# ЛАБОРАТОРНА РОБОТА № 1
# ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ ДАНИХ

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

**Завдання 2.1.1 Бінарізація.**

```
Binarized data:
[[1. 0. 1.]
[0. 1. 0.]
[1. 0. 0.]
[1. 0. 0.]]
```

Рис.1 Результат виконання завдання 2.1.1

**Завдання 2.1.2 Виключення середнього.**

```
BEFORE:
Mean = [ 3.775 -1.15  -1.3  ]
Std deviation = [3.12039661 6.36651396 4.0620192 ]

AFTER:
Mean = [1.11022302e-16 0.00000000e+00 2.77555756e-17]
Std deviation = [1. 1. 1.]
```

Рис.2 Результат виконання завдання 2.1.2

**Завдання 2.1.3 Масштабування.**

```
Min max scaled data:
 [[0.74117647 0.39548023 1.        ]
 [0.         1.         0.        ]
 [0.6        0.5819209  0.87234043]
 [1.         0.         0.17021277]]
```

Рис.3 Результат виконання завдання 2.1.3

**Завдання 2.1.4 Нормалізація.**

```
L1 normalized data:
 [[ 0.45132743 -0.25663717  0.2920354 ]
 [-0.0794702   0.51655629 -0.40397351]
 [ 0.609375    0.0625      0.328125  ]
 [ 0.33640553 -0.4562212  -0.20737327]]

L2 normalized data:
 [[ 0.75765788 -0.43082507  0.49024922]
 [-0.12030718  0.78199664 -0.61156148]
 [ 0.87690281  0.08993875  0.47217844]
 [ 0.55734935 -0.75585734 -0.34357152]]
```

Рис.4 Результат виконання завдання 2.1.4

```python
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=3.0).transform(input_data)
print("\n Binarized data:\n", data_binarized)
print("\nBEFORE: ")
print("Mean =", input_data.mean(axis=0))
print("Std deviation =", input_data.std(axis=0))
data_scaled = preprocessing.scale(input_data)
print("\nAFTER: ")
print("Mean =", data_scaled.mean(axis=0))
print("Std deviation =", data_scaled.std(axis=0))
data_scaler_minmax = preprocessing.MinMaxScaler(feature_range=(0, 1))
data_scaled_minmax = data_scaler_minmax.fit_transform(input_data)
print("\nMin max scaled data:\n", data_scaled_minmax)
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)
```

```
 Binarized data:
 [[1. 0. 1.]
 [0. 1. 0.]
 [1. 0. 0.]
 [1. 0. 0.]]

BEFORE:
Mean = [ 3.775 -1.15  -1.3  ]
Std deviation = [3.12039661 6.36651396 4.0620192 ]

AFTER:
Mean = [1.11022302e-16 0.00000000e+00 2.77555756e-17]
Std deviation = [1. 1. 1.]

Min max scaled data:
 [[0.74117647 0.39548023 1.        ]
 [0.         1.         0.        ]
 [0.6        0.5819209  0.87234043]
 [1.         0.         0.17021277]]

L1 normalized data:
 [[ 0.45132743 -0.25663717  0.2920354 ]
 [-0.0794702   0.51655629 -0.40397351]
 [ 0.609375    0.0625      0.328125  ]
 [ 0.33640553 -0.4562212  -0.20737327]]

L2 normalized data:
 [[ 0.75765788 -0.43082507  0.49024922]
 [-0.12030718  0.78199664 -0.61156148]
 [ 0.87690281  0.08993875  0.47217844]
 [ 0.55734935 -0.75585734 -0.34357152]]
```

Рис. 5 - 6 Результат виконання завдання 2.1.1 – 2.1.4

Як бачимо, L1-нормалізація менш чутлива до викидів.

**Завдання 2.1.5 Кодування міток.**

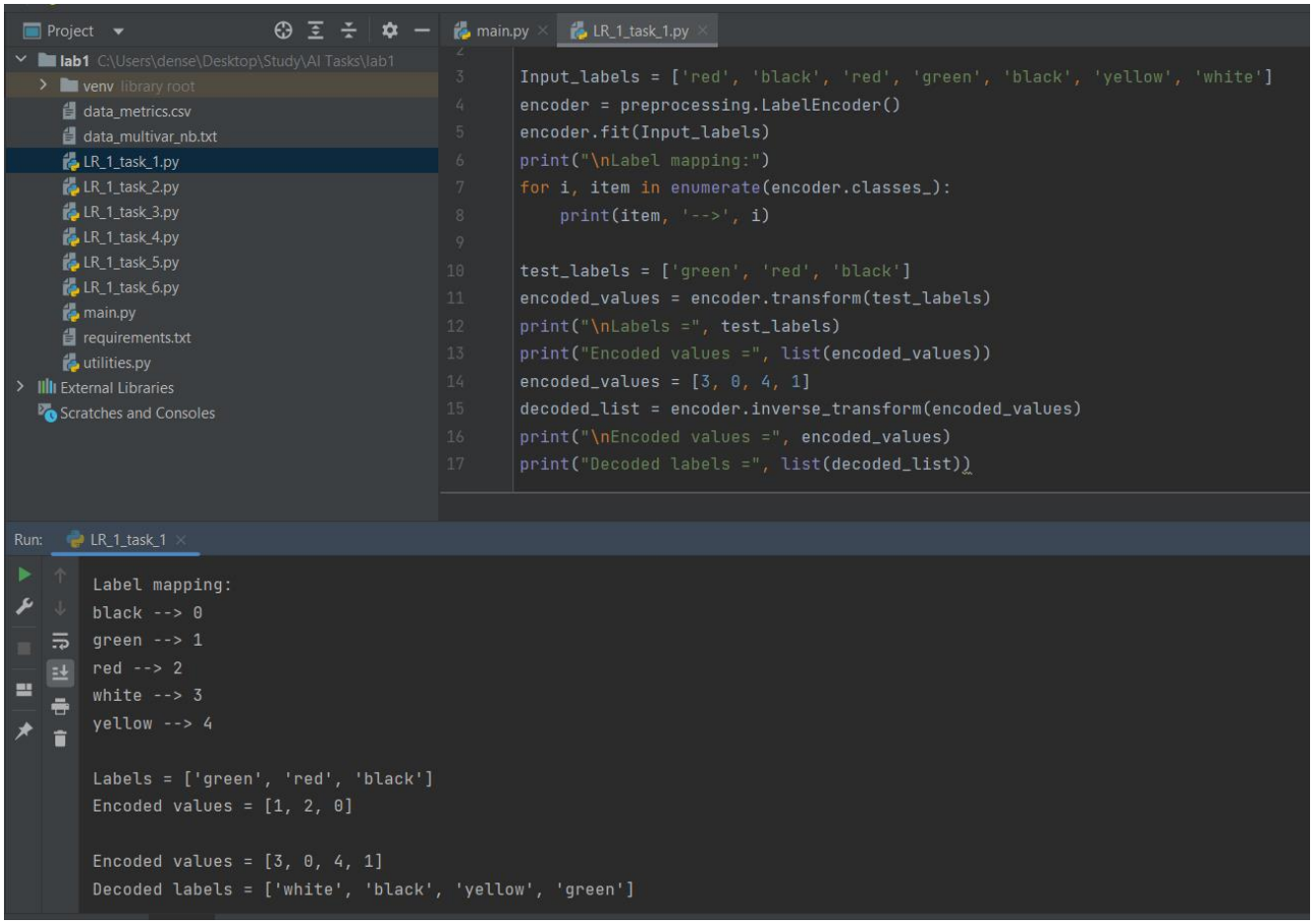



Рис. 7. Результат виконання завдання 2.1.5

**Завдання 2.2 Попередня обробка нових даних.**

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4. | -5.3 | -8.9 | 3.0 | 2.9 | 5.1 | -3.3 | 3.1 | -2.8 | -3.2 | 2.2 | -1.4 | 5.1 | 3.0 |

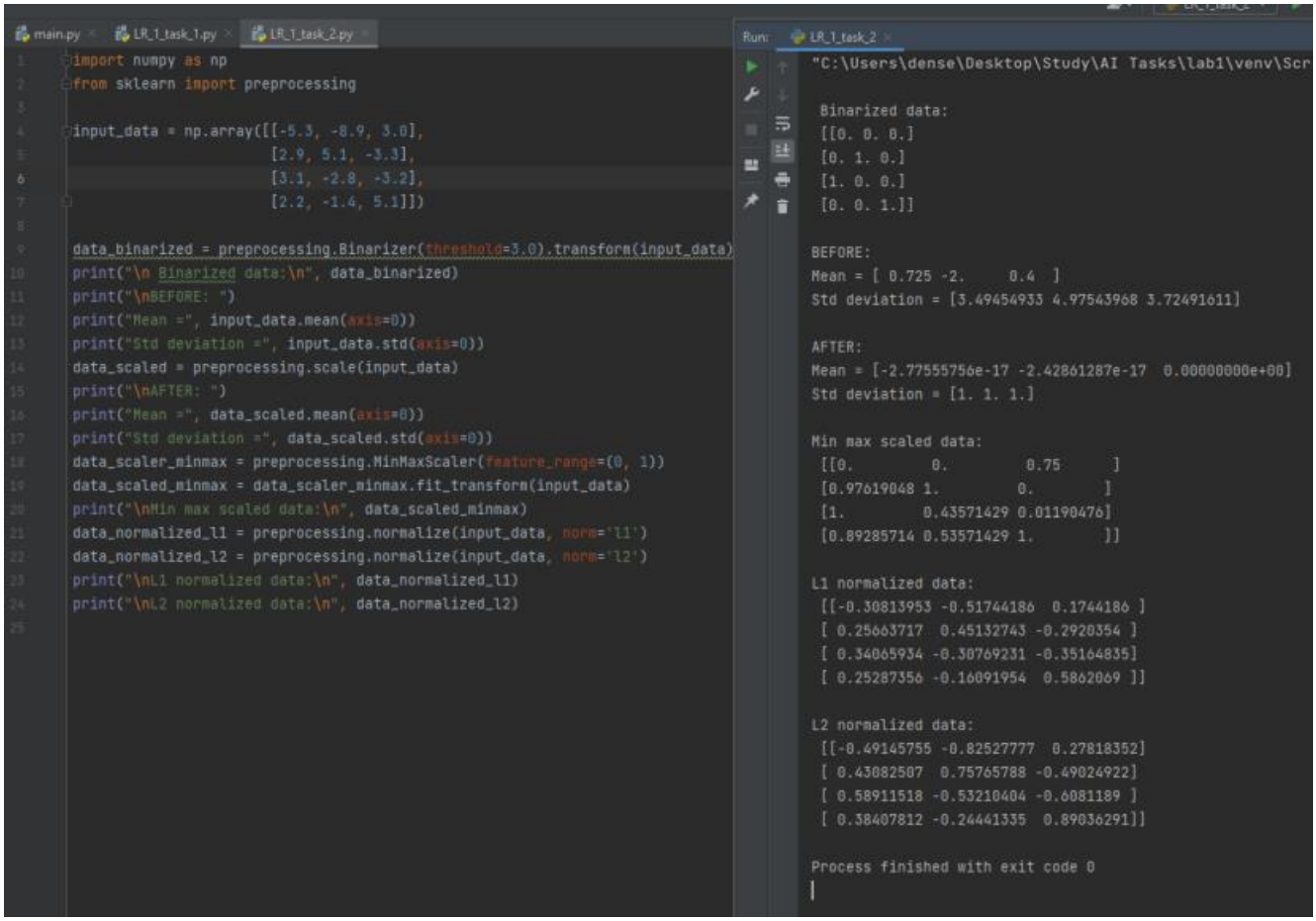



Рис.8. Результат виконання завдання 2.2

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

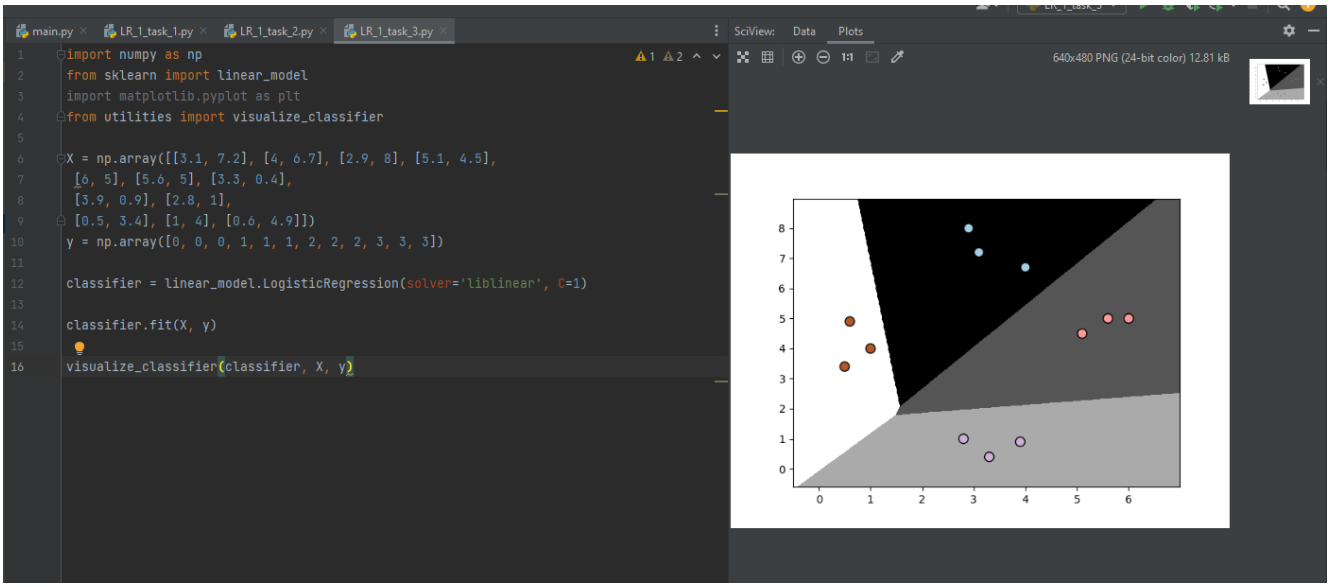
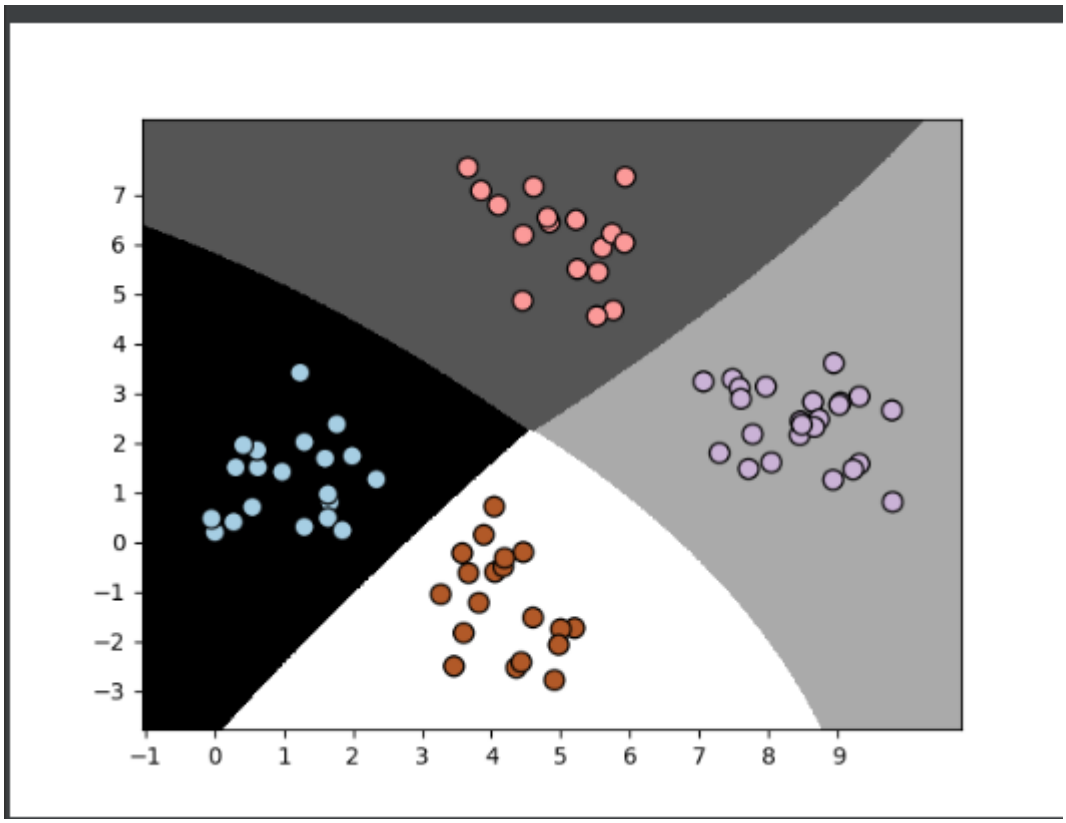



Рис.9 Результат виконання завдання 2.3

**Завдання 2.4 Класифікація наївним байєсовським класифікатором.**

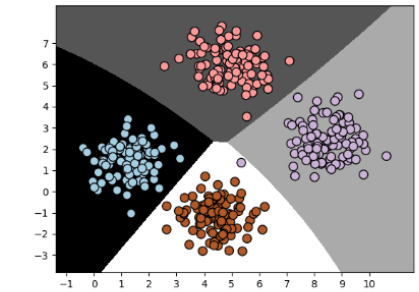



LR_1_task_4.py | utilities.py

```python
import numpy as np
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
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)

# Розбивка даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=3)
classifier_new = GaussianNB()
classifier_new.fit(X_train, y_train)
y_test_pred = classifier_new.predict(X_test)
```

SciView: Data Plots

640x480 PNG (24-bit color) 67.76 kB



```
Accuracy of Naive Bayes classifier = 99.75 %
Accuracy of the new classifier = 100.0 %
Accuracy: 99.75%
Precision: 99.76%
Recall: 99.75%
F1: 99.75%
```

Рис. 10 - 12. Результат виконання завдання 2.4

**Завдання 2.5 Вивчити метрики якості класифікації.**

```
[[5519 2360]
 [2832 5047]]
TP: 5047
FN: 2832
FP: 2360
TN: 5519
0.6705165630156111
Accuracy RF:0.671
0.6405635232897576
Recall RF: 0.641
Recall LR: 0.543
Precision RF: 0.681
Precision LR: 0.636
F1 RF: 0.660
F1 LR: 0.586
scores with threshold = 0.5
Accuracy RF:  0.671
Recall RF: 0.641
Precision RF:  0.681
F1 RF: 0.660

Scores with threshold = 0.75
Accuracy RF:  0.512
Recall RF: 0.025
Precision RF: 0.995
F1 RF: 0.049
AUC RF:0.738
AUC LR:0.666
```

Рис.13 - 15 Результат виконання завдання 2.5

З результатів бачимо, що при збільшенні порогу F1 міра зменшується.

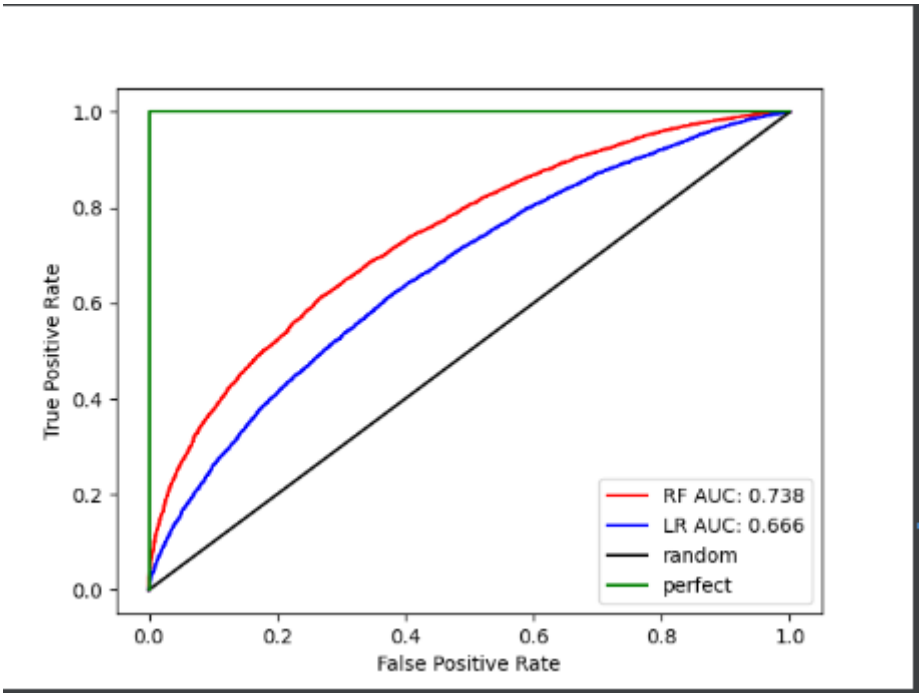
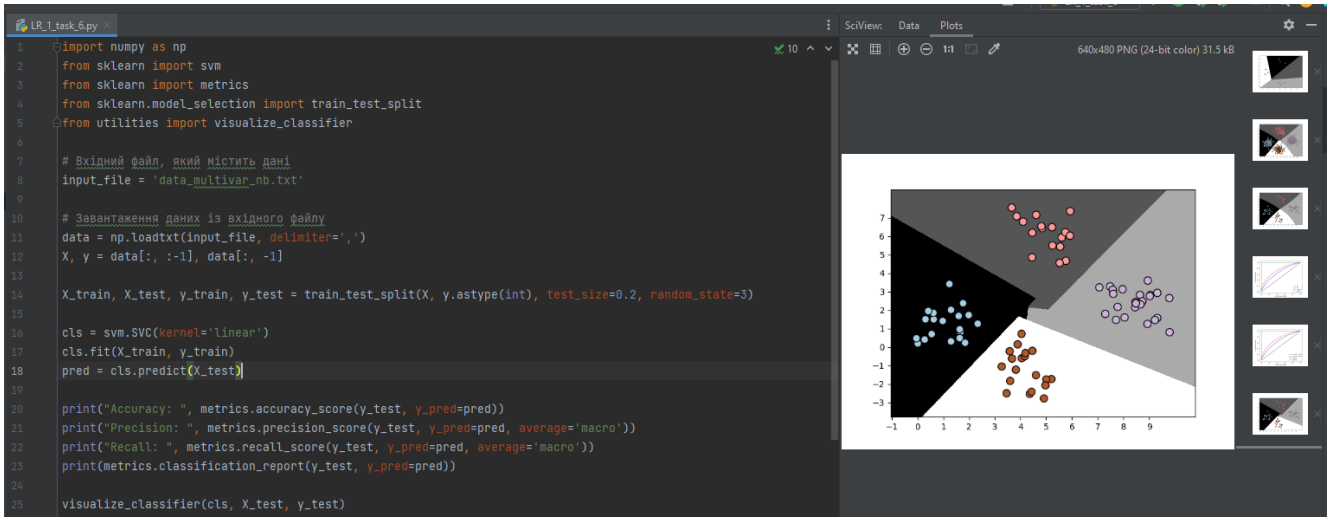



Рис.16 ROC - крива

На графіку бачимо, що RF модель виглядає краще, ніж LR модель , площа під кривою для моделі RF (AUC = 0,738) краще, ніж LR (AUC = 0,666). Проте ефективність кожного з методів залежить від конкретної моделі.

**Завдання 2.6 Розробіть програму класифікації даних в файлі data_multivar_nb.txt за допомогою машини опорних векторів (Support Vector**

**Machine - SVM). Розрахуйте показники якості класифікації. Порівняйте їх з показниками наївного байєсівського класифікатора. Зробіть висновки яку модель класифікації краще обрати і чому.**



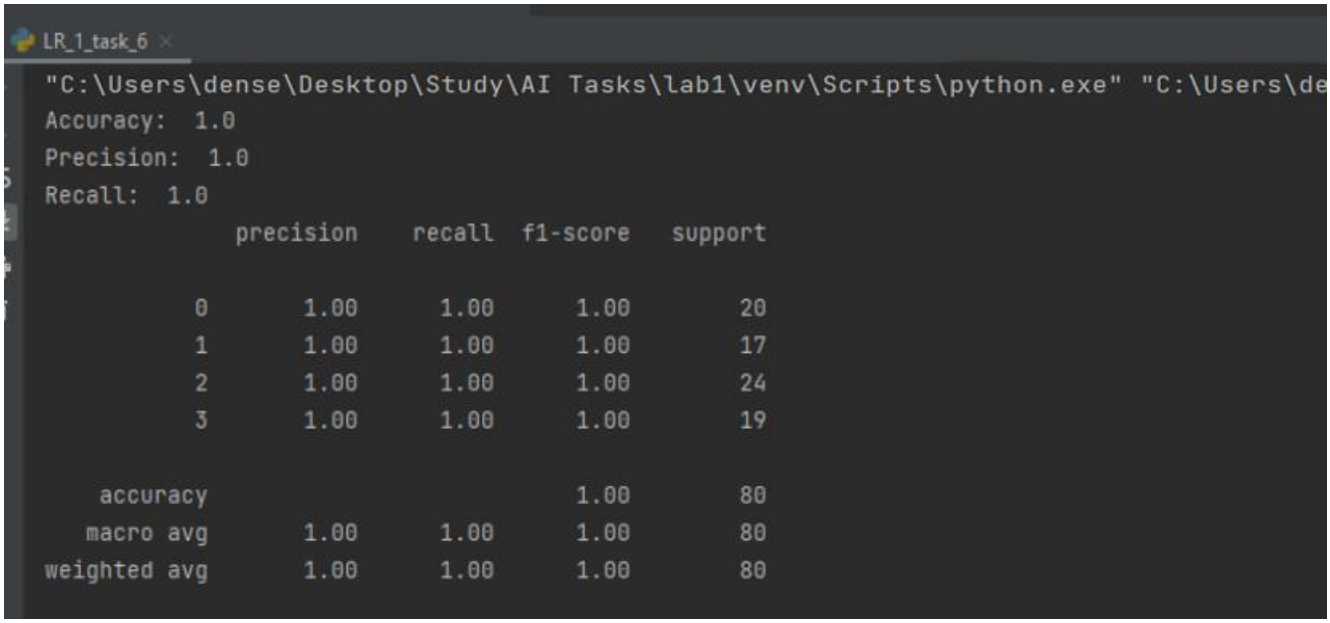




Рис.17 - 18 Результат виконання завдання 2.6

Наївний баєсівський класифікатор і метод опорних векторів (SVM) мають різні параметри, включаючи вибір функції ядра для кожного з них. Обидва алгоритми є дуже чутливими до оптимізації параметрів, тому вибір різних параметрів може суттєво вплинути на отримані результати. Для обраних параметрів NBC працює краще, ніж SVM, що видно в результатах. Проте, за інших параметрів, SVM може показати більш ефективні результати.

**Репозиторій:** https://github.com/SerhiiDenysiuk23/AI_Labs

**Висновок:** в ході виконання лабораторної роботи використовуючи спеціалізовані бібліотеки та мову програмування Python було досліджено попередню обробку та класифікацію даних.