Compte Rendu de TP 02

Diagnostic par apprentissage

Nom : BOUARICHE

Prénom : Iheb

Année : 2023/2024

Spécialité : Instrumentation an2

tp02-da

December 1, 2023

1 Exercice 01

Question 01:

[ ]:
```
import numpy as np
import scipy
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
```

Commentaire: ici on a importé les bibliothèque.

Question 02:

[ ]:
```
np.random.seed(0)
m=100
X = np.linspace(0,10,m).reshape(m,1)
Y = X + np.random.random_sample((m,1))
```

Commentaire :

On a creé 100 donnés entre 0 et 10 et on les suppose comme des entrées “X”. On a ajouté pour chaque entrée un nombre aléatoire. Et cela représente les sorties “Y”.

[ ]:
```
print(X)
```

```
[[ 0.        ]
 [ 0.1010101 ]
 [ 0.2020202 ]
 [ 0.3030303 ]
 [ 0.4040404 ]
 [ 0.50505051]
 [ 0.60606061]
 [ 0.70707071]
 [ 0.80808081]
 [ 0.90909091]
 [ 1.01010101]
 [ 1.11111111]
 [ 1.21212121]
 [ 1.31313131]
 [ 1.41414141]
```

```
[ 1.51515152]
[ 1.61616162]
[ 1.71717172]
[ 1.81818182]
[ 1.91919192]
[ 2.02020202]
[ 2.12121212]
[ 2.22222222]
[ 2.32323232]
[ 2.42424242]
[ 2.52525253]
[ 2.62626263]
[ 2.72727273]
[ 2.82828283]
[ 2.92929293]
[ 3.03030303]
[ 3.13131313]
[ 3.23232323]
[ 3.33333333]
[ 3.43434343]
[ 3.53535354]
[ 3.63636364]
[ 3.73737374]
[ 3.83838384]
[ 3.93939394]
[ 4.04040404]
[ 4.14141414]
[ 4.24242424]
[ 4.34343434]
[ 4.44444444]
[ 4.54545455]
[ 4.64646465]
[ 4.74747475]
[ 4.84848485]
[ 4.94949495]
[ 5.05050505]
[ 5.15151515]
[ 5.25252525]
[ 5.35353535]
[ 5.45454545]
[ 5.55555556]
[ 5.65656566]
[ 5.75757576]
[ 5.85858586]
[ 5.95959596]
[ 6.06060606]
[ 6.16161616]
[ 6.26262626]
```

```
 [ 6.36363636]
 [ 6.46464646]
 [ 6.56565657]
 [ 6.66666667]
 [ 6.76767677]
 [ 6.86868687]
 [ 6.96969697]
 [ 7.07070707]
 [ 7.17171717]
 [ 7.27272727]
 [ 7.37373737]
 [ 7.47474747]
 [ 7.57575758]
 [ 7.67676768]
 [ 7.77777778]
 [ 7.87878788]
 [ 7.97979798]
 [ 8.08080808]
 [ 8.18181818]
 [ 8.28282828]
 [ 8.38383838]
 [ 8.48484848]
 [ 8.58585859]
 [ 8.68686869]
 [ 8.78787879]
 [ 8.88888889]
 [ 8.98989899]
 [ 9.09090909]
 [ 9.19191919]
 [ 9.29292929]
 [ 9.39393939]
 [ 9.49494949]
 [ 9.5959596 ]
 [ 9.6969697 ]
 [ 9.7979798 ]
 [ 9.8989899 ]
 [10.        ]]
```

[ ]:
```
print(Y)
```

```
[[ 0.5488135 ]
 [ 0.81619947]
 [ 0.80478358]
 [ 0.84791349]
 [ 0.8276952 ]
 [ 1.15094462]
 [ 1.04364782]
 [ 1.59884371]
```

```
[ 1.77174357]
[ 1.29253243]
[ 1.80182605]
[ 1.64000603]
[ 1.78016577]
[ 2.23872795]
[ 1.48517747]
[ 1.60228081]
[ 1.63638001]
[ 2.54979156]
[ 2.59633857]
[ 2.78920407]
[ 2.99882036]
[ 2.92037069]
[ 2.68370158]
[ 3.1037615 ]
[ 2.54251685]
[ 3.16517355]
[ 2.76961591]
[ 3.67194164]
[ 3.35013115]
[ 3.34395487]
[ 3.29485864]
[ 3.90554682]
[ 3.68847356]
[ 3.90176728]
[ 3.45313323]
[ 4.15298903]
[ 4.24845936]
[ 4.35430773]
[ 4.78213192]
[ 4.62121424]
[ 4.39991194]
[ 4.5784461 ]
[ 4.94005544]
[ 4.40365982]
[ 5.11121116]
[ 5.21609242]
[ 4.85684721]
[ 4.87640105]
[ 5.1639132 ]
[ 5.31320572]
[ 5.62070182]
[ 5.59011666]
[ 6.24089909]
[ 5.45558016]
[ 5.66342221]
[ 5.71686507]
```

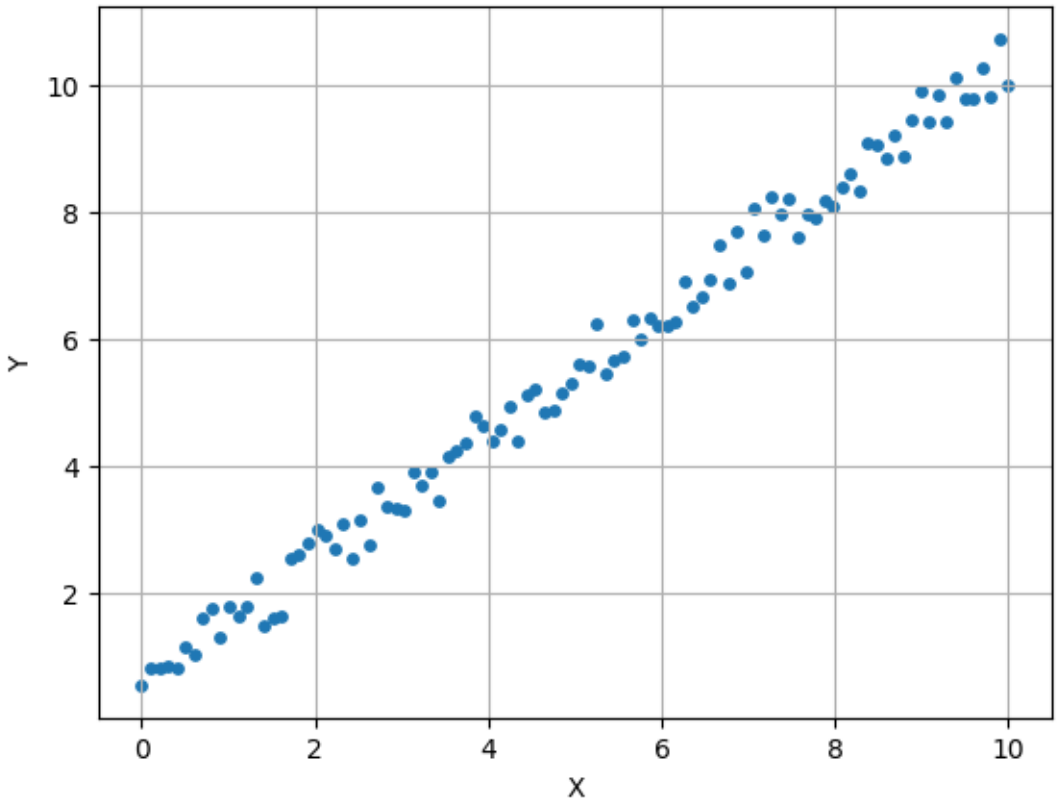
```
 [ 6.30967398]
 [ 6.01086736]
 [ 6.32489663]
 [ 6.20402155]
 [ 6.21957564]
 [ 6.2719913 ]
 [ 6.91895585]
 [ 6.50181931]
 [ 6.66122883]
 [ 6.93438174]
 [ 7.4876599 ]
 [ 6.86477804]
 [ 7.70663178]
 [ 7.06579538]
 [ 8.04716654]
 [ 7.64036837]
 [ 8.24948836]
 [ 7.97858289]
 [ 8.21401105]
 [ 7.61494537]
 [ 7.95957464]
 [ 7.89797434]
 [ 8.17492808]
 [ 8.0985257 ]
 [ 8.39879126]
 [ 8.59608118]
 [ 8.34697578]
 [ 9.0763105 ]
 [ 9.05144994]
 [ 8.85124808]
 [ 9.21011674]
 [ 8.8818193 ]
 [ 9.46483538]
 [ 9.91919519]
 [ 9.40947804]
 [ 9.85932957]
 [ 9.42472716]
 [10.1102666 ]
 [ 9.78435559]
 [ 9.77915096]
 [10.28348263]
 [ 9.81808734]
 [10.72792993]
 [10.00469548]]
```

X représente les entrés et Y sont les sorties.

Question 03:

[ ]:
```
plt.scatter(X,Y,s=15)
plt.grid(True)
plt.xlabel('X')
plt.ylabel('Y')
```

[ ]:
```
Text(0, 0.5, 'Y')
```



On peut voir la sortie en fonction de l'entrée de nos données, et les points sont distribués d'une façon lineaire. ça représente l'entrée qui égale la sortie mais avec l'ajout d'une valeur aléatoire.

Question 4:

[ ]:
```
model = LinearRegression()
```

[ ]:
```
model.fit(X,Y)
```

[ ]:
```
LinearRegression()
```

[ ]:
```
model.score(X,Y)
```

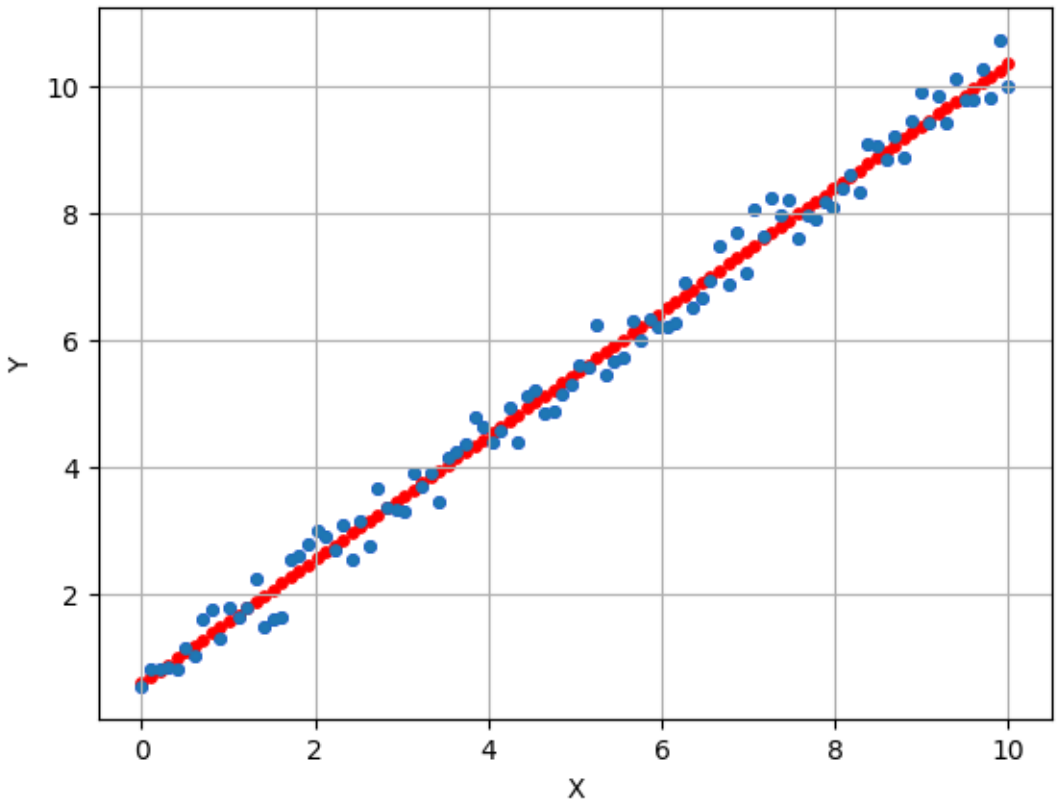
[ ]:
```
0.9904652159218895
```

Commentaire: On a un score de 99% ce qui est un trés bon score pour une application de regression.

[ ]:
```
Y_predict = model.predict(X)
```

[ ]:
```
plt.scatter(X, Y, color='blue', s=15)
plt.scatter(X, Y_predict, color='red',s=15)
plt.scatter(X,Y,s=15)
plt.grid(True)
plt.xlabel('X')
plt.ylabel('Y')
```

[ ]:
```
Text(0, 0.5, 'Y')
```



On remarque que les données prédictés sont lineaires car notre model est lineaire de degré un. mais on peut remarquer que le modele donne des predictions qui son au maximum proche aux points reels. Ce qui justifie que notre model a était bien entrainer sur nos données.

Question 05:

[ ]:
```
np.random.seed(0)
m=100
X = np.linspace(0,10,m).reshape(m,1)
```

```
Y = X**2 + np.random.random_sample((m,1))
```

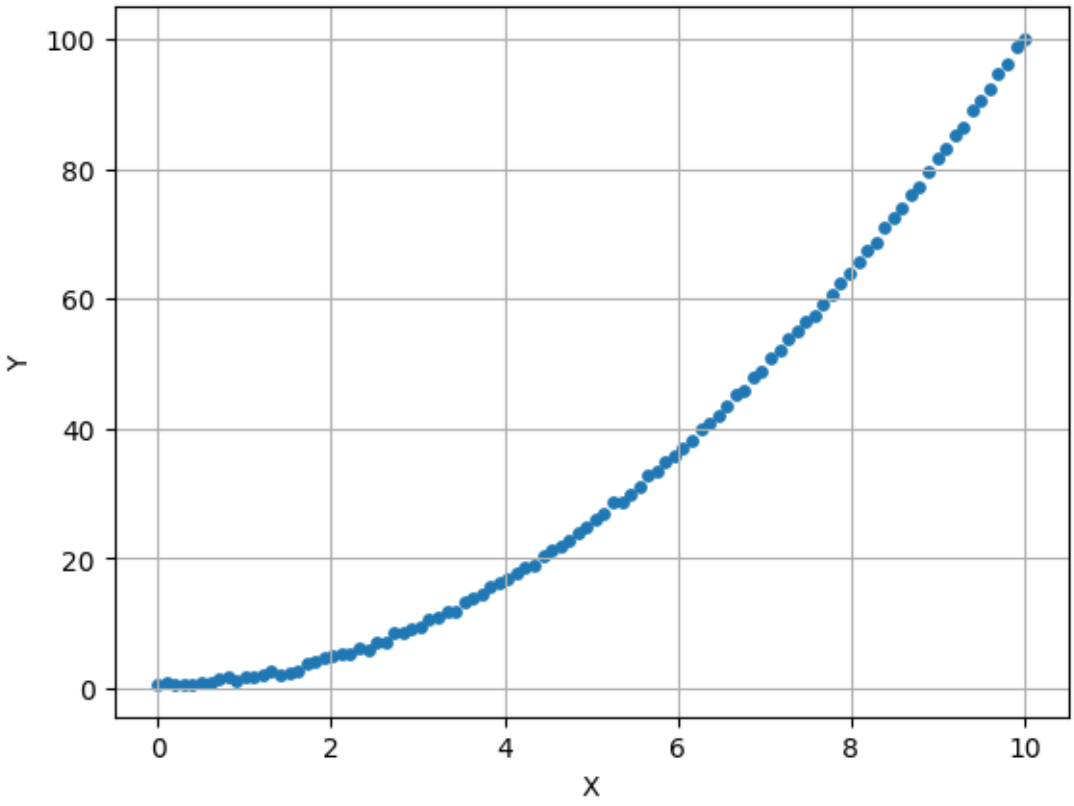
Commentaire:

Maintenant on a construit un jeu de données avec les memes entrées et avec des sortie qui non lineaires (une fonction carrée de deuxieme degrée) et on a aussi ajouté des valeurs aléatoires pour chaque valeur de sortie.

Question 06:

[ ]:
```
plt.scatter(X,Y,s=15)
plt.grid(True)
plt.xlabel('X')
plt.ylabel('Y')
```

[ ]:
```
Text(0, 0.5, 'Y')
```



On remarque que note donnés Y en fonction de X sont distribués de façon non-lineaires.

Question 07:

[ ]:
```
model = LinearRegression()
```

[ ]:
```
model.fit(X,Y)
```

[ ]:
```
LinearRegression()
```

[ ]:
```
coefficients = model.coef_
intercept = model.intercept_
print("La valeur de parametre a :", coefficients)
print("La valeur de parametre b :", intercept)
```

```
La valeur de parametre a : [[9.97498813]]
La valeur de parametre b : [-15.90046332]
```

[ ]:
```
model.score(X,Y)
```

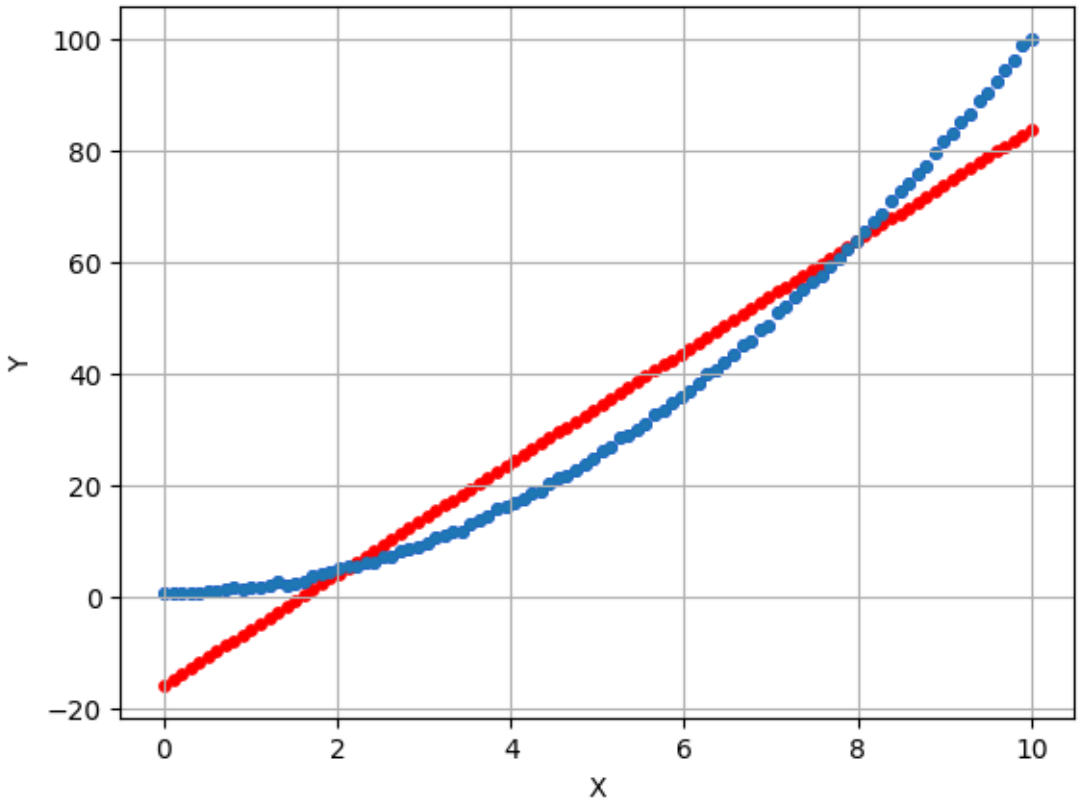
[ ]:
```
0.9355999446562453
```

On voit qu'on a un erreur de 0.07 donc 7%, qui est un peut dégradé par rapport le score obtenu pour les données lineaires.

[ ]:
```
Y_predict = model.predict(X)
```

[ ]:
```
plt.scatter(X, Y, color='blue', s=15)
plt.scatter(X, Y_predict, color='red',s=15)
plt.scatter(X,Y,s=15)
plt.grid(True)
plt.xlabel('X')
plt.ylabel('Y')
```

[ ]:
```
Text(0, 0.5, 'Y')
```



On remarque que la prédiction est lineaire comparant aux doonées de l'entrainement (les données réels), donc on a des rédultats qui sont pas bons.

Question 08

[ ]:
```
from sklearn.svm import SVR
```

[ ]:
```
model = SVR(C=100)
```

Le parametre "C" représente le paramètre de régularisation de model de support vector regression.

[ ]:
```
model.fit(X,Y)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
```

[ ]:
```
SVR(C=100)
```

[ ]:
```
model.score(X,Y)
```

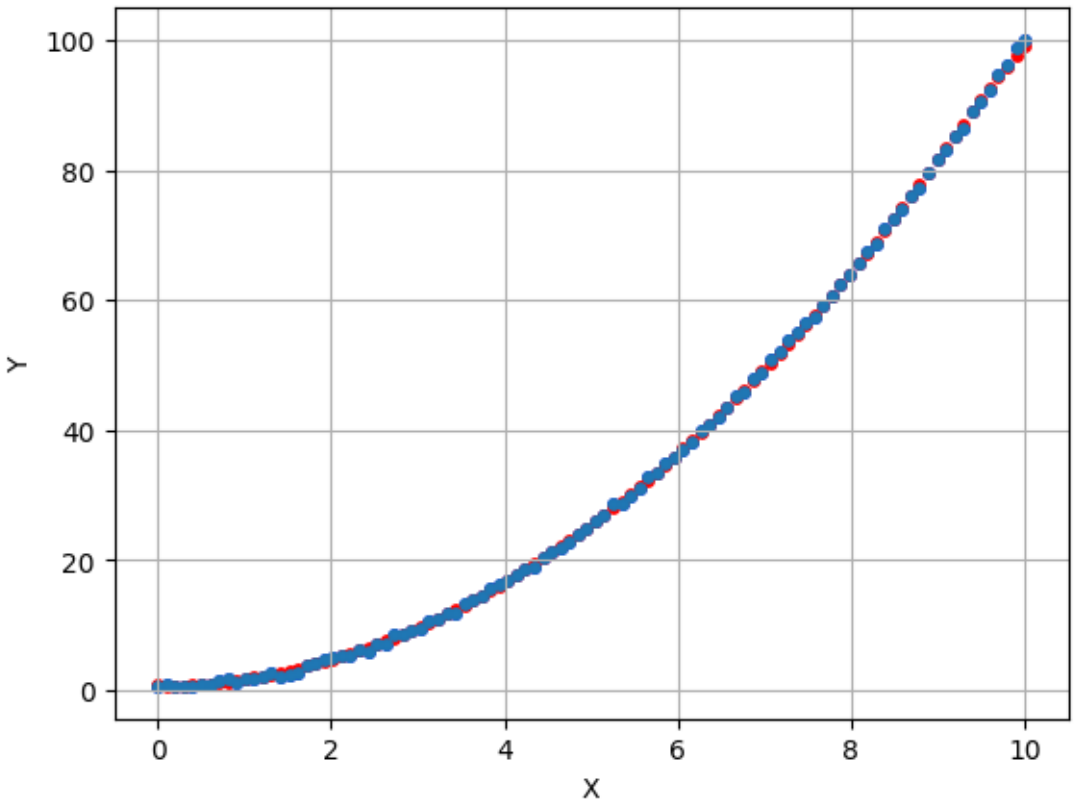
[ ]:
```
0.9998846009080101
```

Avec cette algorithme (model) on a obtenu des bonnes résultats et une erreur qui éguale a 0.0002 (presque nul), et donc on a une bonne regression dans ce cas.

[ ]:
```
Y_predict = model.predict(X)
```

[ ]:
```
plt.scatter(X, Y, color='blue', s=15)
plt.scatter(X, Y_predict, color='red',s=15)
plt.scatter(X,Y,s=15)
plt.grid(True)
plt.xlabel('X')
plt.ylabel('Y')
```

[ ]:
```
Text(0, 0.5, 'Y')
```



D'aprés le graphe on remarque que notre modele a bien prédicté les valeurs de l'entrées. et ça vérifie le score obtenu de 0.9998 (99.98%).

2 Exercice 02:

Question 01

[ ]:
```
import pandas as pd
import seaborn as sns
```

Commentaire: ici on a importé les bibliothèque.

Question 02

[ ]:
```
Titanic = sns.load_dataset('titanic')
print(Titanic.shape)
```

```
(891, 15)
```

Commentaire: On a importé le jeu de données Titanic, et on peut voire la taille de données qui a 15 classes et 891 lines de données.

Question 03:

[ ]:
```
Titanic.head()
```

[ ]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	\
0	0	3	male	22.0	1	0	7.2500	S	Third	
1	1	1	female	38.0	1	0	71.2833	C	First	
2	1	3	female	26.0	0	0	7.9250	S	Third	
3	1	1	female	35.0	1	0	53.1000	S	First	
4	0	3	male	35.0	0	0	8.0500	S	Third	

	who	adult_male	deck	embark_town	alive	alone
0	man	True	NaN	Southampton	no	False
1	woman	False	C	Cherbourg	yes	False
2	woman	False	NaN	Southampton	yes	True
3	woman	False	C	Southampton	yes	False
4	man	True	NaN	Southampton	no	True

La commande “head” est utilisé pour afficher les premiers 5 lignes de données.

Question 04:

[ ]:
```
Titanic.dropna(axis=0,inplace=True)
```

[ ]:
```
X = Titanic[['pclass','sex','age']]
```

[ ]:
```
X
```

[ ]:

	pclass	sex	age
1	1	female	38.0
3	1	female	35.0
6	1	male	54.0

```
10        3  female   4.0
11        1  female  58.0
..      …     …   …
871       1  female  47.0
872       1    male  33.0
879       1  female  56.0
887       1  female  19.0
889       1    male  26.0

[182 rows x 3 columns]
```

[ ]:
```
X['sex'].replace(['male','female'],[0,1],inplace=True)
```

```
<ipython-input-634-1f52f3c94577>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X['sex'].replace(['male','female'],[0,1],inplace=True)
```

[ ]:
```
X
```

[ ]:
```
     pclass  sex   age
1         1    1  38.0
3         1    1  35.0
6         1    0  54.0
10        3    1   4.0
11        1    1  58.0
..      …  …   …
871       1    1  47.0
872       1    0  33.0
879       1    1  56.0
887       1    1  19.0
889       1    0  26.0

[182 rows x 3 columns]
```

[ ]:
```
Y = Titanic[['survived']]
```

[ ]:
```
Y
```

[ ]:
```
     survived
1           1
3           1
6           0
10          1
11          1
```

```
..        …
871         1
872         0
879         1
887         1
889         1

[182 rows x 1 columns]
```

Cette écriture est utilisé pour la préparation de données, on a éliminé les valeurs manquantes et on a pris les entrées pour les données “pclass” , “sex” , “age” et les données de prédiction c’est la classe “survived”. Donc X représente les classes “pcclass”, “sex”, “age” et pour la prédiction on a pris la classe “survived” pour savoir si la personne a survis.

Avant de commencer l’entrainement on va deviser les données a deux groupe.

[ ]:
```
from sklearn.model_selection import train_test_split
t = 0.2
X, X_test, Y, Y_test = train_test_split(X,Y,test_size=t)
print('train set:',X.shape)
print('test set:',X_test.shape)
```

```
train set: (145, 3)
test set: (37, 3)
```

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

[ ]:
```
model = KNeighborsClassifier()
```

D’aprés la documentation de Sklearn, le parametre K est égale a 5. donc il va prendre en considération seulement 5 voisins.

[ ]:
```
model.fit(X,Y)
```

```
/usr/local/lib/python3.10/dist-
packages/sklearn/neighbors/_classification.py:215: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape
of y to (n_samples,), for example using ravel().
  return self._fit(X, y)
```

[ ]:
```
KNeighborsClassifier()
```

[ ]:
```
model.score(X,Y)
```

[ ]:
```
0.8068965517241379
```

Les résultats obtenus sont pas vraiment bon.

Question 05

[ ]:
```
def survie(model, pclass=3, sex=0, age=26):
  x = np.array([pclass,sex,age]).reshape(1,3)
  if model.predict(x):
    print(" 'Survécu' car on, car le modele nous a donné ", model.predict(x))
  else:
    print(" 'non pas survé, car le modele nous a donné' ", model.predict(x))
```

[ ]:
```
survie(model,pclass=3, sex=0, age=26)
```

```
 'Survécu' car on, car le modele nous a donné  [1]
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does
not have valid feature names, but KNeighborsClassifier was fitted with feature
names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does
not have valid feature names, but KNeighborsClassifier was fitted with feature
names
  warnings.warn(
```

[ ]:
```
print(model.predict_proba(np.array([3,0,26]).reshape(1,3)))
```

```
[[0.2 0.8]]
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does
not have valid feature names, but KNeighborsClassifier was fitted with feature
names
  warnings.warn(
```

Commentaire: Ici j'ai crée une fonction qui va utilisé le modele entrainé pour savoir la réponse si la personne a survie ou non.

Question 06

[ ]:
```
train_scores = []
test_score = []
for k in range(10):
  model = KNeighborsClassifier(n_neighbors=k+1)
  model.fit(X,Y)
  score = model.score(X,Y)
  train_scores.append(score)
  test_score.append(model.score(X_test,Y_test))
```

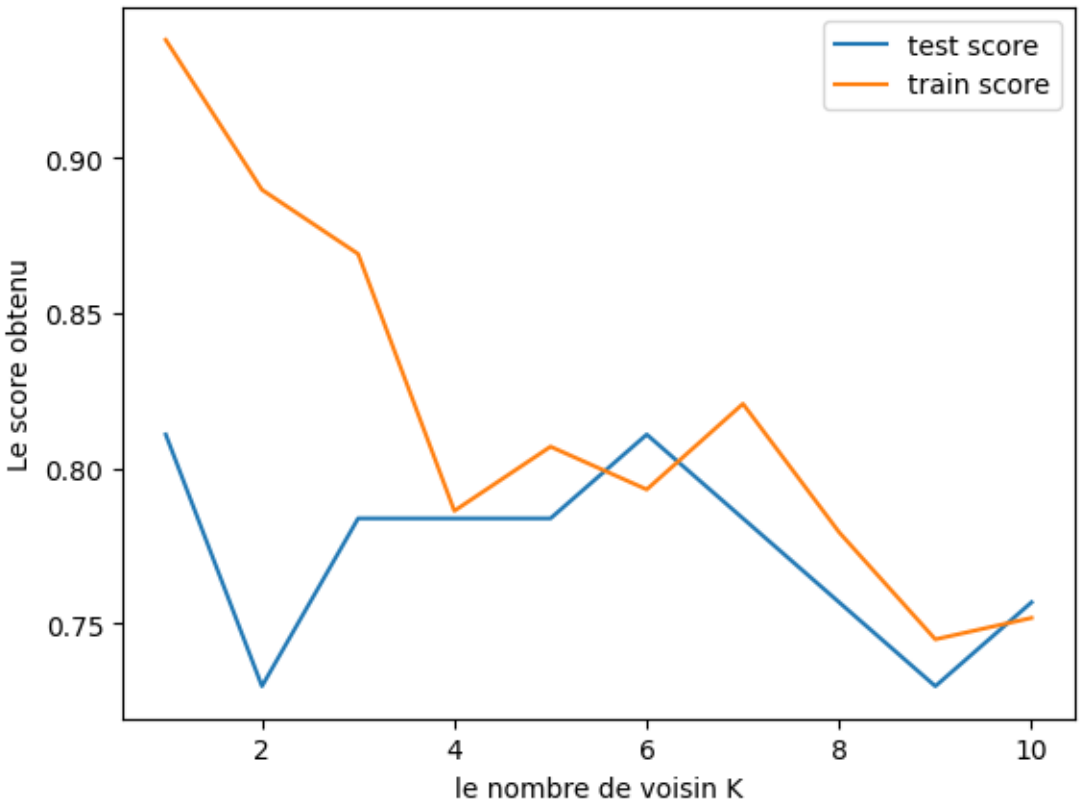
[ ]:
```
print("Le meilleur score de donnés testés est:",max(test_score))
```

```
Le meilleur score de donnés testés est: 0.8108108108108109
```

[ ]:
```
K = [k+1 for k in range(10)]
```

```
[ ]: plt.plot(K,test_score,label="test score")
     plt.plot(K,train_scores,label="train score")
     plt.legend()
     plt.xlabel("le nombre de voisin K")
     plt.ylabel("Le score obtenu")
```

```
[ ]: Text(0, 0.5, 'Le score obtenu')
```



Commentaire: on peux voire que a k = 6 on a obtenu le meilleur résultat, notre modele a le meme score pour les donnés d'entrainement et les données de test. D'aprés le graphe on peux aussi prendre la valeur de k = 5 pour 5 vosins, car on a un bon score de test. pour le cas de k = 1 on peux pas prendre ce modele car on a une grande marge entre les données d'entrainement et de test et aussi on peux pas se contenter sur un seul voisin (pour le cas d'un nombre de voisin de 1).