Auriez-vous survécu au naufrage du titanic?

- Le but de cet exercice est de développer un modèle de prédiction de type kNN sur le jeu de données des survivants du Titanic.
- Ce modèle devra être développé et optimisé suivant la méthodologie présentée à l'exercice 1 (iris)
- Il pourra alors être utilisé pour déterminer qui d'entre nous aurait eu le plus de chance de survivre au naufrage du Titanic
- On pourra préciser les probabilités de survie à l'aide de la fonction predict_proba

1) Chargement et préparation des données

```
#On commence par importer nos bibliothèques habituelles
import sklearn
from sklearn import datasets
import numpy as np
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt

titanic=sb.load_dataset('titanic') # on importe le data set depuis
Seaborn
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
```

```
# Modifier les données de la manière suivante :
# on ne garde que ces 4 caractéristiques :
'survived','pclass','sex','age'
titanic.drop(['sibsp', 'parch', 'fare',
       'embarked', 'class', 'who', 'adult_male', 'deck',
```

```
'embark_town',
       'alive', 'alone'],axis=1,inplace=True)
# on élimine les données incomplètes
titanic.dropna(axis=0,inplace=True)
# on remplace male par 0 et female par 1
titanic['sex'] = titanic['sex'].replace(['male'], 0)
titanic['sex'] = titanic['sex'].replace(['female'], 1)
titanic.head()
```

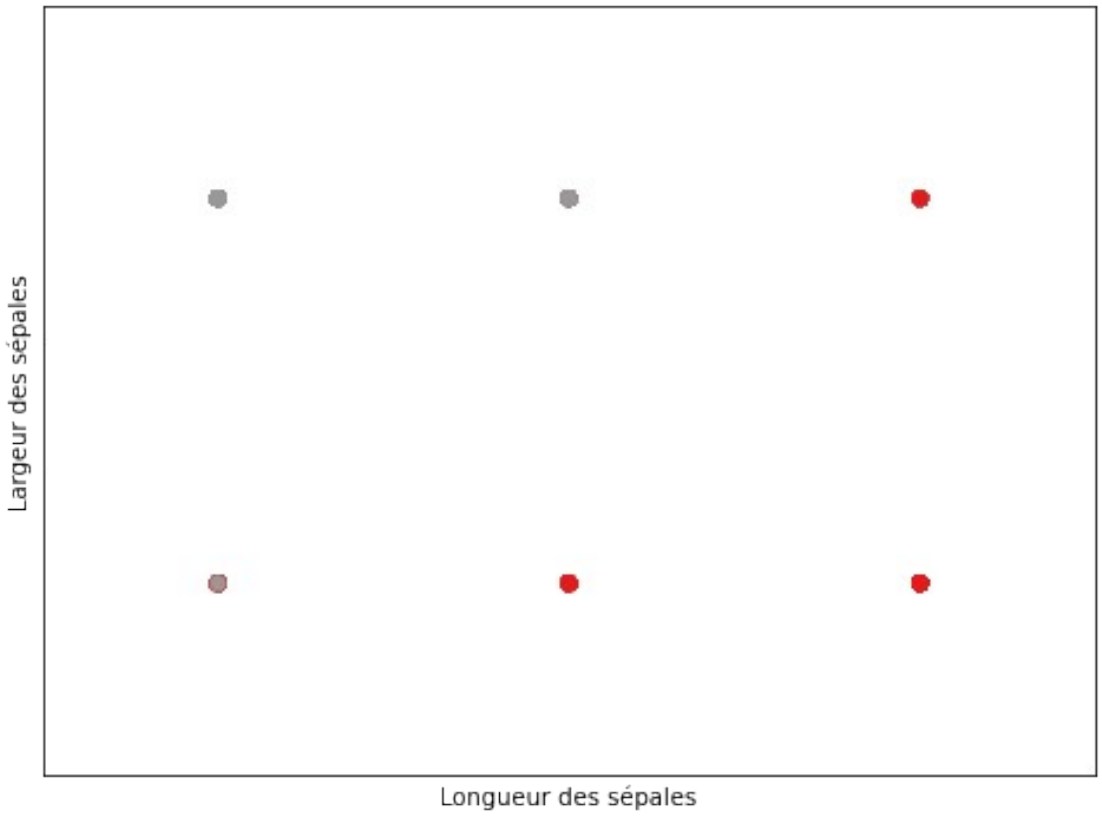
	survived	pclass	sex	age
0	0	3	0	22.0
1	1	1	1	38.0
2	1	3	1	26.0
3	1	1	1	35.0
4	0	3	0	35.0

```
# la colonne survided sera notre y et class, sexe et âges nos
caractéristiques X
caracY=titanic['survived']
caracX=titanic[['pclass','sex','age']]
Y=caracY.to_numpy()
X=caracX.to_numpy()
caracX.head()
```

	pclass	sex	age
0	3	0	22.0
1	1	1	38.0
2	3	1	26.0
3	1	1	35.0
4	3	0	35.0

```
# On représente les données pour se faire une petite idée
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
plt.figure(0, figsize=(8, 6))
plt.clf()
plt.scatter(X[:, 0], X[:, 1],c=Y,cmap=plt.cm.Set1)
plt.xlabel('Longueur des sépales')
plt.ylabel('Largeur des sépales')
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.xticks(())
plt.yticks(())
# Ici on represente le sex en fonction de la class.
```

```
([], <a list of 0 Text major ticklabel objects>)
```




```
#On va faire un split pour partager les donnees en un jeu
d'apprentissage et de test
from sklearn.model_selection import train_test_split
v=[]
V=train_test_split(X, Y, train_size=0.8,test_size=0.2)
X1 = V[0]#les donnees d'apprentissage pour X
X2 = V[1]#Les donnees de test pour X
Y1 = V[2]#Les donnees d'apprentissage pour Y
Y2 = V[3]#les donnees de test pour Y
print(X,Y)
```

```
[[ 3.  0. 22.]
 [ 1.  1. 38.]
 [ 3.  1. 26.]
 ...
 [ 1.  1. 19.]
 [ 1.  0. 26.]
 [ 3.  0. 32.]] [0 1 1 1 0 0 0 1 1 1 1 0 0 0 1 0 0 0 1 1 1 0 1 0 0 0 0
0 0 0 1 0 0 1 1 0 0
 0 1 1 0 1 0 1 0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 1 1 0 1 0 1 1 0 1 0 0 0 0
0 0
 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 1 0 0 1
0 0
 0 1 1 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0
1 1
```

```
 1 0 0 1 0 1 1 1 1 0 0 0 0 0 1 0 0 1 1 1 0 1 0 0 1 1 0 1 0 1 0 0 1 0 1
0 0
 1 0 0 1 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 1 1
0 1
 0 0 1 0 0 0 1 0 1 0 1 1 1 1 0 0 0 0 0 1 0 1 1 0 1 1 1 0 0 0 1 1 0 1 1
0 0
 1 1 1 0 1 1 1 0 0 0 0 1 1 0 1 1 0 0 0 1 1 1 0 0 0 0 0 1 0 0 0 0 0 0 1
1 1
 0 0 0 0 1 0 0 0 1 1 0 1 0 0 1 1 1 1 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 1 0
1 1
 1 1 0 0 0 0 0 0 1 1 1 1 1 0 0 1 0 1 0 0 1 0 0 1 1 1 1 1 1 0 0 1 1 0 1
1 0
 0 0 0 0 1 0 1 1 0 0 0 0 1 0 0 1 1 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0
1 1
 1 1 0 0 1 1 0 1 0 1 0 1 0 0 1 0 0 1 0 1 1 1 0 0 1 0 0 1 0 1 1 0 1 1 0
1 1
 1 0 0 0 0 0 1 1 1 1 0 0 1 1 1 1 1 0 0 1 0 1 0 0 1 0 0 0 0 1 1 0 1 0 0
1 1
 1 0 0 1 0 0 1 0 0 1 1 0 0 0 0 1 0 1 0 1 0 1 0 0 0 0 1 0 1 1 0 1 1 1 0
0 0
 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 0 0 1 0 1 1 0 0 0 0 0 0 0 1 1 1 0 0 0 0
0 0
 1 1 0 0 0 0 1 1 1 1 1 0 0 0 1 1 0 1 0 0 0 1 0 1 0 0 1 0 0 0 0 0 1 0 1
0 1
 0 0 1 0 0 1 1 0 0 1 1 0 0 0 1 0 1 1 0 1 0 0 0 0 0 1 0 1 1 1 1 0 0 0 1
0 1
 0 0 0 0 1 1 0 0 0 1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 1 0 1 1 1
1 0
 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 1 1 1 1 0 0 1 0 1 1 0 1 0 1 0 0 1 1
0 0
 1 1 0 0 0 0 0 0 0 1 1 0]

# On entraine un modele du plus proche voisin
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier()
neigh.fit(X1, Y1)

KNeighborsClassifier()

# on compare les donnees reel de test avec celles de la prediction a
partir des donnees d'apprentissage
print(Y2)
print(neigh.predict(X2))

[0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 1 0 1 1 0 0
1 1
 0 0 0 1 0 0 0 0 1 0 0 0 1 0 1 1 1 1 0 1 1 0 0 0 1 1 0 0 0 0 0 1 0 0 0
0 0
 1 0 0 1 0 1 1 0 1 0 0 0 0 0 0 1 0 0 0 1 1 1 1 0 0 0 1 0 0 0 0 1 1 0 1
1 0
 0 0 1 0 0 1 0 0 0 0 0 1 0 1 0 0 0 1 1 1 0 0 0 1 1 0 1 0 0 0 1 0]
```

```
[0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0 0 1 1 0 1
1 1
 0 0 1 1 0 0 1 1 1 0 0 0 0 1 0 1 0 0 0 1 1 0 1 0 1 1 0 0 0 0 0 1 0 0 0
0 0
 1 0 0 1 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 1 1 1 0 0 0 1 0 1 0 0 0 1 1 1
1 1
 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 1 1 0 0 1 0 1 0 0 1 0 0 0 0 0]
```

```
# Calcul de score sur les données d'apprentissage
print(neigh.score(X1, Y1, sample_weight=None))
# Calcul de score sur les données de test
print(neigh.score(X2, Y2, sample_weight=None))
```

```
0.8283712784588442
0.7762237762237763
```

```
# Regardons les resultats d'une validation croise
from sklearn.model_selection import cross_val_score
print(cross_val_score(neigh.fit(X1, Y1),X2, Y2,n_jobs=5))
print("le score est de",np.mean(cross_val_score(neigh.fit(X1, Y1),X2,
Y2,n_jobs=5)),"avec un ecart type
de",np.std(cross_val_score(neigh.fit(X1, Y1),X, Y, cv=5)) )
```

```
[0.75862069 0.5862069  0.79310345 0.60714286 0.64285714]
le score est de 0.6775862068965517 avec un ecart type de
0.021335332953591652
```

```
# utilisons GridSearchCV afin de trouver le meilleur nombre de voisin
ainsi que la meilleur distance a utiliser
from sklearn.model_selection import GridSearchCV
from numpy import *
GS=GridSearchCV(cv=5, error_score='raise',
       estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform'), n_jobs=1,
       param_grid={'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,
8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19]), 'metric': ['euclidean', 'manhattan']},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring=None, verbose=0)
GS.fit(X1, Y1)
```

```
GridSearchCV(cv=5, error_score='raise',
             estimator=KNeighborsClassifier(n_jobs=1), n_jobs=1,
             param_grid={'metric': ['euclidean', 'manhattan'],
                         'n_neighbors': array([ 1,  2,  3,  4,  5,  6,
7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19])},
             return_train_score=True)
```

```
# Apres avoir afficher les meilleurs scores et distances enregistrons
le meilleur modele sous le nom monModel
print(GS.best_score_)
print(GS.best_params_)
monModel = GS.best_estimator_
monModel.fit(X1, Y1)
```

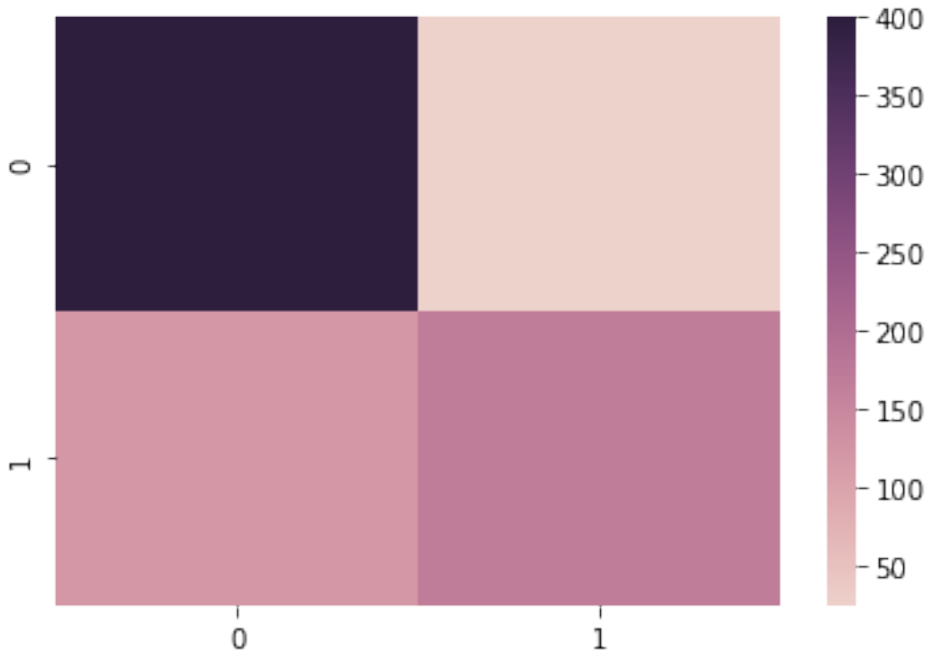
```
0.7793135011441648
{'metric': 'manhattan', 'n_neighbors': 14}
```

```
KNeighborsClassifier(metric='manhattan', n_jobs=1, n_neighbors=14)
```

```
# On peut maintenant afficher la matrice de confusion sur toutes les
donnees
from sklearn.metrics import confusion_matrix
y_true=Y
y_pred=monModel.predict(X)
CM=confusion_matrix(y_true, y_pred)
print(CM)
```

```
[[400  24]
 [120 170]]
```

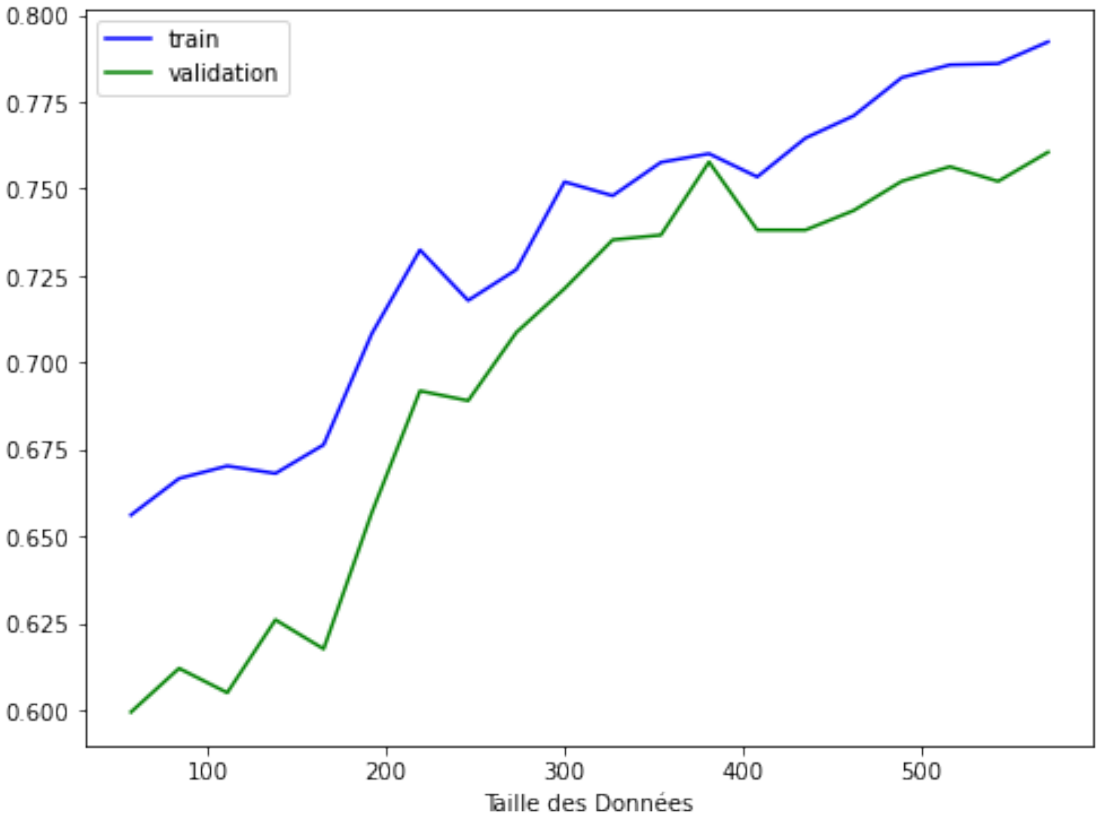
```
# Representons la matrice de confusion grace a heatmap de seaborn afin
de d'avoir une representation visuel des resultats
import seaborn as sns
plt.figure(figsize=(6, 4))
sns.heatmap(CM,cmap=sns.cubehelix_palette(as_cmap=True))
plt.show()
```



```
# On va maintenant afficher grace a learning_curve la courbe
d'apprentissage du modèle monModel
from sklearn.model_selection import learning_curve
train_size_abs, train_scores,
test_scores=learning_curve(monModel.fit(X1, Y1), X,
Y,train_sizes=np.linspace(0.1, 1, 20))
print(train_size_abs)
plt.figure( figsize=(8, 6))
plt.plot(train_size_abs,[i.mean() for i in
train_scores],c='blue',label='train')
plt.plot(train_size_abs,[i.mean() for i in
test_scores],c='green',label='validation')
plt.xlabel('Taille des Données')
plt.legend()
```

```
[ 57  84 111 138 165 192 219 246 273 300 327 354 381 408 435 462 489
516
 543 571]

<matplotlib.legend.Legend at 0x12f61d7e370>
```




```
# enfin utilisons predict_proba pour predire qui d'entre nous pourra
survivre au naufrage du Titanic
predictions = monModel.predict_proba(X)
# la premiere valeur etant 0(survivra pas) et la deuxieme 1(survivra)
```

```
print(predictions[:20])
# si l'on compare les 20 premieres valeur, on voit qu'il y a un
certain manque de precision
print(Y[:20])
```

```
[[0.85714286 0.14285714]
 [0.35714286 0.64285714]
 [0.64285714 0.35714286]
 [0.         1.        ]
 [0.92857143 0.07142857]
 [0.57142857 0.42857143]
 [0.5        0.5       ]
 [0.28571429 0.71428571]
 [0.35714286 0.64285714]
 [0.42857143 0.57142857]
 [0.57142857 0.42857143]
 [0.71428571 0.28571429]
 [0.85714286 0.14285714]
 [0.42857143 0.57142857]
 [0.64285714 0.35714286]
 [0.5        0.5       ]
 [0.57142857 0.42857143]
 [0.64285714 0.35714286]
 [0.57142857 0.42857143]
 [0.5        0.5       ]]
[0 1 1 1 0 0 0 1 1 1 1 0 0 0 1 0 0 0 1 1]
```

```
# Affichons la chance de survivre au naufrage du Titanic
AuriezVousSurvecu=[i[1] for i in predictions]
print(np.mean(AuriezVousSurvecu))
```

```
0.3919567827130852
```