**ЛАБОРАТОРНА РОБОТА № 1**

**ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ**

**ДАНИХ**

***Мета роботи:*** *використовуючи спеціалізовані бібліотеки та мову*

*програмування Python дослідити попередню обробку та класифікацію даних*.

**Посилання на репозиторій:** [**https://github.com/D1nqq/AI\_Labs\_Baginskiy**](https://github.com/D1nqq/AI_Labs_Baginskiy)



Рис.1 Скріншот коду

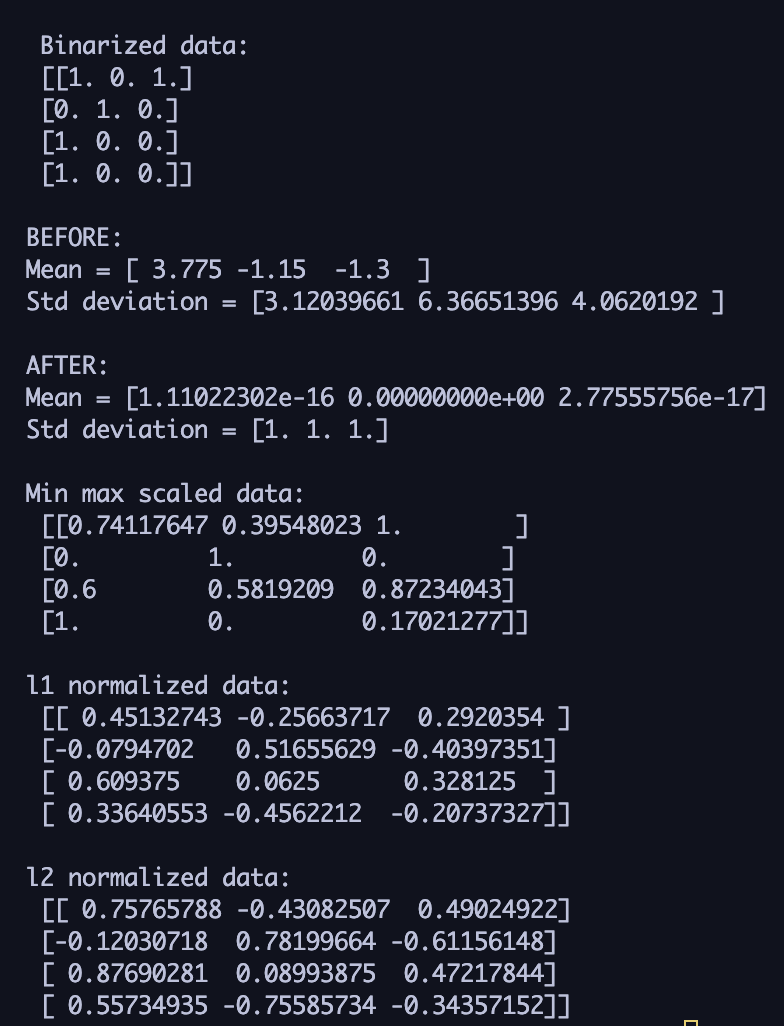


Рис.2 Результат виконання

**Висновок**:

L1-нормалізація зменшує значення на основі абсолютної суми, надаючи розрідженість даних і зберігаючи структуру великих і малих значень. L2-нормалізація зменшує значення на основі суми квадратів і більше фокусується на рівномірному масштабуванні всіх елементів.

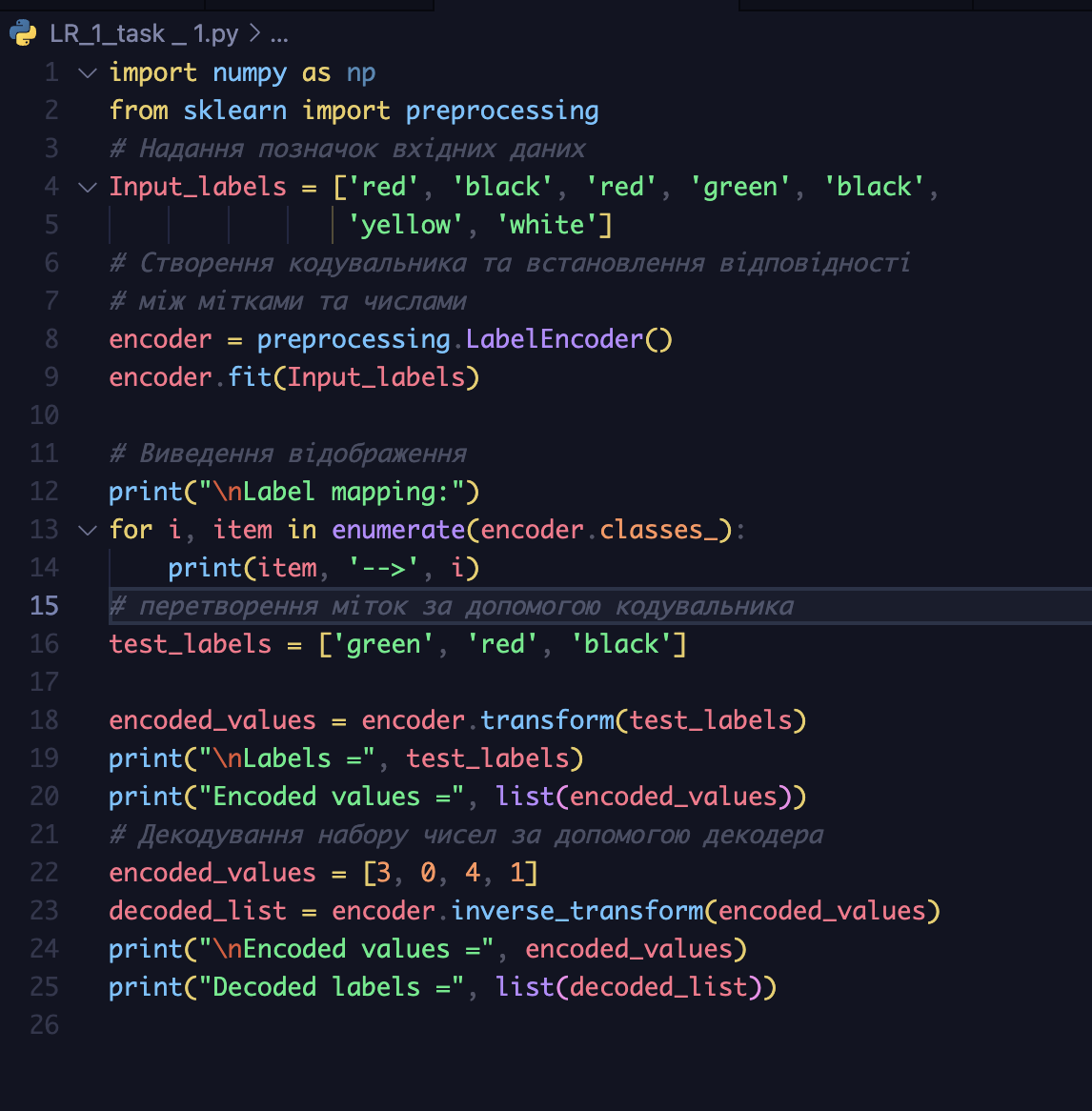


Рис.3. Скрін коду до завдання 1

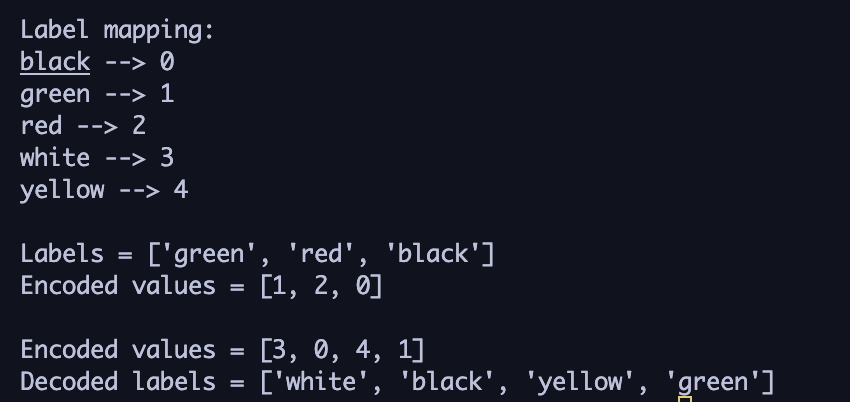


Рис.4. Результат виконання завдання 1



Рис.5. Скрін коду до 2 завдання

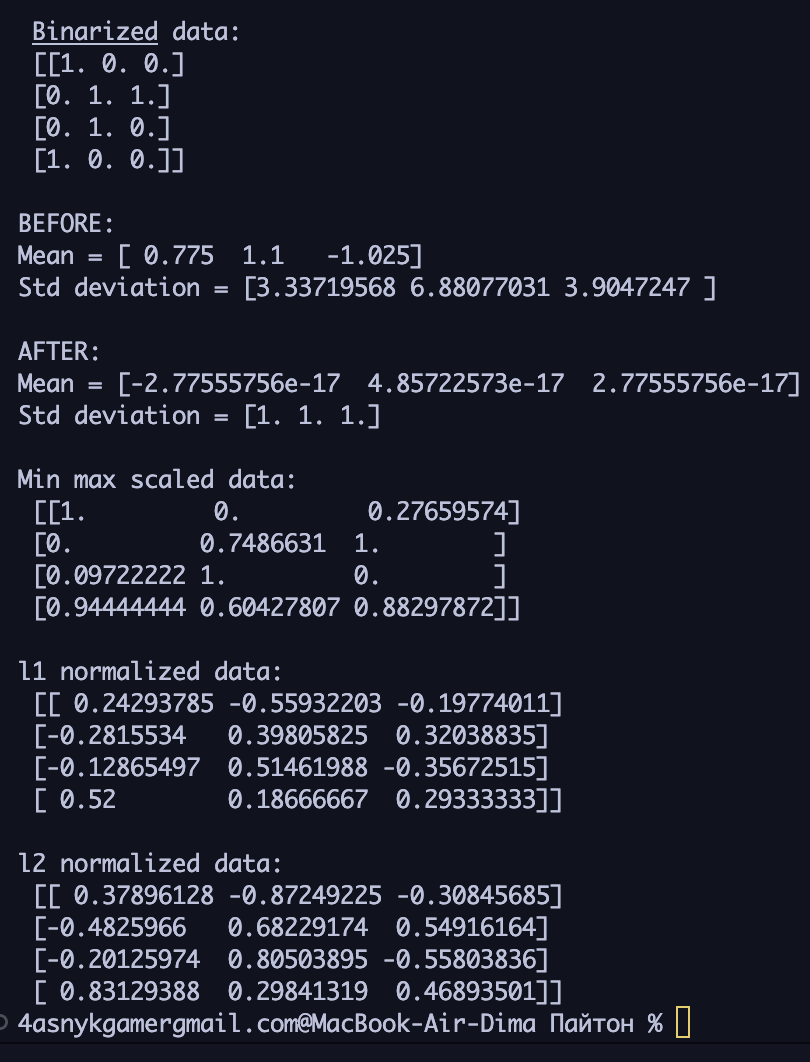


Рис.6. Результат виконання 2 завдання

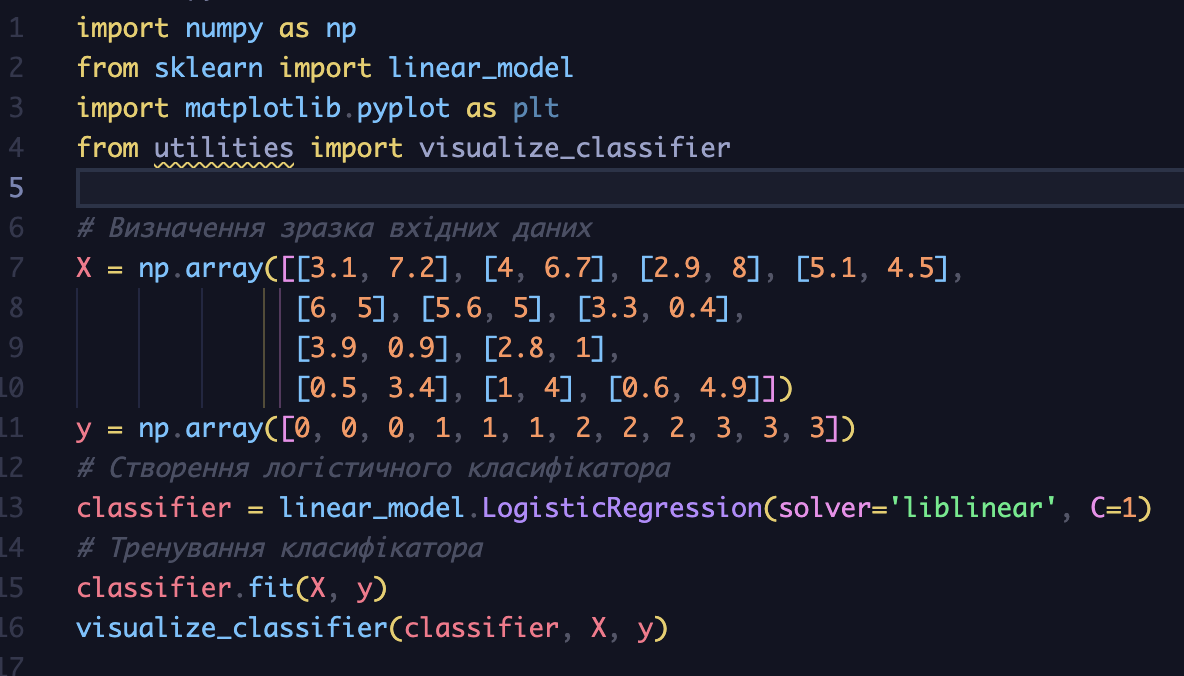


Рис.7. Скрін коду до 3-го завдання

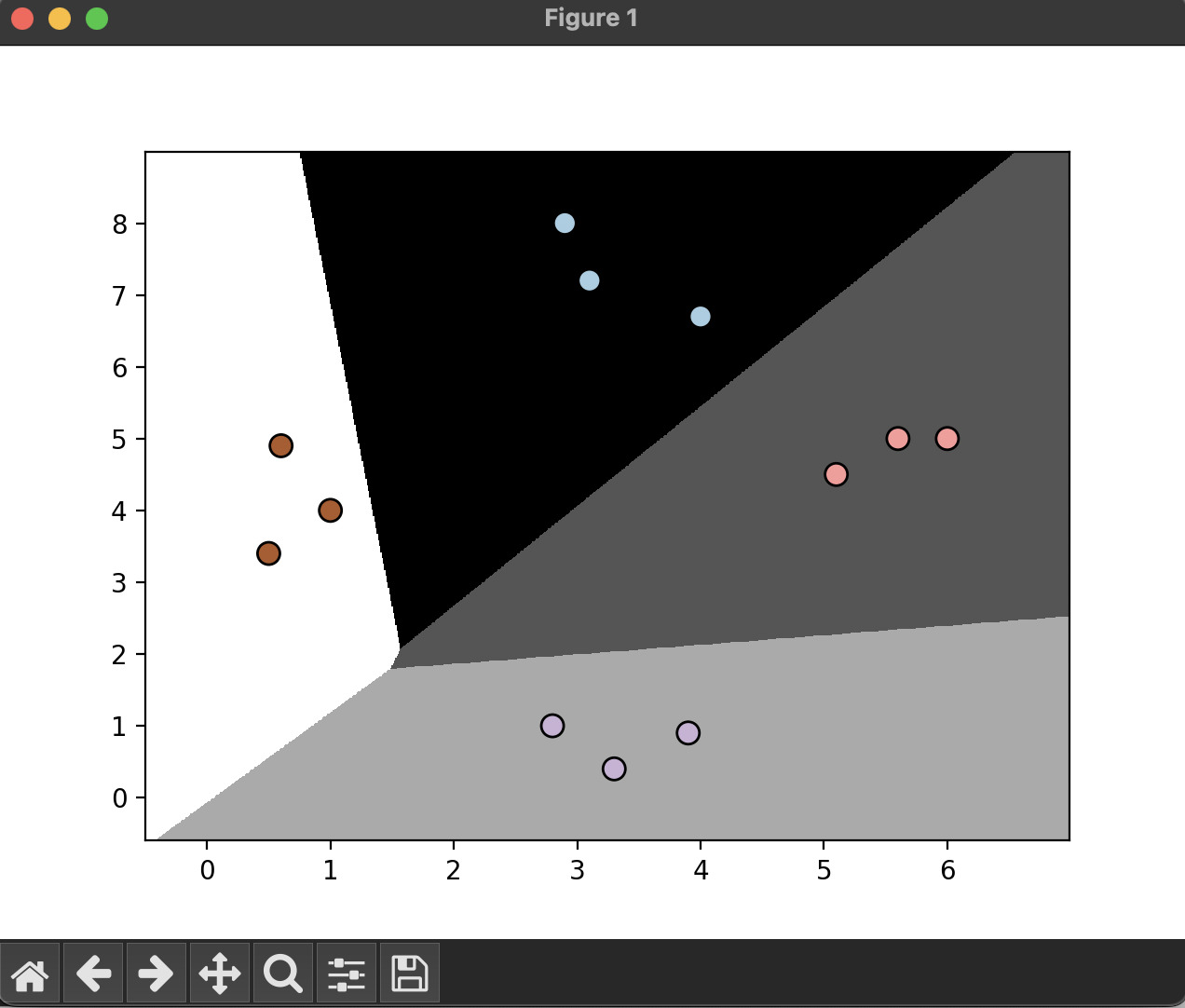


Рис.8. Результат виконання 3 завдання

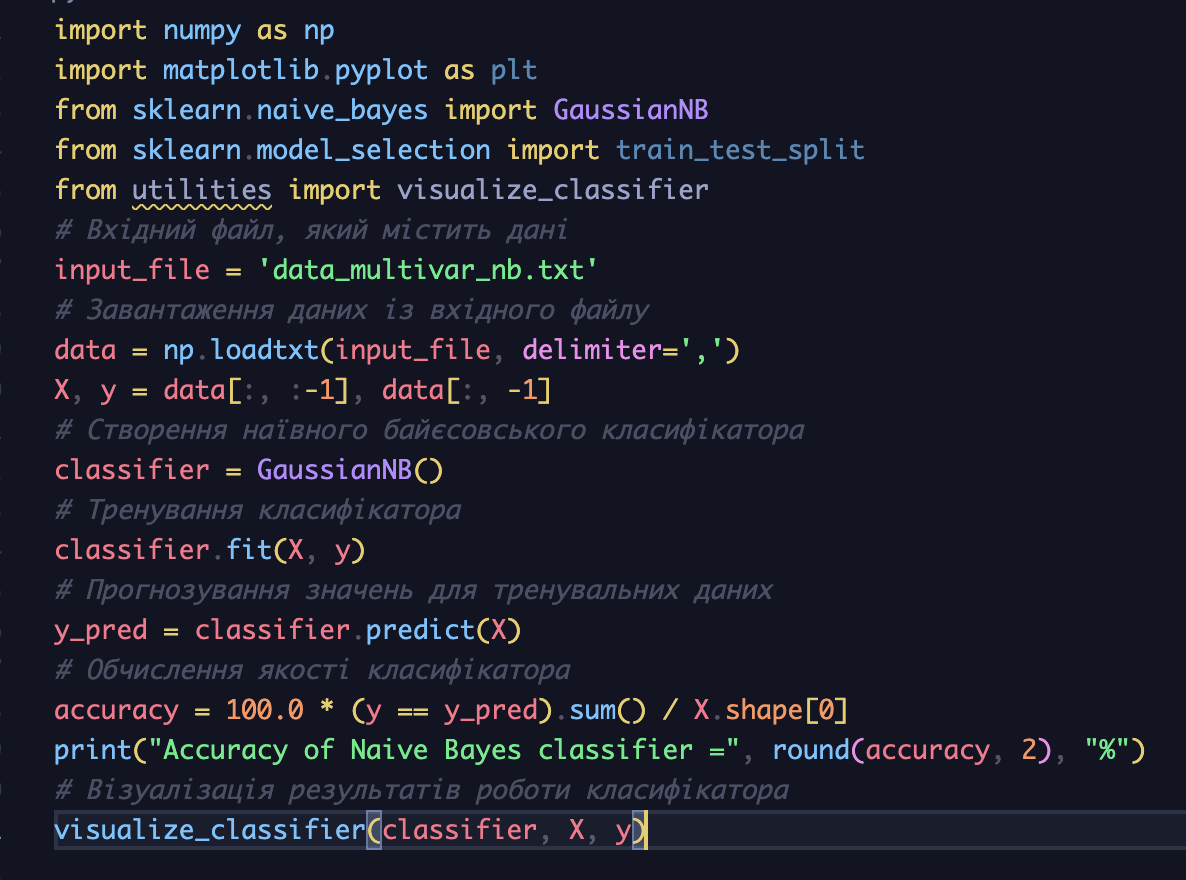


Рис.9. Скрін коду

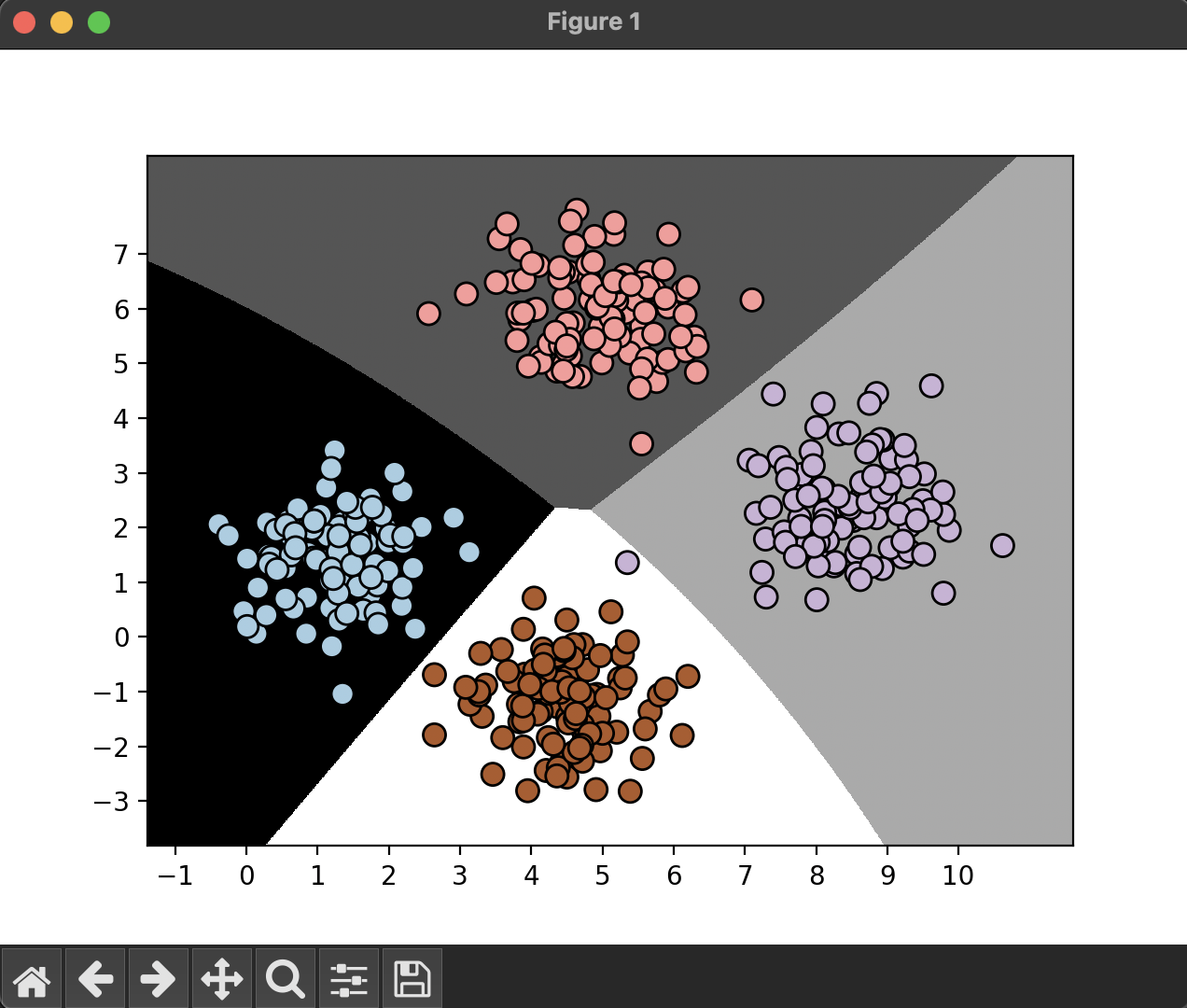




Рис.10. Результат виконання завдання



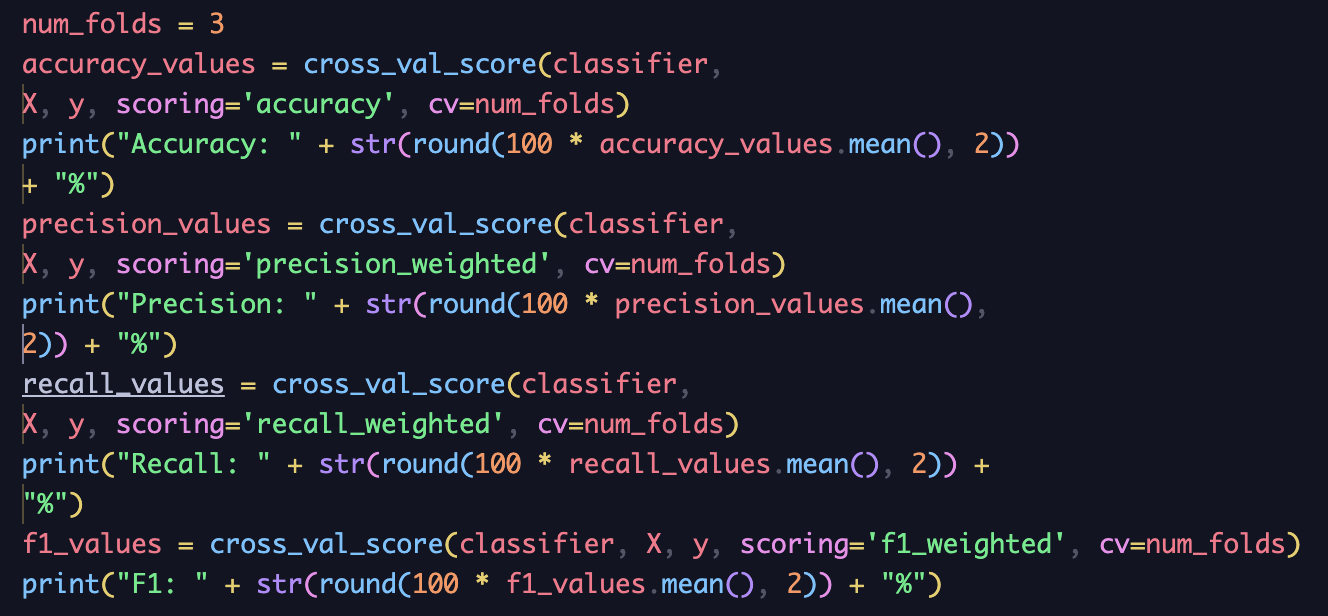


Рис.11. Скріни коду

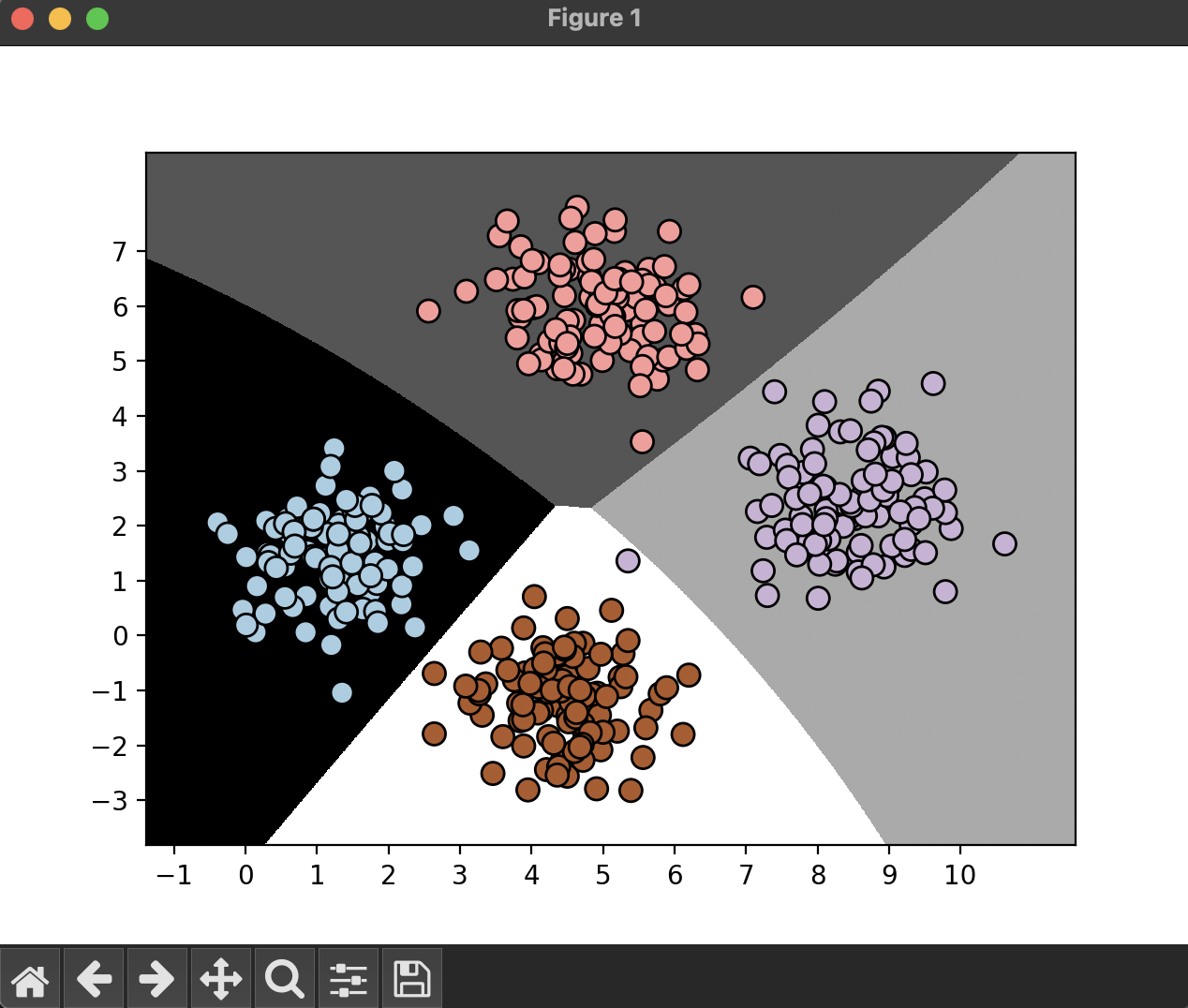


Рис.12. Результат першого прогону

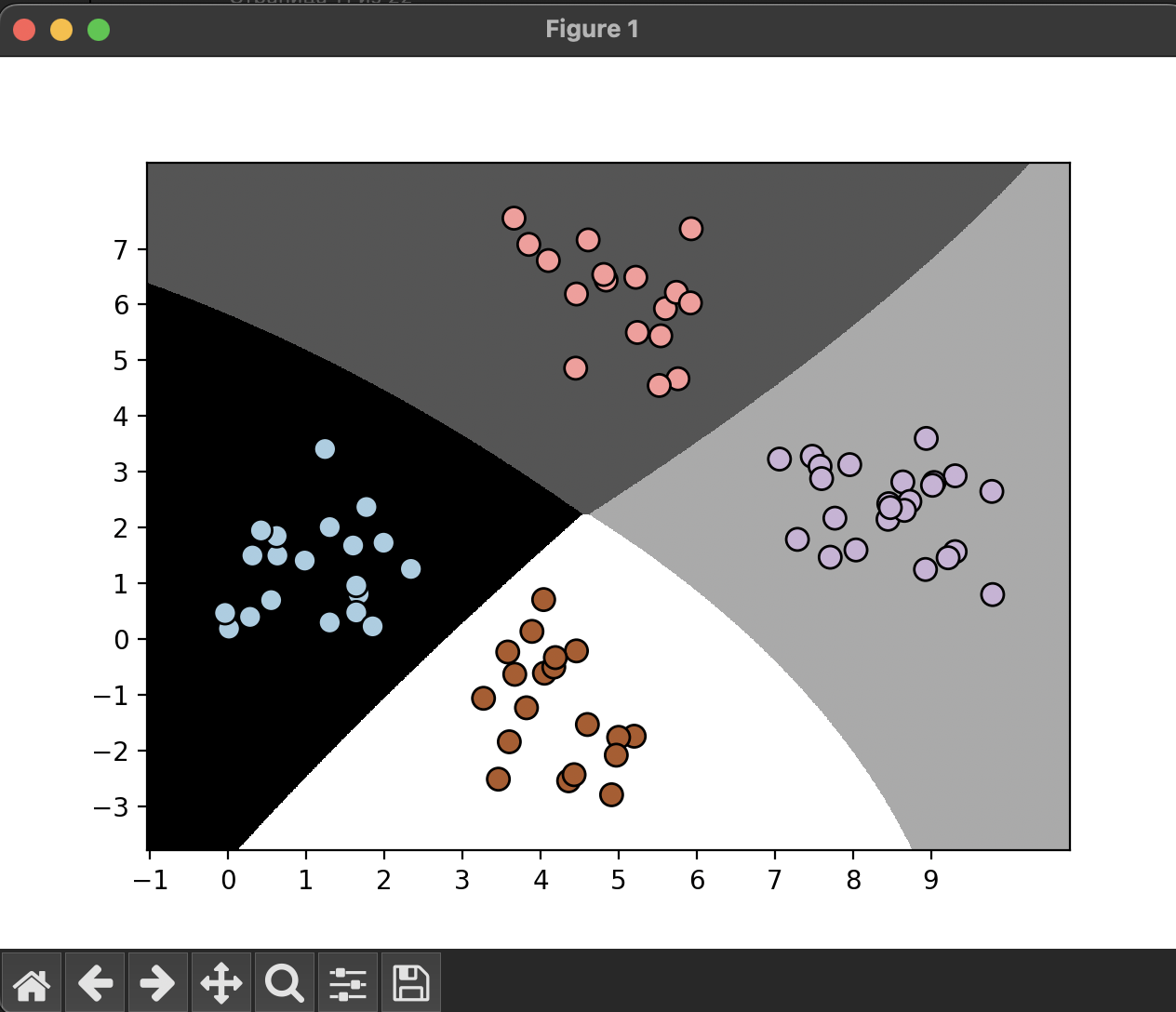


Рис.13. Результат другого прогону

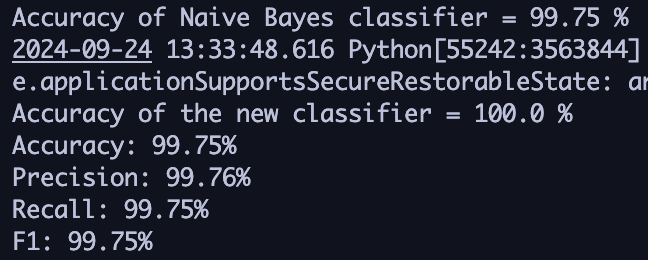


Рис.14. Результат виконання програми

**Точність початкового класифікатора (на всьому наборі даних)**:

**Accuracy**: 99.75%

Це свідчить про дуже високу точність наївного байєсівського класифікатора, коли перевірка проводиться на тому ж наборі даних, що й тренування.

**Точність нового класифікатора (на тестових даних)**:

**Accuracy**: 100.0%

Після розділення даних на тренувальні та тестові набори, новий класифікатор показав 100% точність на тестових даних, що може свідчити про відсутність помилок у прогнозуванні на цьому наборі даних.

**Перехресна перевірка**:

**Accuracy**: 99.75%

**Precision**: 99.76%

**Recall**: 99.75%

**F1**: 99.75%

Ці результати показують стабільно високу якість класифікації на всіх метриках, що підтверджує ефективність моделі.

#### **Висновки**

Наївний байєсівський класифікатор продемонстрував високу точність та стабільні результати. Оскільки точність на тестових даних досягла 100%, модель може бути надто пристосованою до цих даних, але загальні результати свідчать про хорошу загальну узгодженість моделі з навчальними даними.

Перехресна перевірка також підтвердила, що модель ефективно справляється з класифікацією, показуючи стабільні показники **Accuracy**, **Precision**, **Recall** та **F1**-міри.

***Порівняйте результати для різних порогів та зробіть висновки.***

***Порівняння***

**Accuracy**:

При порозі 0.5 точність моделі становить 0.671, що свідчить про те, що модель правильно класифікує приблизно 67.1% всіх зразків. Однак, при зменшенні порогу до 0.25 точність знижується до 0.502, що означає, що лише 50.2% зразків були правильно класифіковані.

**Recall**:

**Recall** при порозі 0.5 становить 0.641. Це означає, що 64.1% позитивних зразків були вірно передбачені. При зниженні порогу до 0.25 згадка зростає до 1.000, що вказує на те, що всі позитивні зразки (100%) були правильно класифіковані, але це призводить до великої кількості хибних сповіщень.

**Precision**:

При порозі 0.5 точність становить 0.681, що означає, що з усіх передбачених позитивних зразків 68.1% дійсно позитивні. При порозі 0.25 точність знижується до 0.501, що вказує на те, що лише 50.1% передбачених позитивних зразків є дійсно позитивними. Це свідчить про велику кількість хибних позитивів при нижчому порозі.

**F1-метрика**:

F1-метрика при порозі 0.5 становить 0.660, тоді як при порозі 0.25 зростає до 0.668. F1-метрика є гарним компромісом між точністю та згадкою, тому зростання значення F1-метрики при зниженні порогу свідчить про покращення збалансованості між згадкою та точністю, незважаючи на те, що точність знизилась.

### **Висновок:**

Зменшення порогу з 0.5 до 0.25 призводить до збільшення **Recall**, але за рахунок значного зниження точності. Модель стає "агресивнішою" у виявленні позитивних випадків, але також збільшує кількість хибних позитивів, що може бути неприйнятним у деяких контекстах. Вибір оптимального порогу залежить від конкретних вимог задачі: якщо важливіше виявити якомога більше позитивних випадків, може бути доцільно зменшити поріг. Якщо ж важливіше уникнути хибних сповіщень, то вищий поріг може бути кращим вибором.

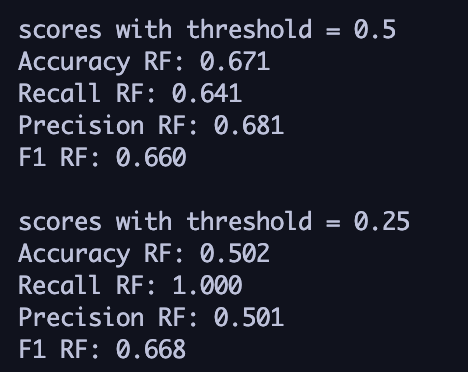
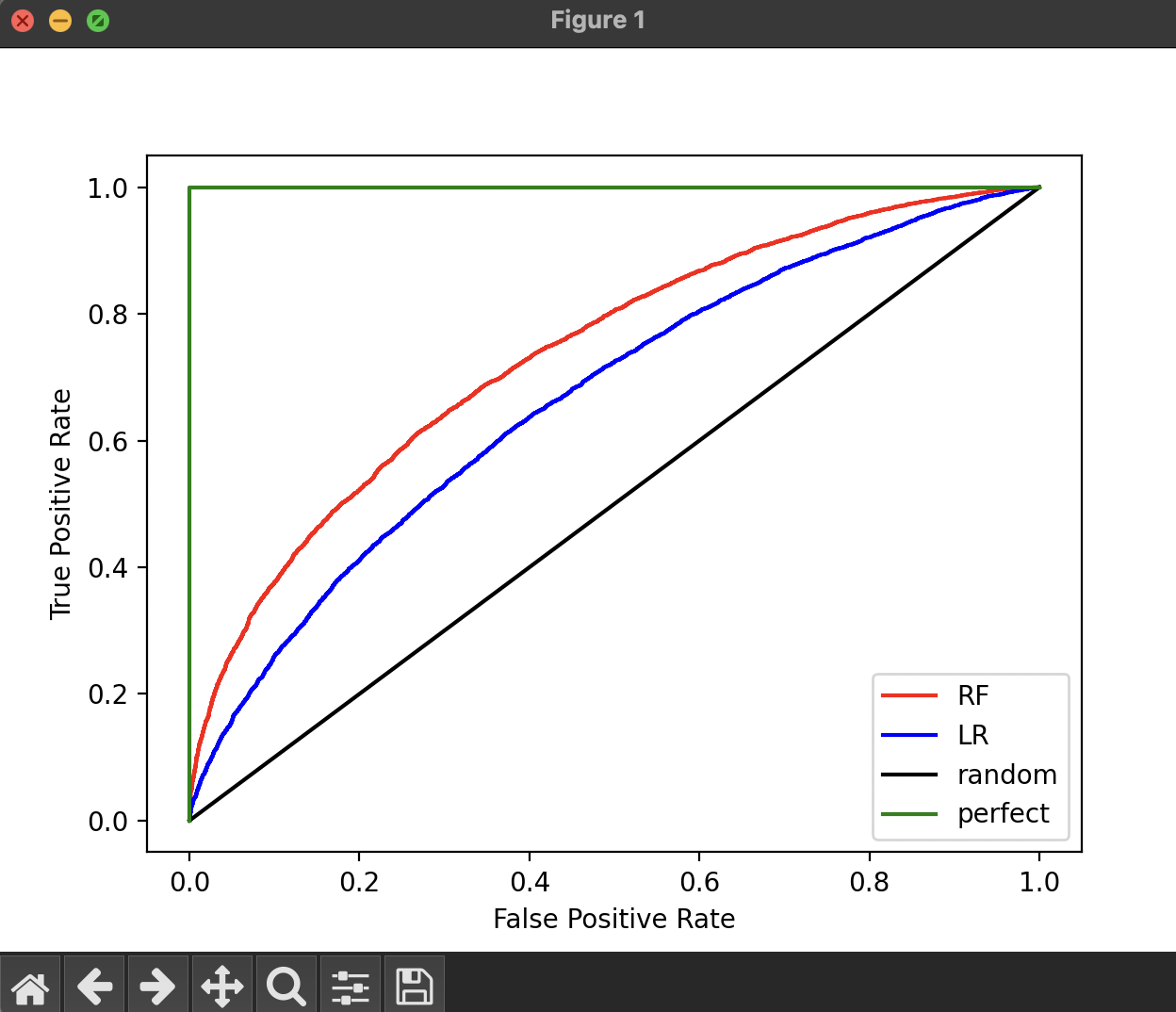


Рис.15. Результат виконання з різними порогами



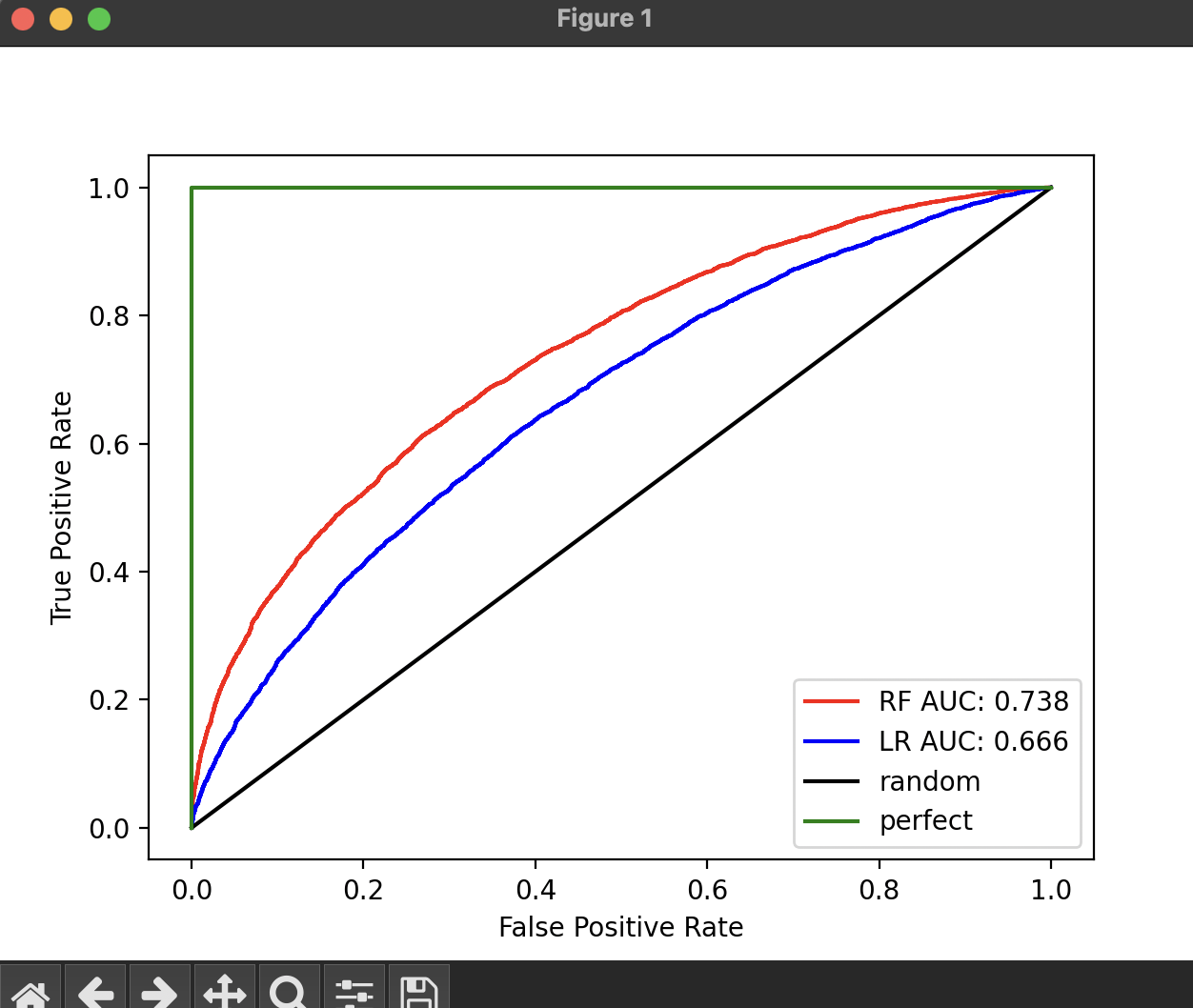


Рис.16. Крива Roc

### **Висновок: RF** показує кращі результати, ніж **LR**, оскільки модель **RF** має вищу точність і кращий баланс між точністю та згадкою, що підтверджується **F1**-метрикою. Крім того, **RF** краще працює в задачах з більш складною структурою даних, оскільки враховує нелінійні залежності, тоді як **LR** має обмеження через лінійність. Враховуючи ці фактори, можна зробити висновок, що **модель RF є кращим вибором** у даному випадку.

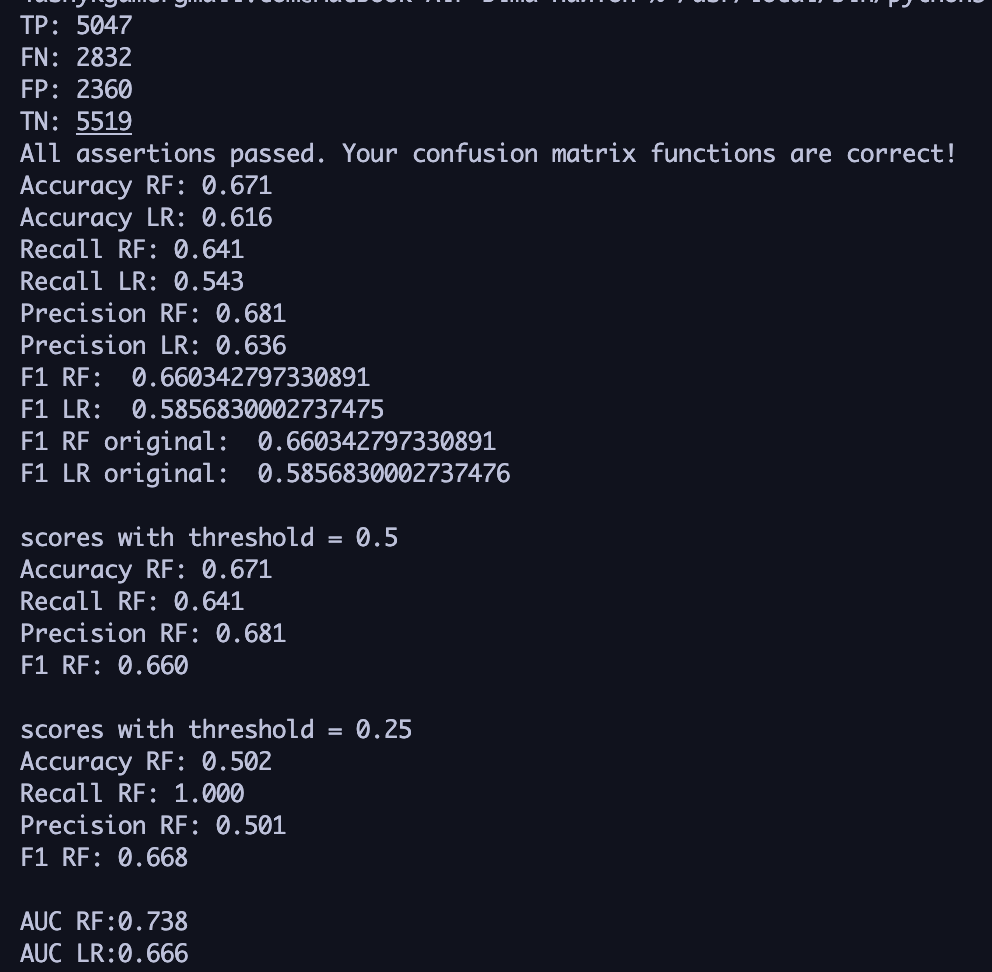


Рис.17. Результати виконання

Лістинг коду:

from sklearn.metrics import roc\_auc\_score

import matplotlib.pyplot as plt

from sklearn.metrics import roc\_curve

import numpy as np

import pandas as pd

from sklearn.metrics import confusion\_matrix, accuracy\_score, recall\_score, precision\_score, f1\_score

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

df.head()

thresh = 0.5

thresholds = [0.5, 0.25]

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

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

df.head()

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

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

*# counts the number of true positives (y\_true = 1, y\_pred = 1)*

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

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

*# counts the number of false negatives (y\_true = 1, y\_pred = 0)*

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

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

*# counts the number of false positives (y\_true = 0, y\_pred = 1)*

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

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

*# counts the number of true negatives (y\_true = 0, y\_pred = 0)*

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

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))

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 baginskiy\_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]])

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

*# Перевірка коректності матриці плутанини*

assert np.array\_equal(baginskiy\_confusion\_matrix(df.actual\_label.values, df.predicted\_RF.values),

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

'baginskiy\_confusion\_matrix() is not correct for RF'

assert np.array\_equal(baginskiy\_confusion\_matrix(df.actual\_label.values, df.predicted\_LR.values),

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

'baginskiy\_confusion\_matrix() is not correct for LR'

print("All assertions passed. Your confusion matrix functions are correct!")

*# Ваша власна функція для обчислення точності*

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

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

total = TP + FN + FP + TN

return (TP + TN) / total if total > 0 else 0.0 *# Обчислення точності*

*# Перевірка точності за допомогою assert*

assert baginskiy\_accuracy\_score(df.actual\_label.values, df.predicted\_RF.values) == accuracy\_score(

df.actual\_label.values, df.predicted\_RF.values), 'my\_accuracy\_score failed on RF'

assert baginskiy\_accuracy\_score(df.actual\_label.values, df.predicted\_LR.values) == accuracy\_score(

df.actual\_label.values, df.predicted\_LR.values), 'my\_accuracy\_score failed on LR'

*# Виведення результатів*

print('Accuracy RF: %.3f' % baginskiy\_accuracy\_score(

df.actual\_label.values, df.predicted\_RF.values))

print('Accuracy LR: %.3f' % baginskiy\_accuracy\_score(

df.actual\_label.values, df.predicted\_LR.values))

*# Ваша власна функція для обчислення відзиву (recall)*

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

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

total\_actual\_positives = TP + FN

*# Обчислення відзиву*

return TP / total\_actual\_positives if total\_actual\_positives > 0 else 0.0

*# Перевірка відзиву за допомогою assert*

assert baginskiy\_recall\_score(df.actual\_label.values, df.predicted\_RF.values) == recall\_score(

df.actual\_label.values, df.predicted\_RF.values), 'my\_recall\_score failed on RF'

assert baginskiy\_recall\_score(df.actual\_label.values, df.predicted\_LR.values) == recall\_score(

df.actual\_label.values, df.predicted\_LR.values), 'my\_recall\_score failed on LR'

*# Виведення результатів*

print('Recall RF: %.3f' % baginskiy\_recall\_score(

df.actual\_label.values, df.predicted\_RF.values))

print('Recall LR: %.3f' % baginskiy\_recall\_score(

df.actual\_label.values, df.predicted\_LR.values))

*# Ваша власна функція для обчислення точності (precision)*

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

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

total\_predicted\_positives = TP + FP

*# Обчислення точності*

return TP / total\_predicted\_positives if total\_predicted\_positives > 0 else 0.0

*# Перевірка точності за допомогою assert*

assert baginskiy\_precision\_score(df.actual\_label.values, df.predicted\_RF.values) == precision\_score(

df.actual\_label.values, df.predicted\_RF.values), 'my\_precision\_score failed on RF'

assert baginskiy\_precision\_score(df.actual\_label.values, df.predicted\_LR.values) == precision\_score(

df.actual\_label.values, df.predicted\_LR.values), 'my\_precision\_score failed on LR'

*# Виведення результатів*

print('Precision RF: %.3f' % baginskiy\_precision\_score(

df.actual\_label.values, df.predicted\_RF.values))

print('Precision LR: %.3f' % baginskiy\_precision\_score(

df.actual\_label.values, df.predicted\_LR.values))

*# Ваша власна функція для обчислення F1-метрики*

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

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

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

if precision + recall > 0:

*# Обчислення F1-метрики*

return 2 \* (precision \* recall) / (precision + recall)

else:

return 0.0 *# Повертає 0, якщо precision і recall обидва 0*

*# Перевірка F1-метрики за допомогою assert*

*# assert my\_f1\_score(df.actual\_label.values, df.predicted\_RF.values) == f1\_score(df.actual\_label.values, df.predicted\_RF.values), 'my\_f1\_score failed on RF'*

*# assert my\_f1\_score(df.actual\_label.values, df.predicted\_LR.values) == f1\_score(df.actual\_label.values, df.predicted\_LR.values), 'my\_f1\_score failed on LR'*

*# Виведення результатів*

print('F1 RF: ', baginskiy\_f1\_score(

df.actual\_label.values, df.predicted\_RF.values))

print('F1 LR: ', baginskiy\_f1\_score(

df.actual\_label.values, df.predicted\_LR.values))

print('F1 RF original: ', f1\_score(

df.actual\_label.values, df.predicted\_RF.values))

print('F1 LR original: ', f1\_score(

df.actual\_label.values, df.predicted\_LR.values))

print('')

for threshold in thresholds:

print(f'scores with threshold = {threshold}')

*# Генерація прогнозованих міток на основі поточного порогу*

predicted\_RF = (df.model\_RF >= threshold).astype('int')

*# Виведення показників*

print('Accuracy RF: %.3f' %

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

print('Recall RF: %.3f' %

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

print('Precision RF: %.3f' %

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

print('F1 RF: %.3f' % (baginskiy\_f1\_score(

df.actual\_label.values, predicted\_RF)))

print('')

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() """*

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)

import matplotlib.pyplot as plt

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')

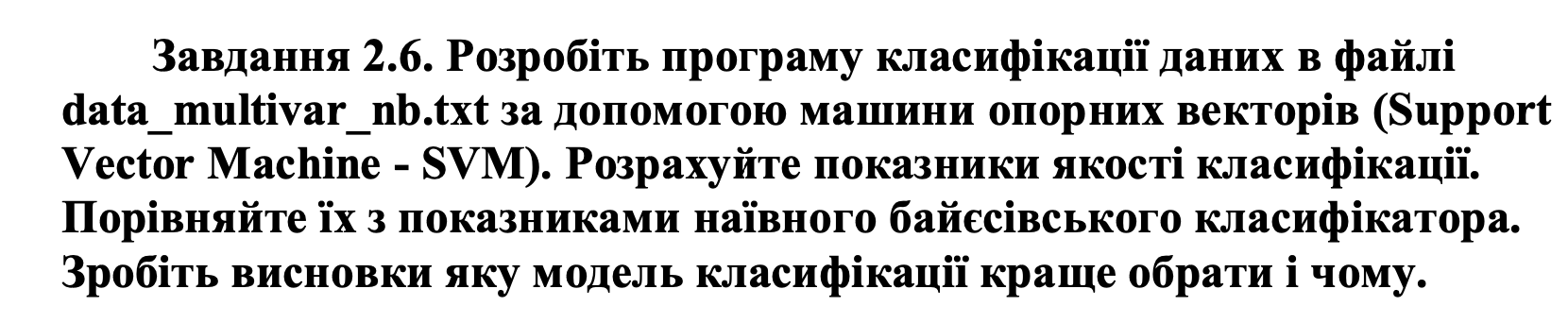
plt.plot([0,0,1,1],[0,1,1,1],'g-',label='perfect')

plt.legend()

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.show()



Лістинг:

import numpy as np

import pandas as pd

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

from sklearn.svm import SVC

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

*# Завантаження даних*

data = np.loadtxt('data\_multivar\_nb.txt', delimiter=',')

*# Розбиття на ознаки і мітки*

X = data[:, :-1] *# ознаки*

y = data[:, -1] *# мітки*

*# Розділення на навчальну та тестову вибірки*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

*# Створення та навчання моделі SVM*

svm\_model = SVC()

svm\_model.fit(X\_train, y\_train)

*# Прогнозування на тестових даних*

y\_pred\_svm = svm\_model.predict(X\_test)

*# Розрахунок показників для SVM*

accuracy\_svm = accuracy\_score(y\_test, y\_pred\_svm)

precision\_svm = precision\_score(y\_test, y\_pred\_svm, average='macro')

recall\_svm = recall\_score(y\_test, y\_pred\_svm, average='macro')

f1\_svm = f1\_score(y\_test, y\_pred\_svm, average='macro')

print(f"SVM - Accuracy: {accuracy\_svm:.3f}, Precision: {precision\_svm:.3f}, Recall: {recall\_svm:.3f}, F1 Score: {f1\_svm:.3f}")

*# Створення та навчання моделі наївного байєсівського класифікатора*

nb\_model = GaussianNB()

nb\_model.fit(X\_train, y\_train)

*# Прогнозування на тестових даних*

y\_pred\_nb = nb\_model.predict(X\_test)

*# Розрахунок показників для наївного Байєса*

accuracy\_nb = accuracy\_score(y\_test, y\_pred\_nb)

precision\_nb = precision\_score(y\_test, y\_pred\_nb, average='macro')

recall\_nb = recall\_score(y\_test, y\_pred\_nb, average='macro')

f1\_nb = f1\_score(y\_test, y\_pred\_nb, average='macro')

print(f"Naive Bayes - Accuracy: {accuracy\_nb:.3f}, Precision: {precision\_nb:.3f}, Recall: {recall\_nb:.3f}, F1 Score: {f1\_nb:.3f}")

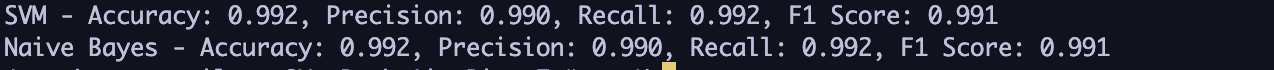


Рис.18. Результат виконання

### **Порівняння результатів:**

Обидві моделі — машина опорних векторів (SVM) та наївний байєсівський класифікатор — показали ідентичні результати за всіма основними метриками:

* **Точність (Accuracy)**: 0.992
* **Точність (Precision)**: 0.990
* **Повнота (Recall)**: 0.992
* **F1-оцінка (F1 Score)**: 0.991

Це свідчить про те, що обидві моделі працюють майже ідеально для даного набору даних, класифікуючи приклади з мінімальною кількістю помилок.

### **Висновки:** так як дві моделі показали однаково високі результати, вибір між ними буде залежати від ваших потреб. **SVM -** це потужна модель, для складних даних, особливо коли класифікаційні межі нелінійні, але як мінус вона більш ресурсомістка, особливо на великих наборах даних, і час навчання може бути довшим. **Наївний Байєс -** це більш простіша та швидкіша модель, яка краще працює на текстових даних або коли ознаки незалежні. Його основна перевага — швидкість і ефективність на невеликих наборах даних.