Лабораторна робота №1

**Тема:** ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ ДАНИХ

***Мета:***використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити попередню обробку та класифікацію даних*.*

Хід роботи

**Репозиторій: https://github.com/FrancIwanicki/OAI.git**

**Завдання 2.1:** Попередня обробка даних.

Лістинг програми:

import numpy as np

from sklearn import preprocessing

input\_data = np.array([[5.1, -2.9, 3.3],

[-1.2, 7.8, -6.1],

[3.9, 0.4, 2.1],

[7.3, -9.9, -4.5]])

data\_binarized = preprocessing.Binarizer(threshold=2.1).transform(input\_data)

print("\n Binarized data:\n", data\_binarized)

# Виведення середнього значення та стандартного відхилення

print("\nДо: ")

print("Середнє =", input\_data.mean(axis=0))

print("Середньоквадратичне =", input\_data.std(axis=0))

# Виключення середнього

data\_scaled = preprocessing.scale(input\_data)

print("\nПісля: ")

print("Середнє =", data\_scaled.mean(axis=0))

print("Std deviation =", data\_scaled.std(axis=0))

#Масштабування

data\_scaler\_minmax = preprocessing.MinMaxScaler(feature\_range=(0, 1))

data\_scaled\_minmax = data\_scaler\_minmax.fit\_transform(input\_data)

print("\nМin max scaled data:\n", data\_scaled\_minmax)

#Нормалізація даних

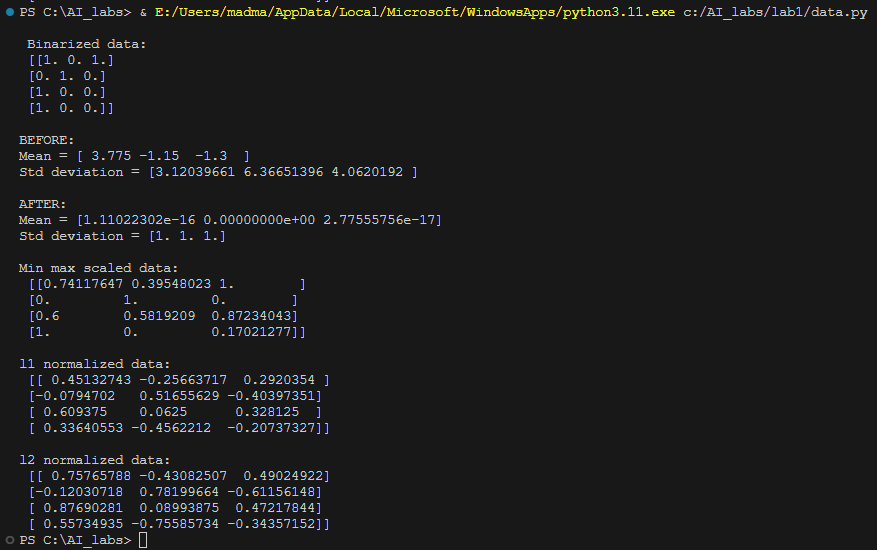
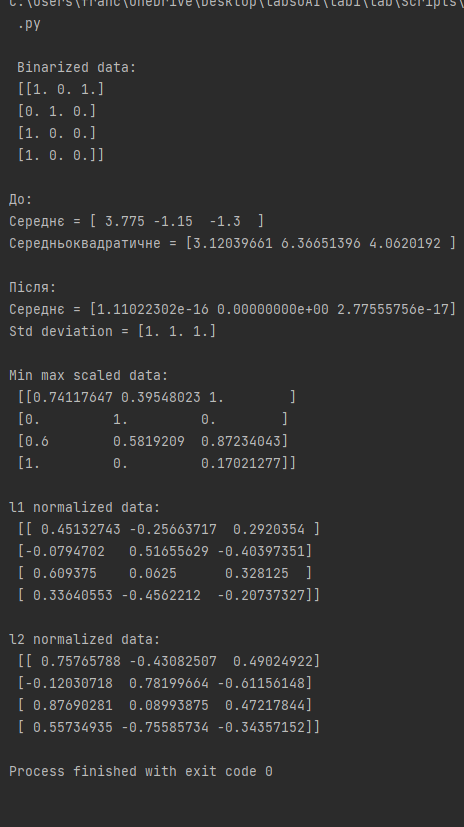
data\_normalized\_l1 = preprocessing.normalize(input\_data, norm='l1')

data\_normalized\_l2 = preprocessing.normalize(input\_data, norm='l2')

print("\nl1 normalized data:\n", data\_normalized\_l1)

print("\nl2 normalized data:\n", data\_normalized\_l2)

Результат роботи програми:

Рисунок 1

**Висновок:** L1-нормалізація та L2-нормалізація - це методи нормалізації даних, які використовуються для перетворення векторів або матриць в одиничні вектори з різними підходами до обчислення норми (або довжини) вектора.

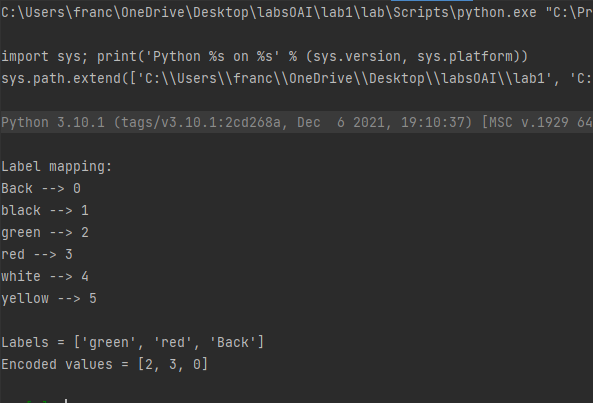
* У L1-нормалізації норма визначається як сума абсолютних значень всіх елементів вектора (або рядка матриці), що розраховується за формулою ||x||₁ = |x₁| + |x₂| + ... + |xₙ|. Цей метод чутливий до величини та знаку кожного окремого елемента вектора.
* У L2-нормалізації норма визначається як квадратний корінь з суми квадратів всіх елементів вектора (або рядка матриці), що розраховується за формулою ||x||₂ = √(x₁² + x₂² + ... + xₙ²). Цей метод менше чутливий до величини окремих елементів та акцентує загальну кількість енергії вектора.

**Завдання 2.1.5:** Кодування міток.

Лістинг програми:

import numpy as np  
from sklearn import preprocessing  
  
# Надання позначок вхідних даних  
input\_labels = ['red', 'Back', 'red', 'green', 'black', 'yellow', 'white']  
  
# Створення кодувальника та встановлення відповідності  
# між мітками та числами  
encoder = preprocessing.LabelEncoder()  
encoder.fit(input\_labels)  
  
# Виведення відображення  
print("\nLabel mapping:")  
for i, item in enumerate(encoder.classes\_):  
 print(item, '-->', i)  
  
# перетворення міток за допомогою кодувальника  
test\_labels = ['green', 'red', 'Back']  
encoded\_values = encoder.transform(test\_labels )  
print("\nLabels =", test\_labels )  
print("Encoded values =", list (encoded\_values ) )

Результат виконання програми:



**Завдання 2.2:** Попередня обробка нових даних

Варіант 3:

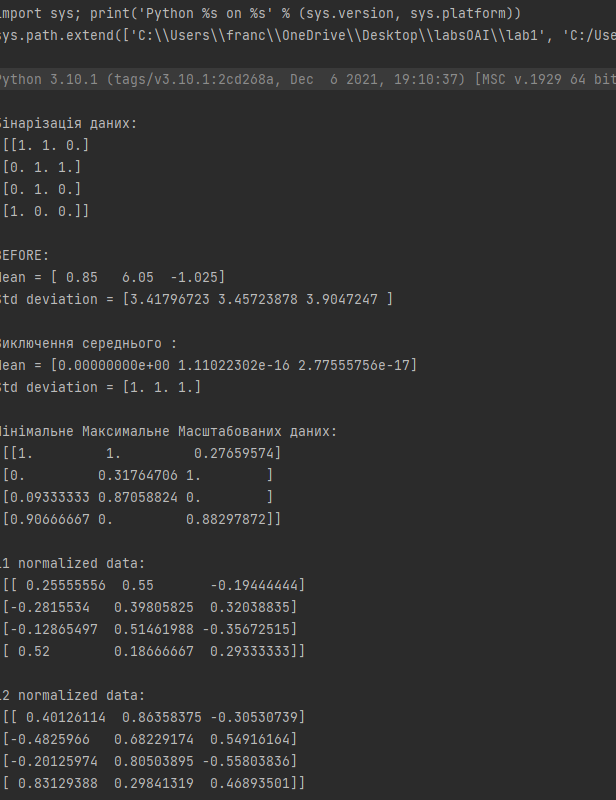




Лістинг програми:

import numpy as np  
from sklearn import preprocessing  
input\_data = np.array([  
 [4.6, 9.9, -3.5],  
 [-2.9, 4.1, 3.3],  
 [-2.2, 8.8, -6.1],  
 [3.9, 1.4, 2.2]])  
# Бінаризація даних  
data\_binarized = preprocessing.Binarizer(threshold=2.2).transform(input\_data)  
print("\nБінарізація даних:\n", data\_binarized)  
  
# Виведення середнього значення та стандартного відхилення  
print("\nBEFORE: ")  
print("Mean =", input\_data.mean(axis=0))  
print("Std deviation =", input\_data.std(axis=0))  
  
# Виключення середнього  
print("\nВиключення середнього :")  
data\_scaled = preprocessing.scale(input\_data)  
print("Mean =", data\_scaled.mean(axis=0))  
print("Std deviation =", data\_scaled.std(axis=0))  
  
# Масштабування MinМax  
data\_scaler\_minmax = preprocessing.MinMaxScaler(feature\_range=(0, 1))  
data\_scaled\_minmax = data\_scaler\_minmax.fit\_transform(input\_data)  
print("\nМінімальне Максимальне Масштабованих даних:\n", data\_scaled\_minmax)  
  
#Нормалізація  
data\_normalized\_l1 = preprocessing.normalize(input\_data, norm='l1')  
data\_normalized\_l2 = preprocessing.normalize(input\_data, norm='l2')  
print("\nl1 normalized data:\n", data\_normalized\_l1)  
print("\nl2 normalized data:\n", data\_normalized\_l2)

Результат виконання програми:



**Завдання 2.3:** Класифікація логістичною регресією або логістичний класифікатор

Лістинг програми:

import numpy as np

from sklearn import linear\_model

import matplotlib.pyplot as plt

from utilities import visualize\_classifier

# Визначення зразка вхідних даних

X = np.array([[3.1, 7.2], [4, 6.7], [2.9, 8], [5.1, 4.5],

[6, 5], [5.6, 5], [3.3, 0.4],

[3.9, 0.9], [2.8, 1],

[0.5, 3.4], [1, 4], [0.6, 4.9]])

y = np.array([0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3])

# Створення логістичного класифікатора

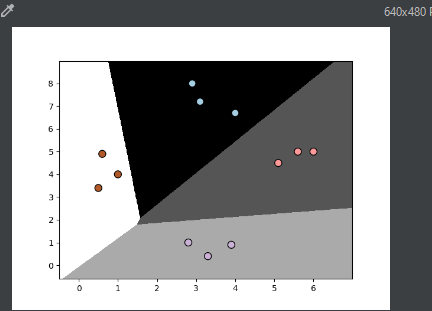
classifier = linear\_model.LogisticRegression(solver='liblinear',C=1)

# Тренування класифікатора

classifier.fit(X, y)

visualize\_classifier(classifier, X, y)

Результат виконання програми:



**Завдання 2.4:** Класифікація наївним байєсовським класифікатором.

Лістинг програми:

import numpy as np

import matplotlib.pyplot as plt

from sklearn.naive\_bayes import GaussianNB

from sklearn.model\_selection import train\_test\_split

from utilities import visualize\_classifier

# Вхідний файл, який містить дані

input\_file = 'data\_multivar\_nb.txt'

# Завантаження даних із вхідного файлу

data = np.loadtxt(input\_file, delimiter=',')

X, y = data[:, :-1], data[:, -1]

# Створення наївного байєсовського класифікатора

classifier = GaussianNB()

# Тренування класифікатора

classifier.fit(X, y)

# Прогнозування значень для тренувальних даних

y\_pred = classifier.predict(X)

# Обчислення якості класифікатора

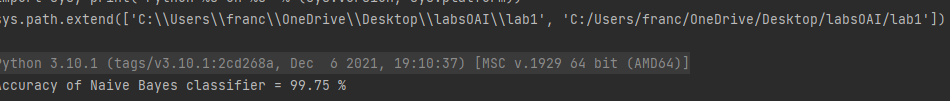
accuracy = 100.0 \* (y == y\_pred).sum() / X.shape[0]

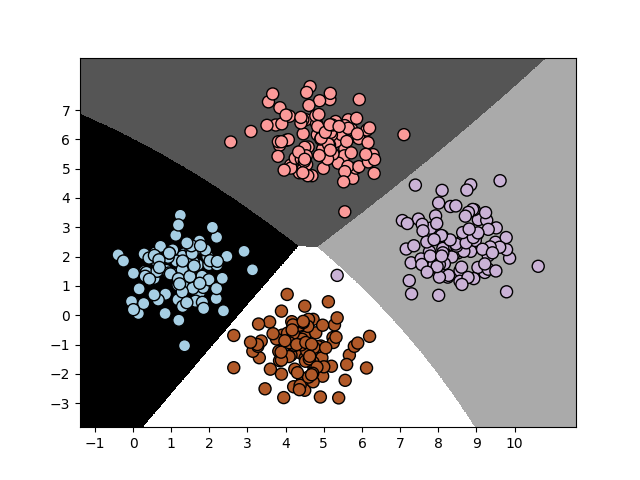
print("Accuracy of Naive Bayes classifier =", round(accuracy, 2), "%")

# Візуалізація результатів роботи класифікатора

visualize\_classifier(classifier, X, y)

Результат виконання програми:





Лістинг програми після розбиття даних на навчальний та тестовий набори:

import pandas as pd

import numpy as np

df = pd.read\_csv('data\_metrics.csv')

df.head()

thresh = 0.5

df['predicted\_RF'] = (df.model\_RF >= 0.5).astype('int')

df['predicted\_LR'] = (df.model\_LR >= 0.5).astype('int')

df.head()

from sklearn.metrics import confusion\_matrix, recall\_score

confusion\_matrix(df.actual\_label.values, df.predicted\_RF.values)

def find\_TP(y\_true, y\_pred):

return sum((y\_true == 1) & (y\_pred == 1))

def find\_FN(y\_true, y\_pred):

return sum((y\_true == 1) & (y\_pred == 0))

def find\_FP(y\_true, y\_pred):

return sum((y\_true == 0) & (y\_pred == 1))

def find\_TN(y\_true, y\_pred):

return sum((y\_true == 0) & (y\_pred == 0))

def find\_conf\_matrix\_values(y\_true,y\_pred):

TP = find\_TP(y\_true,y\_pred)

FN = find\_FN(y\_true,y\_pred)

FP = find\_FP(y\_true,y\_pred)

TN = find\_TN(y\_true,y\_pred)

return TP,FN,FP,TN

def ivanytskyi\_confusion\_matrix(y\_true, y\_pred):

TP,FN,FP,TN = find\_conf\_matrix\_values(y\_true,y\_pred)

return np.array([[TN,FP],[FN,TP]])

ivanytskyi\_confusion\_matrix(df.actual\_label.values,df.predicted\_RF.values)

# Перевірка результатів

print('TP:', find\_TP(df.actual\_label.values, df.predicted\_RF.values))

print('FN:', find\_FN(df.actual\_label.values, df.predicted\_RF.values))

print('FP:', find\_FP(df.actual\_label.values, df.predicted\_RF.values))

print('TN:', find\_TN(df.actual\_label.values, df.predicted\_RF.values))

assert np.array\_equal(ivanytskyi\_confusion\_matrix(df.actual\_label.values,

df.predicted\_RF.values), confusion\_matrix(df.actual\_label.values,

df.predicted\_RF.values) ), 'ivanytskyi\_confusion\_matrix() is not correct for RF'

assert np.array\_equal(ivanytskyi\_confusion\_matrix(df.actual\_label.values,

df.predicted\_LR.values),confusion\_matrix(df.actual\_label.values,

df.predicted\_LR.values) ), 'ivanytskyi\_confusion\_matrix() is not correct for LR'

# accuracy\_score

from sklearn.metrics import accuracy\_score

accuracy\_score(df.actual\_label.values, df.predicted\_RF.values)

def ivanytskyi\_accuracy\_score(y\_true, y\_pred):

TP,FN,FP,TN = find\_conf\_matrix\_values(y\_true,y\_pred)

accuracy = (TP + TN) / (TP + TN + FP + FN)

return accuracy

assert ivanytskyi\_accuracy\_score(df.actual\_label.values, df.predicted\_RF.values) == accuracy\_score(df.actual\_label.values, df.predicted\_RF.values),'ivanytskyi\_accuracy\_score failed on RF'

assert ivanytskyi\_accuracy\_score(df.actual\_label.values, df.predicted\_LR.values) == accuracy\_score(df.actual\_label.values, df.predicted\_LR.values),'ivanytskyi\_accuracy\_score failed on LR'

print('Accuracy RF: %.2f' % (ivanytskyi\_accuracy\_score(df.actual\_label.values,

df.predicted\_RF.values)))

print('Accuracy LR: %.2f' % (ivanytskyi\_accuracy\_score(df.actual\_label.values,

df.predicted\_LR.values)))

# accuracy recall\_score

from sklearn.metrics import accuracy\_score

accuracy\_score(df.actual\_label.values, df.predicted\_RF.values)

def ivanytskyi\_recall\_score(y\_true, y\_pred):

TP, FN, FP, TN = find\_conf\_matrix\_values(y\_true, y\_pred)

recall = TP / (TP + FN)

return recall

assert ivanytskyi\_recall\_score(df.actual\_label.values, df.predicted\_RF.values) == ivanytskyi\_recall\_score(df.actual\_label.values, df.predicted\_RF.values), 'ivanytskyi\_recall\_score failed on RF'

assert ivanytskyi\_recall\_score(df.actual\_label.values, df.predicted\_LR.values) == ivanytskyi\_recall\_score(df.actual\_label.values, df.predicted\_LR.values), 'ivanytskyi\_recall\_score failed on LR'

print('Recall RF: %.3f' % (ivanytskyi\_recall\_score(df.actual\_label.values,df.predicted\_RF.values)))

print('Recall LR: %.3f' % (ivanytskyi\_recall\_score(df.actual\_label.values,df.predicted\_LR.values)))

# precision\_score

from sklearn.metrics import precision\_score

precision\_score(df.actual\_label.values, df.predicted\_RF.values)

def ivanytskyi\_precision\_score(y\_true, y\_pred):

TP, FN, FP, TN = find\_conf\_matrix\_values(y\_true, y\_pred)

precision = TP / (TP + FP)

return precision

assert ivanytskyi\_precision\_score(df.actual\_label.values, df.predicted\_LR.values) ==precision\_score(df.actual\_label.values, df.predicted\_LR.values),'my\_accuracy\_score failed on LR'

print('Precision RF: %.3f' % (ivanytskyi\_precision\_score(df.actual\_label.values,df.predicted\_RF.values)))

print('Precision LR: %.3f' % (ivanytskyi\_precision\_score(df.actual\_label.values,df.predicted\_LR.values)))

# f1\_score

from sklearn.metrics import f1\_score

f1\_score(df.actual\_label.values, df.predicted\_RF.values)

def ivanytskyi\_f1\_score(y\_true, y\_pred):

# calculates the F1 score

recall = ivanytskyi\_recall\_score(y\_true, y\_pred)

precision = ivanytskyi\_precision\_score(y\_true, y\_pred)

f1\_score = (2 \* (precision \* recall)) / (precision + recall)

return f1\_score

assert ivanytskyi\_f1\_score(df.actual\_label.values, df.predicted\_LR.values) == f1\_score(df.actual\_label.values, df.predicted\_LR.values), 'my\_accuracy\_scorefailed on LR'

print('F1 RF: %.3f'%(ivanytskyi\_f1\_score(df.actual\_label.values,df.predicted\_RF.values)))

print('F1 LR: %.3f'%(ivanytskyi\_f1\_score(df.actual\_label.values,df.predicted\_LR.values)))

print('scores with threshold = 0.5')

print('Accuracy RF: %.3f'%(ivanytskyi\_accuracy\_score(df.actual\_label.values,df.predicted\_RF.values)))

print('Recall RF: %.3f'%(ivanytskyi\_recall\_score(df.actual\_label.values,df.predicted\_RF.values)))

print('Precision RF: %.3f'%(ivanytskyi\_precision\_score(df.actual\_label.values,df.predicted\_RF.values)))

print('F1 RF: %.3f'%(ivanytskyi\_f1\_score(df.actual\_label.values,df.predicted\_RF.values)))

print('')

print('scores with threshold = 0.25')

print('Accuracy RF: %.3f'%(ivanytskyi\_accuracy\_score(df.actual\_label.values,(df.model\_RF >= 0.25).astype('int').values)))

print('Recall RF: %.3f'%(ivanytskyi\_recall\_score(df.actual\_label.values, (df.model\_RF>= 0.25).astype('int').values)))

print('Precision RF: %.3f'%(ivanytskyi\_precision\_score(df.actual\_label.values,(df.model\_RF >= 0.25).astype('int').values)))

print('F1 RF: %.3f'%(ivanytskyi\_f1\_score(df.actual\_label.values, (df.model\_RF >=0.25).astype('int').values)))

from sklearn.metrics import roc\_curve

import matplotlib.pyplot as plt

fpr\_RF, tpr\_RF, thresholds\_RF = roc\_curve(df.actual\_label.values,df.model\_RF.values)

fpr\_LR, tpr\_LR, thresholds\_LR = roc\_curve(df.actual\_label.values,df.model\_LR.values)

plt.plot(fpr\_RF, tpr\_RF, 'r-', label='RF')

plt.plot(fpr\_LR, tpr\_LR, 'b-', label='LR')

plt.plot([0, 1], [0, 1], 'k-', label='random')

plt.plot([0, 0, 1, 1], [0, 1, 1, 1], 'g-', label='perfect')

plt.legend()

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.show()

from sklearn.metrics import roc\_auc\_score

import matplotlib.pyplot as plt

auc\_RF = roc\_auc\_score(df.actual\_label.values, df.model\_RF.values)

auc\_LR = roc\_auc\_score(df.actual\_label.values, df.model\_LR.values)

print('AUC RF: %.3f' % auc\_RF)

print('AUC LR: %.3f' % auc\_LR)

plt.plot(fpr\_RF, tpr\_RF, 'r-', label='RF AUC: %.3f' % auc\_RF)

plt.plot(fpr\_LR, tpr\_LR, 'b-', label='LR AUC: %.3f' % auc\_LR)

plt.plot([0, 1], [0, 1], 'k-', label='random')

plt.plot([0, 0, 1, 1], [0, 1, 1, 1], 'g-', label='perfect')

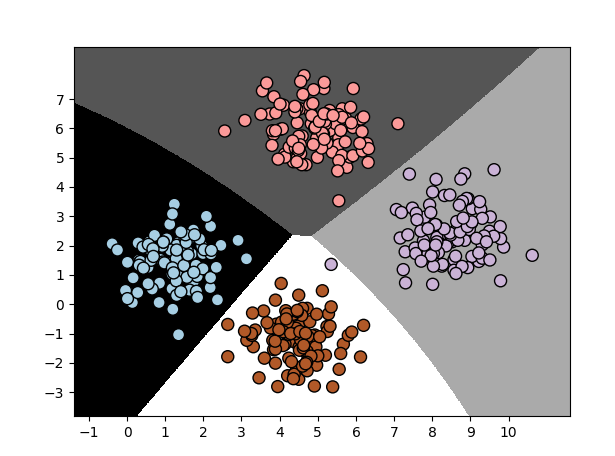
plt.legend()

plt.xlabel('False Positive Rate')

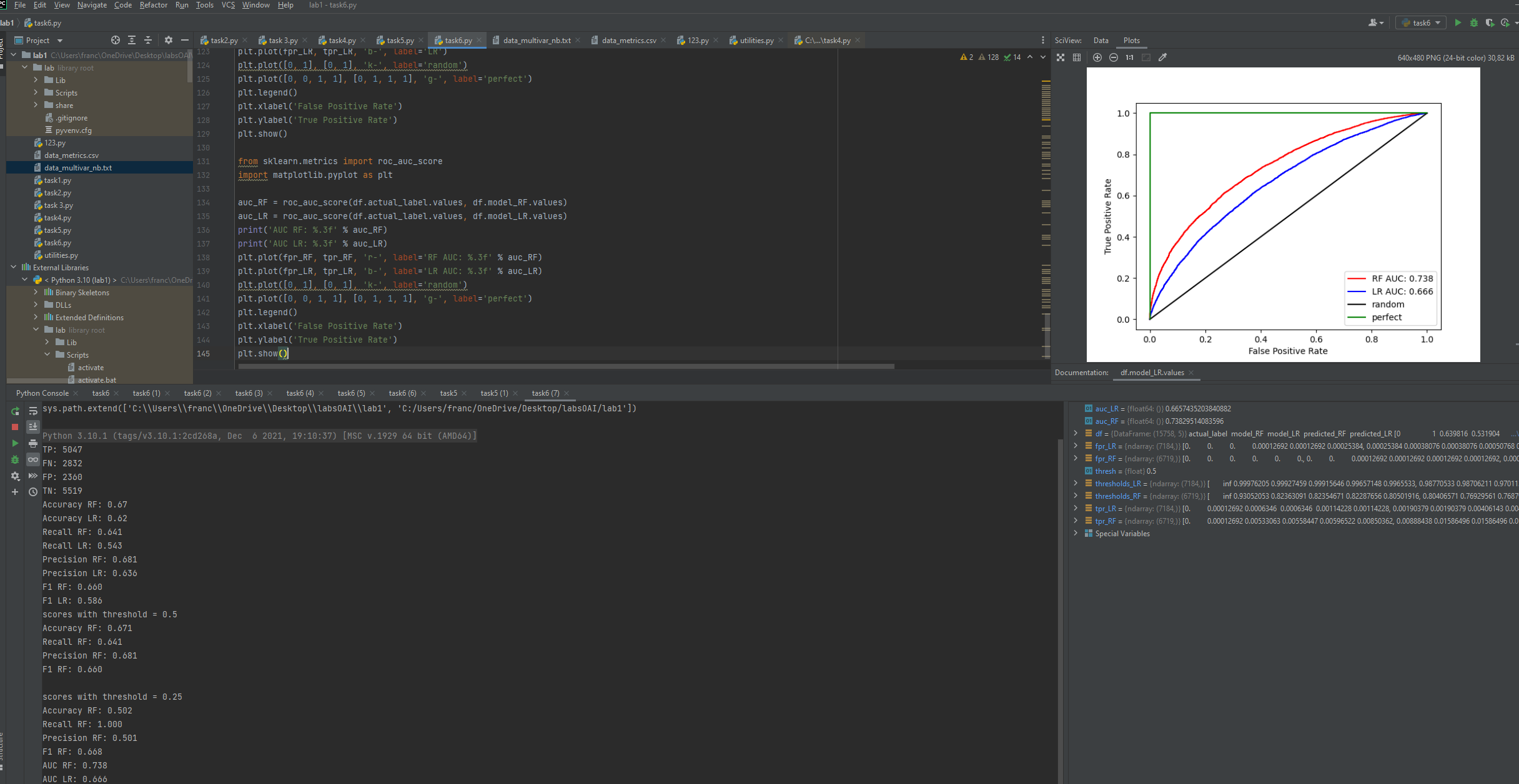
plt.ylabel('True Positive Rate')

plt.show()

Результат виконання програми:



Результати класифікації:



**Завдання №2.7**

Лістинг програми:

from sklearn.metrics import roc\_curve

import matplotlib.pyplot as plt

from task6 import df

fpr\_RF, tpr\_RF, thresholds\_RF = roc\_curve(df.actual\_label.values,df.model\_RF.values)

fpr\_LR, tpr\_LR, thresholds\_LR = roc\_curve(df.actual\_label.values,df.model\_LR.values)

plt.plot(fpr\_RF, tpr\_RF, 'r-', label='RF')

plt.plot(fpr\_LR, tpr\_LR, 'b-', label='LR')

plt.plot([0, 1], [0, 1], 'k-', label='random')

plt.plot([0, 0, 1, 1], [0, 1, 1, 1], 'g-', label='perfect')

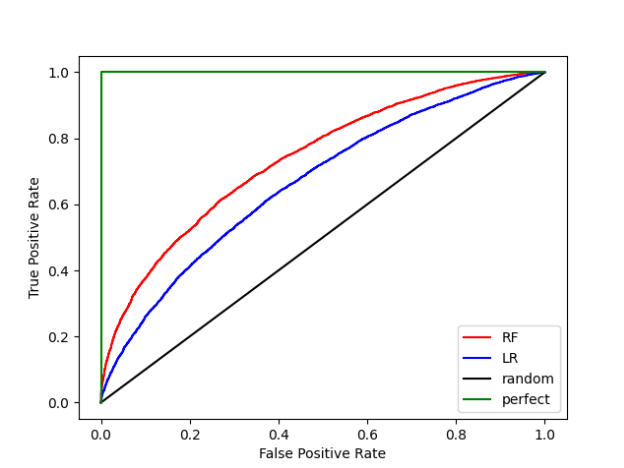
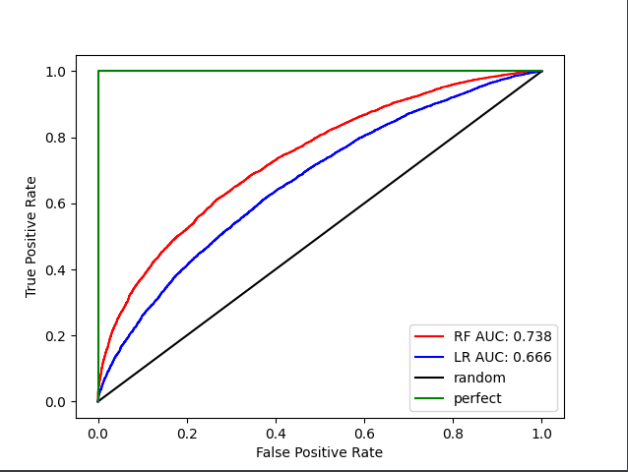
plt.legend()

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.show()

Результат виконання програми:

 ****

**Висновок:** Розділення набору даних на навчальну і тестову вибірки є ключовим етапом для оцінки ефективності моделі на нових даних, які раніше не були використані для її навчання. Це дозволяє визначити, наскільки добре модель узагальнює дані та уникає перенавчання. Додатково, модифікований підхід використовує крос-валідацію, що дозволяє отримати більш об'єктивні метрики ефективності моделі. Це досягається оцінкою моделі на кількох різних підвибірках даних, що допомагає зменшити вплив випадковості при розділенні на навчальний і тестовий набори. Такий підхід сприяє більш надійній оцінці здатності моделі до узагальнення та зниженню можливості виникнення викривлених результатів.