  Régression logistique
# Entrainement du modèle avec toutes les données
model.fit(X, y)
```

Out[23]:

LogisticRegression

LogisticRegression(C=1e+20)

In [24]:
```
Xnew = np.array([
    [5.5, 2.5],
    [7, 3],
    [3,2],
    [5,3]
])
```

In [25]:
```
model.predict(Xnew)
```

Out[25]:
```
array([1, 1, 0, 0])
```

In [26]:
```
type(Xnew)
```

Out[26]:
```
numpy.ndarray
```

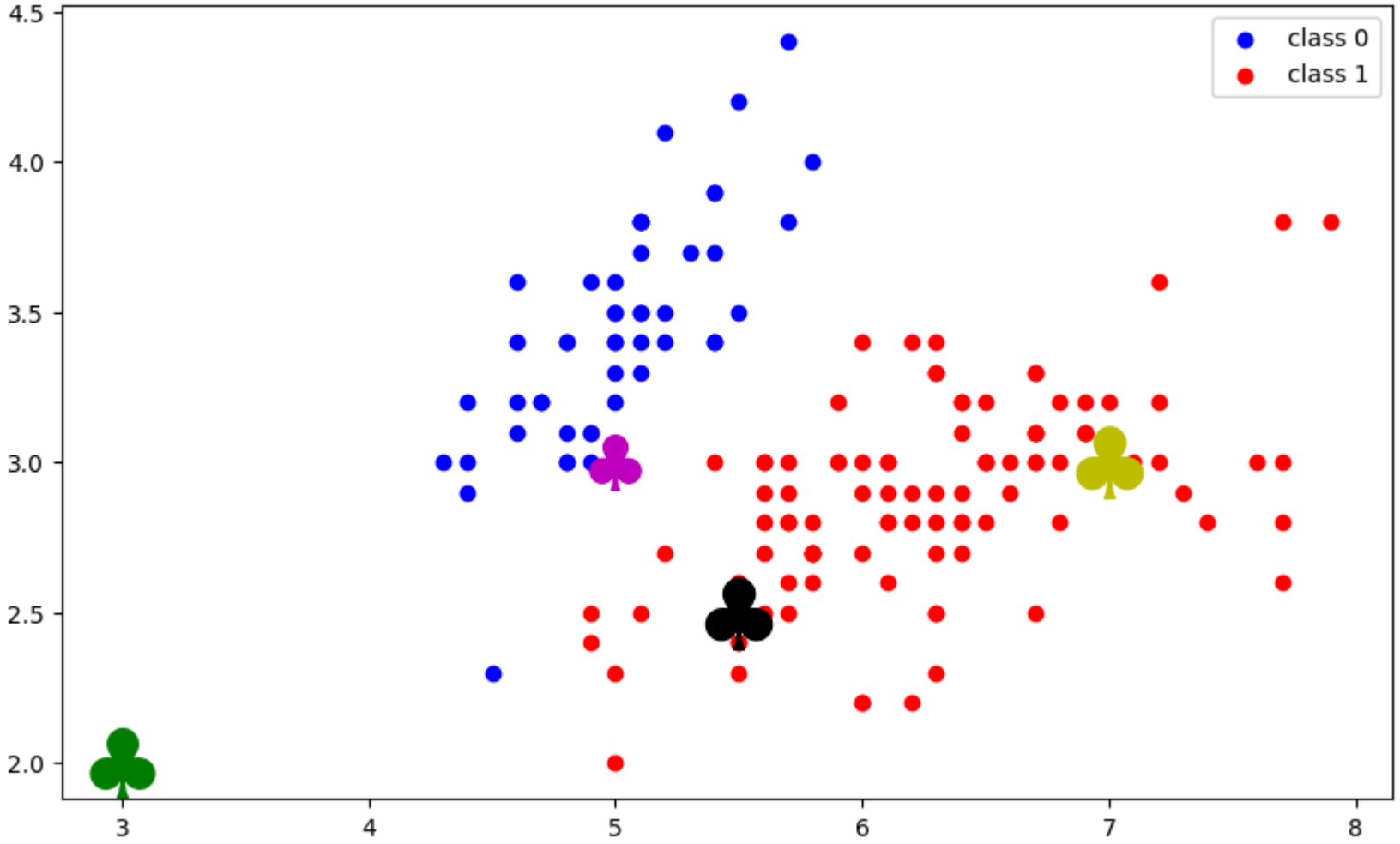
Analyse des résultats :

* La première observation [5.5, 2.5] est de classe 1
* La deuxième observation [7, 3] est de classe 1
* La troisième observation [3,2] est de classe 0
* La quatrième observation [5,3] est de classe 0

In [27]:
```
#vérification visuelle ,,,

#visualisation des données
plt.figure(figsize=(10, 6))
plt.scatter(X[y == 0][:, 0], X[y == 0][:, 1], color='b', label='class 0')
plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], color='r', label='class 1')

s = np.random.rand(*Xnew[:, 0].shape) * 800 + 500
```

```
print(s.shape)
Color='kygm' #noir jaune vert magneta
for i in range(Xnew.shape[0]):
    plt.scatter(Xnew[i, 0], Xnew[i, 1],s[i], color=Color[i],marker=r'$\clubsuit$',)
plt.legend();
```

```
(4,)
```



II- Classification

In [28]:
```
# classification d'images de chiffres
from sklearn.datasets import load_digits

digits = load_digits()
digits.images.shape
```

Out[28]:
```
(1797, 8, 8)
```

In [29]:
```
#visualiser les données
fig, axes = plt.subplots(10, 10, figsize=(8, 8),
                                        subplot_kw={'xticks':[], 'yticks':[]},
                                        gridspec_kw=dict(hspace=0.1, wspace=0.1))

for i, ax in enumerate(axes.flat):
    ax.imshow(digits.images[i], cmap='binary', interpolation='nearest')
    ax.text(0.05, 0.05, str(digits.target[i]),transform=ax.transAxes, color='red')
```

In [30]:
```
X = digits.data
X.shape
```

Out[30]: (1797, 64)

In [31]:
```
y = digits.target
y.shape
```

Out[31]: (1797,)

Préparation pour l'apprentissage : train et test

In [32]:
```
#former les bases d'apprentissage et de test
from sklearn.model_selection import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y,test_size=0.20, random_state=10)
```

In [33]:
```
print(Xtrain.shape)
print(Xtest.shape)
print('pourcentage:' ,Xtrain.shape[0]/X.shape[0])

print(ytrain.shape)
```

```
(1437, 64)
(360, 64)
pourcentage: 0.7996661101836394
(1437,)
```

Classification avec l'arbre de décision

In [34]:
```
from sklearn.tree import DecisionTreeClassifier

Arbre_decision = DecisionTreeClassifier(random_state=0, max_depth=20)
clf = Arbre_decision.fit(Xtrain, ytrain)
```

In [35]:
```
from sklearn.metrics import accuracy_score
ypredit = clf.predict(Xtest)

accuracy_score(ytest, ypredit)
```

Out[35]: 0.8527777777777777

In [36]:
```
from sklearn import  metrics
print(metrics.confusion_matrix(ytest, ypredit))
```

```
[[34  0  0  0  1  0  0  0  1  1]
 [ 0 29  2  1  0  0  0  0  2  0]
 [ 0  2 29  1  0  0  0  0  2  0]
 [ 0  0  1 36  0  0  0  0  1  2]
 [ 0  0  0  0 27  1  2  1  1  2]
 [ 0  0  0  1  0 29  0  1  0  1]
 [ 0  0  0  0  2  1 34  0  0  0]
 [ 0  0  0  0  5  0  0 35  0  0]
 [ 0  2  2  2  1  0  0  0 24  2]
 [ 0  1  1  2  2  1  0  0  2 30]]
```

Classification avec le plus proche voisin

In [37]:
```
from sklearn.neighbors import KNeighborsClassifier

KNN = KNeighborsClassifier()
clf = KNN .fit(Xtrain, ytrain)

ypredit = clf.predict(Xtest)

accuracy_score(ytest, ypredit)
```

Out[37]: 0.9861111111111112

In [38]:
```
print(metrics.confusion_matrix(ytest, ypredit))
```

```
[[37  0  0  0  0  0  0  0  0  0]
 [ 0 34  0  0  0  0  0  0  0  0]
 [ 0  0 34  0  0  0  0  0  0  0]
 [ 0  0  0 40  0  0  0  0  0  0]
 [ 0  0  0  0 33  0  0  0  1  0]
 [ 0  0  0  0  0 32  0  0  0  0]
 [ 0  0  0  0  0  0 37  0  0  0]
 [ 0  0  0  0  0  0  0 40  0  0]
 [ 0  1  0  1  0  0  0  0 31  0]
 [ 0  0  0  0  0  1  0  0  1 37]]
```

Classification avec SVM

In [39]:
```
from sklearn import  svm

clf = svm.SVC(gamma=0.001)
clf.fit(Xtrain,ytrain)
```

Out[39]:
```
SVC(gamma=0.001)
```

In [40]:
```
from sklearn.metrics import accuracy_score
ypredit = clf.predict(Xtest)
accuracy_score(ytest, ypredit)
```

Out[40]:
```
0.9916666666666667
```

In [41]:
```
ypredit = clf.predict(Xtest)
print(metrics.confusion_matrix(ytest, ypredit))
```

```
[[37  0  0  0  0  0  0  0  0  0]
 [ 0 34  0  0  0  0  0  0  0  0]
 [ 0  0 34  0  0  0  0  0  0  0]
 [ 0  0  0 40  0  0  0  0  0  0]
 [ 0  0  0  0 33  0  0  0  1  0]
 [ 0  0  0  0  0 32  0  0  0  0]
 [ 0  0  0  0  0  0 37  0  0  0]
 [ 0  0  0  0  0  0  0 40  0  0]
 [ 0  1  0  0  0  0  0  0 32  0]
 [ 0  0  0  0  0  0  0  0  1 38]]
```

********** Pratique avec le dataset Titanic

In [42]:
```
import pandas as pd

df=pd.read_csv('./titanic/train.csv')
```

In [43]:
```
df.head(5)
```

Out[43]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

In [44]:
```
df.shape
```

Out[44]:
```
(891, 12)
```

In [45]:
```
df.describe()
```

Out[45]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

In [46]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

La signification des colonnes :

PassengerID : identifiant

Survived : 0 si ce passager n'a pas survécu, 1 sinon

Pclass : la classe (1, 2 ou 3)

Name : le nom du passager

Sex : femme ou homme (sexe)

Age : l'âge (en années)

SibSp : le nombre de frère, soeur et/ou épouse à bord

Parch : le nombre de parent et/ou d'enfant à bord

Ticket : numéro du ticket

Fare : prix du billet

Cabin : numéro de cabine

Embarked : port d'embarquement(C = Cherbourg, Q = Queenstown, S = Southampton)

Exo 1 : dans un premier temps on explore pour comprendre les données

- Lire et explorer le dataset à partir des fichiers csv.
- Vérifier que le dataset contient des valeurs manquantes.
- Proposer une solution pour remplacer les valeurs manquantes.
- Est il possible de visualiser les données ?

- Créer une nouvelle colonne qui mentionne le sexe du passager.
- Combien de personnes ont survécu ?
- Est ce que la Pclass a plus d'importance que l'âge pour la survie ?
- Pour répondre à cette question, nous allons tester la classification : survived en fontion de l'âge (repectivement Pclass).
- On peut tirer une conclusion à partir de la performance mais relativement au modèle utilisé.

Exo 2 : dans un deuxième temps on va transformer quelques colonnes

- Transformer le sexe en nombre.
- On teste si (âge et sexe) est mieux que (classe, sexe).
- On teste la performance de trois différentes méthodes pour la clasification.
- Proposer comment transformer d'autres colonnes.
- Tester l'importance de chaque colonne à part, des combinaisons, etc.
- Conclure après plusieurs tests : modèles + features.