Entrée [2]:

```
pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in c:\users\fanny\anaconda3\lib
\site-packages (0.24.1)
Requirement already satisfied: scipy>=0.19.1 in c:\users\fanny\anaconda3\lib
\site-packages (from scikit-learn) (1.6.1)
Requirement already satisfied: joblib>=0.11 in c:\users\fanny\anaconda3\lib
\site-packages (from scikit-learn) (1.0.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\fanny\anacon
da3\lib\site-packages (from scikit-learn) (2.1.0)
Requirement already satisfied: numpy>=1.13.3 in c:\users\fanny\anaconda3\lib
\site-packages (from scikit-learn) (1.19.2)
Note: you may need to restart the kernel to use updated packages.

WARNING: You are using pip version 21.1.1; however, version 21.3.1 is availa
ble.
You should consider upgrading via the 'C:\Users\fanny\anaconda3\python.exe -
m pip install --upgrade pip' command.
```

Entrée [2]:

```
import sklearn
```

Entrée [3]:

```
import pandas as pd
import numpy as np
```

Entrée [4]:

```
from sklearn import datasets
```

Entrée [5]:

```
from sklearn.model_selection import train_test_split
```

Entrée [6]:

```
from sklearn import preprocessing
```

Entrée [7]:

```
boston = datasets.load_boston()
```

Entrée [8]:

```
boston
```

Out[8]:

```
{'data': array([[6.3200e-03, 1.8000e+01, 2.3100e+00, ..., 1.5300e+01,
3.9690e+02,
         4.9800e+00],
        [2.7310e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9690e+0
2,
         9.1400e+00],
        [2.7290e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9283e+0
2,
         4.0300e+00],
        ...,
        [6.0760e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+0
2,
         5.6400e+00],
        [1.0959e-01, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9345e+0
2,
         6.4800e+00],
        [4.7410e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+0
2,
         7.8800e+00]]),
 'target': array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5,
18.9, 15. ,
        18.9, 21.7, 20.4, 18.2, 19.9, 23.1, 17.5, 20.2, 18.2, 13.6, 19.
6,
        15.2, 14.5, 15.6, 13.9, 16.6, 14.8, 18.4, 21. , 12.7, 14.5, 13.
2,
        13.1, 13.5, 18.9, 20. , 21. , 24.7, 30.8, 34.9, 26.6, 25.3, 24.
7,
        21.2, 19.3, 20. , 16.6, 14.4, 19.4, 19.7, 20.5, 25. , 23.4, 18.
9,
        35.4, 24.7, 31.6, 23.3, 19.6, 18.7, 16. , 22.2, 25. , 33. , 23.
5,
        19.4, 22. , 17.4, 20.9, 24.2, 21.7, 22.8, 23.4, 24.1, 21.4, 20.
,
        20.8, 21.2, 20.3, 28. , 23.9, 24.8, 22.9, 23.9, 26.6, 22.5, 22.
2,
        23.6, 28.7, 22.6, 22. , 22.9, 25. , 20.6, 28.4, 21.4, 38.7, 43.
8,
        33.2, 27.5, 26.5, 18.6, 19.3, 20.1, 19.5, 19.5, 20.4, 19.8, 19.
4,
        21.7, 22.8, 18.8, 18.7, 18.5, 18.3, 21.2, 19.2, 20.4, 19.3, 22.
,
        20.3, 20.5, 17.3, 18.8, 21.4, 15.7, 16.2, 18. , 14.3, 19.2, 19.
6,
        23. , 18.4, 15.6, 18.1, 17.4, 17.1, 13.3, 17.8, 14. , 14.4, 13.
4,
        15.6, 11.8, 13.8, 15.6, 14.6, 17.8, 15.4, 21.5, 19.6, 15.3, 19.
4,
        17. , 15.6, 13.1, 41.3, 24.3, 23.3, 27. , 50. , 50. , 50. , 22.
7,
        25. , 50. , 23.8, 23.8, 22.3, 17.4, 19.1, 23.1, 23.6, 22.6, 29.
4,
        23.2, 24.6, 29.9, 37.2, 39.8, 36.2, 37.9, 32.5, 26.4, 29.6, 50.
,
```

```
        32. , 29.8, 34.9, 37. , 30.5, 36.4, 31.1, 29.1, 50. , 33.3, 30.
3,
        34.6, 34.9, 32.9, 24.1, 42.3, 48.5, 50. , 22.6, 24.4, 22.5, 24.
4,
        20. , 21.7, 19.3, 22.4, 28.1, 23.7, 25. , 23.3, 28.7, 21.5, 23.
,
        26.7, 21.7, 27.5, 30.1, 44.8, 50. , 37.6, 31.6, 46.7, 31.5, 24.
3,
        31.7, 41.7, 48.3, 29. , 24. , 25.1, 31.5, 23.7, 23.3, 22. , 20.
1,
        22.2, 23.7, 17.6, 18.5, 24.3, 20.5, 24.5, 26.2, 24.4, 24.8, 29.
6,
        42.8, 21.9, 20.9, 44. , 50. , 36. , 30.1, 33.8, 43.1, 48.8, 31.
,
        36.5, 22.8, 30.7, 50. , 43.5, 20.7, 21.1, 25.2, 24.4, 35.2, 32.
4,
        32. , 33.2, 33.1, 29.1, 35.1, 45.4, 35.4, 46. , 50. , 32.2, 22.
,
        20.1, 23.2, 22.3, 24.8, 28.5, 37.3, 27.9, 23.9, 21.7, 28.6, 27.
1,
        20.3, 22.5, 29. , 24.8, 22. , 26.4, 33.1, 36.1, 28.4, 33.4, 28.
2,
        22.8, 20.3, 16.1, 22.1, 19.4, 21.6, 23.8, 16.2, 17.8, 19.8, 23.
1,
        21. , 23.8, 23.1, 20.4, 18.5, 25. , 24.6, 23. , 22.2, 19.3, 22.
6,
        19.8, 17.1, 19.4, 22.2, 20.7, 21.1, 19.5, 18.5, 20.6, 19. , 18.
7,
        32.7, 16.5, 23.9, 31.2, 17.5, 17.2, 23.1, 24.5, 26.6, 22.9, 24.
1,
        18.6, 30.1, 18.2, 20.6, 17.8, 21.7, 22.7, 22.6, 25. , 19.9, 20.
8,
        16.8, 21.9, 27.5, 21.9, 23.1, 50. , 50. , 50. , 50. , 50. , 13.
8,
        13.8, 15. , 13.9, 13.3, 13.1, 10.2, 10.4, 10.9, 11.3, 12.3,  8.
8,
         7.2, 10.5,  7.4, 10.2, 11.5, 15.1, 23.2,  9.7, 13.8, 12.7, 13.
1,
        12.5,  8.5,  5. ,  6.3,  5.6,  7.2, 12.1,  8.3,  8.5,  5. , 11.
9,
        27.9, 17.2, 27.5, 15. , 17.2, 17.9, 16.3,  7. ,  7.2,  7.5, 10.
4,
         8.8,  8.4, 16.7, 14.2, 20.8, 13.4, 11.7,  8.3, 10.2, 10.9, 11.
,
         9.5, 14.5, 14.1, 16.1, 14.3, 11.7, 13.4,  9.6,  8.7,  8.4, 12.
8,
        10.5, 17.1, 18.4, 15.4, 10.8, 11.8, 14.9, 12.6, 14.1, 13. , 13.
4,
        15.2, 16.1, 17.8, 14.9, 14.1, 12.7, 13.5, 14.9, 20. , 16.4, 17.
7,
        19.5, 20.2, 21.4, 19.9, 19. , 19.1, 19.1, 20.1, 19.9, 19.6, 23.
2,
        29.8, 13.8, 13.3, 16.7, 12. , 14.6, 21.4, 23. , 23.7, 25. , 21.
8,
        20.6, 21.2, 19.1, 20.6, 15.2,  7. ,  8.1, 13.6, 20.1, 21.8, 24.
5,
        23.1, 19.7, 18.3, 21.2, 17.5, 16.8, 22.4, 20.6, 23.9, 22. , 11.
9]),
 'feature_names': array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'A
GE', 'DIS', 'RAD',
        'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7'),
```

```
 'DESCR': ".. _boston_dataset:\n\nBoston house prices dataset\n--------
-------------------\n\n**Data Set Characteristics:**  \n\n    :Number o
f Instances: 506 \n\n    :Number of Attributes: 13 numeric/categorical
predictive. Median Value (attribute 14) is usually the target.\n\n    :
Attribute Information (in order):\n        - CRIM     per capita crime
rate by town\n        - ZN       proportion of residential land zoned f
or lots over 25,000 sq.ft.\n        - INDUS    proportion of non-retail
business acres per town\n        - CHAS     Charles River dummy variabl
e (= 1 if tract bounds river; 0 otherwise)\n        - NOX      nitric o
xides concentration (parts per 10 million)\n        - RM       average
number of rooms per dwelling\n        - AGE      proportion of owner-oc
cupied units built prior to 1940\n        - DIS      weighted distances
to five Boston employment centres\n        - RAD      index of accessib
ility to radial highways\n        - TAX      full-value property-tax ra
te per $10,000\n        - PTRATIO  pupil-teacher ratio by town\n
- B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by to
wn\n        - LSTAT    % lower status of the population\n        - MEDV
Median value of owner-occupied homes in $1000's\n\n    :Missing Attribu
te Values: None\n\n    :Creator: Harrison, D. and Rubinfeld, D.L.\n\nTh
is is a copy of UCI ML housing dataset.\nhttps://archive.ics.uci.edu/m
l/machine-learning-databases/housing/\n\n\nThis dataset was taken from
the StatLib library which is maintained at Carnegie Mellon Universit
y.\n\nThe Boston house-price data of Harrison, D. and Rubinfeld, D.L.
'Hedonic\nprices and the demand for clean air', J. Environ. Economics &
Management,\nvol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Reg
ression diagnostics\n...', Wiley, 1980.   N.B. Various transformations
are used in the table on\npages 244-261 of the latter.\n\nThe Boston ho
use-price data has been used in many machine learning papers that addre
ss regression\nproblems.   \n     \n.. topic:: References\n\n   - Belsl
ey, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data
and Sources of Collinearity', Wiley, 1980. 244-261.\n   - Quinlan,R. (1
993). Combining Instance-Based and Model-Based Learning. In Proceedings
on the Tenth International Conference of Machine Learning, 236-243, Uni
versity of Massachusetts, Amherst. Morgan Kaufmann.\n",
 'filename': 'C:\\Users\\fanny\\anaconda3\\lib\\site-packages\\sklearn
\\datasets\\data\\boston_house_prices.csv'}
```

Entrée [9]:

```
boston.data
```

Out[9]:

```
array([[6.3200e-03, 1.8000e+01, 2.3100e+00, ..., 1.5300e+01, 3.9690e+02,
        4.9800e+00],
       [2.7310e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9690e+02,
        9.1400e+00],
       [2.7290e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9283e+02,
        4.0300e+00],
       ...,
       [6.0760e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
        5.6400e+00],
       [1.0959e-01, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9345e+02,
        6.4800e+00],
       [4.7410e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
        7.8800e+00]])
```

Entrée [10]:

```
X=boston.data
Y=boston.target
print(X.shape)
```

```
(506, 13)
```

Entrée [11]:

```
X_train, X_test,Y_train,Y_test=train_test_split(X, Y, test_size= 0.2)
```

Entrée [12]:

```
scaler = preprocessing.StandardScaler().fit(X_train)
scaler
```

Out[12]:

```
StandardScaler()
```

Entrée [13]:

```
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Entrée [14]:

```
X_train
```

Out[14]:

```
array([[-0.42621026,  0.6130065 , -0.88421061, ...,  0.59095155,
         0.4149821 , -0.43424386],
       [ 0.27990093, -0.48407256,  0.99885759, ...,  0.81897871,
         0.41619392,  0.54963599],
       [-0.40614552, -0.48407256, -0.55754111, ...,  0.54534612,
         0.36595854, -0.12339889],
       ...,
       [-0.42435827, -0.48407256,  0.39197825, ..., -1.09644937,
         0.42952392,  0.44555843],
       [-0.43820806, -0.48407256, -1.31251062, ...,  0.13489725,
         0.4045164 , -1.34457557],
       [-0.41742955, -0.48407256, -0.0595516 , ...,  0.3629244 ,
         0.21216776, -0.70623321]])
```

Entrée [15]:

```
Y_train
```

Out[15]:

```
array([25. , 18.4, 18.5, 19.6, 23.7, 19.5, 18.5, 16.1, 27.1, 24.4, 15.6,
       13.1, 44. , 22.8, 18.4, 19.4, 25.2, 23.6, 17.1, 21.4,  7. , 21.6,
       21.7, 13.8,  6.3, 23.8, 32. , 15.2, 37.6, 23.2, 19.1, 19.8, 22.5,
       23.9, 48.5, 18.9, 19.5, 42.8, 15.4, 10.5, 15. , 17.8, 15.2, 20.1,
       19.9, 25. , 18. , 11.8, 14.4, 20.8, 13. , 27.5, 14.6, 27. , 17.2,
       32.9, 31. , 24.3, 22.2, 24.3, 22.8, 20.8, 18.2, 14.1, 13.9, 21.9,
        8.3, 21.8, 19.1, 20.2, 22.1, 23.9, 16.7, 32.2, 35.2, 29.4, 19.6,
       20.9, 23. , 41.7, 13.4, 24. , 13.4, 22.9, 26.4, 50. , 23.1, 28.7,
       46.7, 29. , 21.9, 30.7, 22. , 15.2, 17. , 14.5, 50. , 23.2, 34.6,
       14.3, 22.3, 23.1, 17.1,  5. , 16.8, 16.7, 17.8, 13.1, 30.1, 38.7,
       17.8, 36.1, 34.9, 34.7, 11.7, 21.7, 23.3, 18.9, 29.6, 14. ,  9.7,
       17.1, 27.9, 31.2, 37. , 16.5, 21.1, 22.6, 18.3, 24.6, 12.3, 24.4,
       28.2, 23.2, 17.5, 25.1, 17.5, 19.6, 25. , 43.1, 19.1, 50. , 19.8,
       29.1, 24.7, 21.5, 11. , 32. , 23.9, 50. , 18.7, 24.4, 21.2, 20.5,
       20. , 21.4, 19.4, 21.7, 50. , 16.1, 13.3, 16.1, 24.8, 18.5, 17.9,
       22.8, 18.2, 30.1, 26.6, 28.7, 17.8,  5. , 36.2,  7.4,  8.1, 15.1,
       20.2, 22. , 14.2, 22.7, 23.9, 12.1, 21.4, 18.7, 37.3, 11.9, 36.4,
       22.2, 26.2, 21.5, 22.5, 50. , 17.5, 21.7, 20.3, 19.7, 19.8, 28.4,
        7. , 13.8, 19.1, 18.2, 23.8, 15.6, 35.4, 20.3, 14.6, 23.9, 10.2,
       46. , 19.9, 42.3, 50. , 22.9, 33.2, 18.9, 16.8, 10.9, 33.4, 22.4,
       20.9, 13.3, 23.2, 33.8, 20. , 20.6, 50. , 11.8,  7.5, 16.2, 22.8,
       23.7, 21.8, 18.6, 14.1, 24.5, 12.7, 22.3, 13.8, 33. , 15.6, 20.1,
       10.5, 22. , 21.7, 15.6, 48.8,  8.4, 23.1, 13.3, 25. , 10.8,  7.2,
       28.4, 25. , 17.2,  8.7, 29.1, 36.2, 21.9, 15. , 29.8, 23. , 19.4,
       27.5, 20.8, 24.7, 22.6, 33.3, 16.5, 22.2, 50. , 21. , 21.2, 19.3,
       22.2, 23.3, 18.1, 23.3, 10.2, 33.1, 18.8, 23.4, 19.2, 21.7, 39.8,
       13.1, 23. , 31.5, 26.5, 20.7, 17.4, 21.1, 10.4, 21.2, 32.7, 20.6,
       23.7, 22.6, 19.5, 22. , 12. , 11.7, 13.6, 20. , 13.4, 22.6, 20. ,
       15.6, 14.8, 18.5, 21.2,  8.5, 50. , 20.3, 50. , 50. , 23.7, 27.5,
       24.4, 20.6, 36. , 35.1, 20.1, 48.3, 41.3, 29. , 15.7, 22.9, 25. ,
       22.9,  8.4, 11.5, 24. , 19.3, 30.3, 25. , 21. , 24.1, 18.9, 45.4,
       22.4,  8.8, 21.2, 22. , 23.6, 50. , 14.9, 20. , 24.1, 18.6, 19.4,
       32.5, 19.4, 20.4, 50. , 16. , 28.7, 20.1, 19.9,  7.2, 19.7, 15. ,
       23. , 23.1, 28.5,  9.5, 13.5, 24.8, 21.4, 22.7, 24.6, 22. , 20.6,
       19.3, 17.7, 13.5, 19.2, 11.9, 20.5, 13.8, 31.5, 23.1, 20.6, 14.4,
       17.2, 12.5, 43.5, 34.9, 44.8,  8.3, 19.3, 50. , 17.4,  5.6, 37.2,
       20.1, 21.7, 24.3, 26.6, 18.7, 20.3, 33.4, 23.4])
```

Entrée [16]:

```
from sklearn import svm
regressor = svm.SVR(kernel="linear")
```

Entrée [17]:

```
from sklearn.model_selection import cross_val_score
cross_val_score(regressor,X_train,Y_train,n_jobs=-1)
```

Out[17]:

```
array([0.65099354, 0.70964232, 0.7552304 , 0.64817343, 0.55902178])
```

Entrée [18]:

```
from sklearn.model_selection import GridSearchCV
parameters = {'gamma':[0.01,0.1,0.5]}
grid=GridSearchCV(svm.SVR(),parameters,n_jobs=-1,cv=5)
grid.fit(X_train,Y_train)
print(grid.best_score_,grid.best_estimator_)
```

```
0.5845482809789289 SVR(gamma=0.1)
```

Entrée [19]:

```
parameters = {'C':[0.5,1,1.5], 'gamma':[0.5,0.1,0.15]}
grid=GridSearchCV(svm.SVR(),parameters,n_jobs=-1)
grid.fit(X_train,Y_train)
print(grid.best_score_,grid.best_estimator_)
```

```
0.6411486753085509 SVR(C=1.5, gamma=0.1)
```

Entrée [20]:

```
parameters = {'C':[2.5,2,1.5], 'gamma':[0.5,0.1,0.15],"kernel":["rbf","poly","sigmoid"]}
grid=GridSearchCV(svm.SVR(),parameters,n_jobs=-1)
grid.fit(X_train,Y_train)
print(grid.best_score_,grid.best_estimator_)
```

```
0.7294842313558243 SVR(C=2.5, gamma=0.15, kernel='poly')
```

Entrée [21]:

```
parameters = {'C':[4,2,3], "degree":[1,3,5],'gamma':[0.5,0.1,0.15],"kernel":["rbf","poly","
grid=GridSearchCV(svm.SVR(),parameters,n_jobs=-1)
grid.fit(X_train,Y_train)
print(grid.best_score_,grid.best_estimator_)

#this is the best model with svr
```

```
0.7393377334922759 SVR(C=4, gamma=0.15, kernel='poly')
```

Entrée [38]:

```
import seaborn as sns
import random
```

Entrée [130]:

```
#loading data
data= pd.read_csv('train_titanic.csv')
#test=pd.read_csv("test_titanic.csv")
```

Entrée [111]:

```
data.columns
```

Out[111]:

```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

Entrée [116]:

```
data["Embarked"].unique()
```

Out[116]:

```
array(['S', 'C', 'Q', nan], dtype=object)
```

Entrée [131]:

```
data=data.drop(["Name","Ticket","PassengerId","Cabin"],axis=1)
data.loc[data["Sex"]=="male","Sex"]=0
data.loc[data["Sex"]=="female","Sex"]=1
data.loc[data["Embarked"]=="S","Embarked"]=0
data.loc[data["Embarked"]=="C","Embarked"]=1
data.loc[data["Embarked"]=="Q","Embarked"]=2
data.dropna(how='any',axis=0)
data = data.dropna(subset=['Age'])
data = data.dropna(subset=['Embarked'])
```

Entrée [132]:

```
data.isna().sum()
```

Out[132]:

```
Survived    0
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    0
dtype: int64
```

Entrée [133]:

```
train = data[:2*(int(len(data)/3))]#on crée les données d'apprentissage
test = data[2*(int((len(data)/3))):]
```

Entrée [135]:

```
test
```

Out[135]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
603	0	3	0	44.0	0	0	8.0500	0
604	1	1	0	35.0	0	0	26.5500	1
605	0	3	0	36.0	1	0	15.5500	0
606	0	3	0	30.0	0	0	7.8958	0
607	1	1	0	27.0	0	0	30.5000	0
...	...	...	...	...	...	...	...	...
885	0	3	1	39.0	0	5	29.1250	2
886	0	2	0	27.0	0	0	13.0000	0
887	1	1	1	19.0	0	0	30.0000	0
889	1	1	0	26.0	0	0	30.0000	1
890	0	3	0	32.0	0	0	7.7500	2

238 rows × 8 columns

Entrée [136]:

```
X_train = train.drop(['Survived'],axis=1)
Y_train = train["Survived"]
X_test = test.drop(['Survived'],axis=1)
Y_test = test["Survived"]
```

Entrée [137]:

```
X_train
```

Out[137]:

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	3	0	22.0	1	0	7.2500	0
1	1	1	38.0	1	0	71.2833	1
2	3	1	26.0	0	0	7.9250	0
3	1	1	35.0	1	0	53.1000	0
4	3	0	35.0	0	0	8.0500	0
...	...	...	...	...	...	...	...
594	2	0	37.0	1	0	26.0000	0
595	3	0	36.0	1	1	24.1500	0
597	3	0	49.0	0	0	0.0000	0
599	1	0	49.0	1	0	56.9292	1
600	2	1	24.0	2	1	27.0000	0

474 rows × 7 columns

Entrée [138]:

```
Y_train
```

Out[138]:

```
0      0
1      1
2      1
3      1
4      0
      ..
594    0
595    0
597    0
599    1
600    1
Name: Survived, Length: 474, dtype: int64
```

Entrée [ ]:

```
from sklearn.model_selection import StratifiedKFold,cross_val_score,RandomizedSearchCV,Grid
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifier,GradientBoostingClas
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import f1_score,recall_score,precision_score,roc_auc_score
from xgboost import XGBClassifier

# split train set and validation set
stk = StratifiedKFold(n_splits=N_SPLITS,random_state=SEED,shuffle=True)
for train_index,val_index in stk.split(X,y):
    X_train,y_train = X.iloc[train_index],y.iloc[train_index]
    X_val,y_val = X.iloc[val_index],y.iloc[val_index]


# build model
knn = KNeighborsClassifier()
knn.fit(X_train,y_train)
pred_knn = knn.predict(X_val)
print("knn recall:",round(recall_score(y_val,pred_knn),2))
print("knn precision:",round(precision_score(y_val,pred_knn),2))
print("knn f1_score:",round(f1_score(y_val,pred_knn),2))
print("knn rou_auc_score:",round(roc_auc_score(y_val,pred_knn),2))

# optimize model
params = {'algorithm': ['auto'],
          'weights': ['uniform', 'distance'],
          'leaf_size': range(1,30),
          'n_neighbors': range(3,20)}
gs =GridSearchCV(knn, param_grid = params,
                 verbose=True,
                 cv=GRID_SEARCH_CV_NUM,
                 scoring = SCORING
                )
gs.fit(X_train, y_train)
print(gs.best_score_)
print(gs.best_estimator_)
print(gs.best_params_)

# evaluate model
pred_val = gs.predict(X_val)
print("gs recall:",round(recall_score(y_val,pred_val),2))
print("gs precision:",round(precision_score(y_val,pred_val),2))
print("gs f1_score:",round(f1_score(y_val,pred_val),2))
print("gs rou_auc_score:",round(roc_auc_score(y_val,pred_val),2))
```

```
##Avec un knn
```

Entrée [144]:

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train,Y_train)
pred_knn = knn.predict(X_test)
```

Entrée [145]:

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(Y_test, pred_knn))
print(classification_report(Y_test, pred_knn))
```

```
[[118  28]
 [ 39  53]]
              precision    recall  f1-score   support

           0       0.75      0.81      0.78       146
           1       0.65      0.58      0.61        92

    accuracy                           0.72       238
   macro avg       0.70      0.69      0.70       238
weighted avg       0.71      0.72      0.71       238
```

Entrée [149]:

```
)[1,0]+confusion_matrix(Y_test, pred_knn)[1,1]+confusion_matrix(Y_test, pred_knn)[0,0])
```

Out[149]:

```
0.7184873949579832
```

Entrée [154]:

```
import statistics
```

Entrée [156]:

```
tabk=[]
k=[]
for i in range(2,11):#les k sont testé de 2 à 10 car il est très peu probable que k>10
    testsduk=[]
    for j in range(6):#on test cette valeur 10 fois
        knn = KNeighborsClassifier(n_neighbors=i)
        knn.fit(X_train,Y_train)
        pred_knn = knn.predict(X_test)

        testsduk.append((confusion_matrix(Y_test, pred_knn)[0,0]+confusion_matrix(Y_test, p

    tabk.append(testsduk)
stat=[[statistics.mean(tabk[j]),statistics.pvariance(tabk[j]),j+2] for j in range(len(tabk)
k=sorted(stat, reverse = True)
  #  print(k)

if abs(k[0][0]-k[1][0])<0.0002: #si deux modele sont très proche en performance on prefere
    if k[0][1]>k[1][1]:
           # print(k)
        print(k[1][2])
    else :
           # print(k)
        print(k[0][2])
else :
       # print(k)
        print(k[0][2])
print(k)
```

```
7
[[0.7394957983193278, 0.0, 7], [0.7226890756302521, 0.0, 6], [0.718487394957
9832, 0.0, 5], [0.7142857142857143, 0.0, 8], [0.7100840336134454, 0.0, 10],
[0.7016806722689075, 0.0, 9], [0.6932773109243697, 0.0, 3], [0.6848739495798
319, 0.0, 4], [0.6596638655462185, 0.0, 2]]
```

Entrée [ ]:

```
#the best k for titanic knn is 7 with a performance of 73% accuracy
```

Entrée [158]:

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import f1_score,recall_score,precision_score,roc_auc_score
gbdt = GradientBoostingClassifier(random_state=123)
gbdt.fit(X_train,Y_train)
gbdt_pre = gbdt.predict(X_test)
print("gbdt recall:",round(recall_score(Y_test,gbdt_pre),2))
print("gbdt precision:",round(precision_score(Y_test,gbdt_pre),2))
print("gbdt f1_score:",round(f1_score(Y_test,gbdt_pre),2))
print("gbdt rou_auc_score:",round(roc_auc_score(Y_test,gbdt_pre),2))
```

```
gbdt recall: 0.71
gbdt precision: 0.79
gbdt f1_score: 0.75
gbdt rou_auc_score: 0.8
```

Entrée [ ]:

```
#boosting has 79% of accuracy, it's a better model than knn
```

Entrée [160]:

```
from sklearn.linear_model import LogisticRegression
logisticRegr = LogisticRegression()
logisticRegr.fit(X_train, Y_train)
y_pred = logisticRegr.predict(X_test)
print(confusion_matrix(Y_test, y_pred))
print(classification_report(Y_test, y_pred))
```

```
[[129  17]
 [ 28  64]]
              precision    recall  f1-score   support

           0       0.82      0.88      0.85       146
           1       0.79      0.70      0.74        92

    accuracy                           0.81       238
   macro avg       0.81      0.79      0.80       238
weighted avg       0.81      0.81      0.81       238


C:\Users\fanny\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.p
y:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scik
it-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regre
ssion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression)
  n_iter_i = _check_optimize_result(
```

Entrée [161]:

```
(confusion_matrix(Y_test, y_pred)[0,0]+confusion_matrix(Y_test, y_pred)[1,1])/(confusion_ma
```

Out[161]:

```
0.8109243697478992
```

Entrée [ ]:

```
#logistic regression is the best model, with an accuracy of 81%
```