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

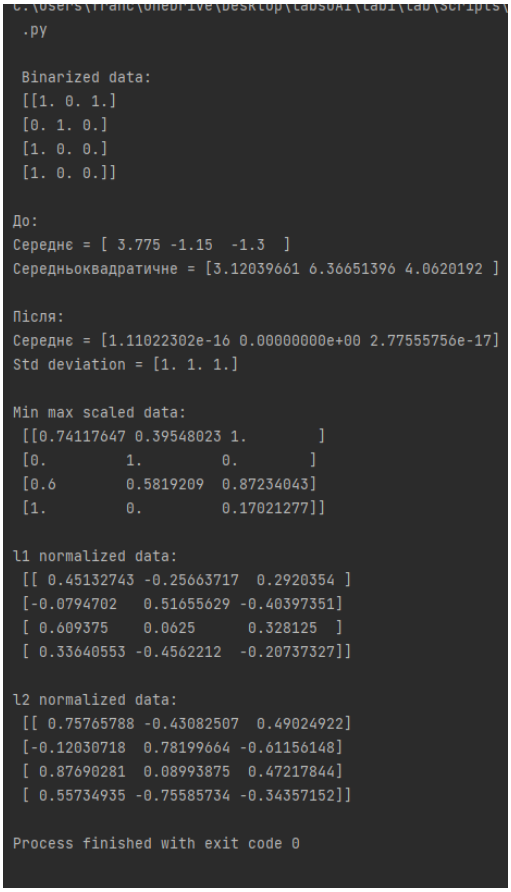
Завдання 2.1: Попередня обробка даних.

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("\nMin 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)
```

					ДУ «Житомирська політехніка».23.121.8.000 – Лр1		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Іваницький Ф.А.			Звіт з лабораторної роботи	Літ.	Арк. 1 / Аркушів
Перевір.		Голенко М. Ю.					
Керівник						ФІКТ Гр. ІПЗ-20-3	
Н. контр.							
Зав. каф.							

Результат роботи програми:



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 $||x||_1 = |x_1| + |x_2| + ... + |x_n|$. 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 $||x||_2 = \sqrt{(x_1^2 + x_2^2 + ... + x_n^2)}$. 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:

```
C:\Users\franc\OneDrive\Desktop\labsOAI\lab1\lab\Scripts\python.exe "C:\Pr

import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend(['C:\\Users\\franc\\OneDrive\\Desktop\\labsOAI\\lab1', 'C:

Python 3.10.1 (tags/v3.10.1:2cd268a, Dec  6 2021, 19:10:37) [MSC v.1929 64

Label mapping:
Back --> 0
black --> 1
green --> 2
red --> 3
white --> 4
yellow --> 5

Labels = ['green', 'red', 'Back']
Encoded values = [2, 3, 0]
```

Завдання 2.2: Попередня обробка нових даних
Варіант 3:

№ варіанту	Значення змінної input_data												Поріг бінаризації
8.	4.6	9.9	-3.5	-2.9	4.1	3.3	-2.2	8.8	-6.1	3.9	1.4	2.2	2.2

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 MinMax
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:

```
mport sys; print('Python %s on %s' % (sys.version, sys.platform))
ys.path.extend(['C:\\Users\\franc\\OneDrive\\Desktop\\labsOAI\\lab1', 'C:/Use

ython 3.10.1 (tags/v3.10.1:2cd268a, Dec  6 2021, 19:10:37) [MSC v.1929 64 bit

inарізація даних:
[[1. 1. 0.]
[0. 1. 1.]
[0. 1. 0.]
[1. 0. 0.]]

EFORE:
ean = [ 0.85   6.05  -1.025]
td deviation = [3.41796723 3.45723878 3.9047247 ]

иключення середнього :
ean = [0.00000000e+00 1.11022302e-16 2.77555756e-17]
td deviation = [1. 1. 1.]

інімальне Максимальне Масштабованих даних:
[[1.         1.         0.27659574]
[0.         0.31764706 1.        ]
[0.09333333 0.87058824 0.        ]
[0.90666667 0.         0.88297872]]

1 normalized data:
[[ 0.25555556  0.55       -0.19444444]
[-0.2815534   0.39805825  0.32038835]
[-0.12865497  0.51461988 -0.35672515]
[ 0.52        0.18666667  0.29333333]]

2 normalized data:
[[ 0.40126114  0.86358375 -0.30530739]
[-0.4825966   0.68229174  0.54916164]
[-0.20125974  0.80503895 -0.55803836]
[ 0.83129388  0.29841319  0.46893501]]
```

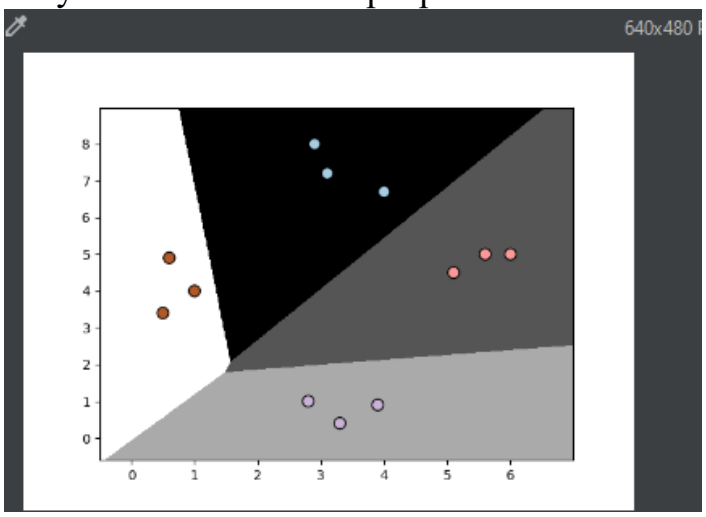
Завдання 2.3: Класифікація логістичною регресією або логістичний класифікатор

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:



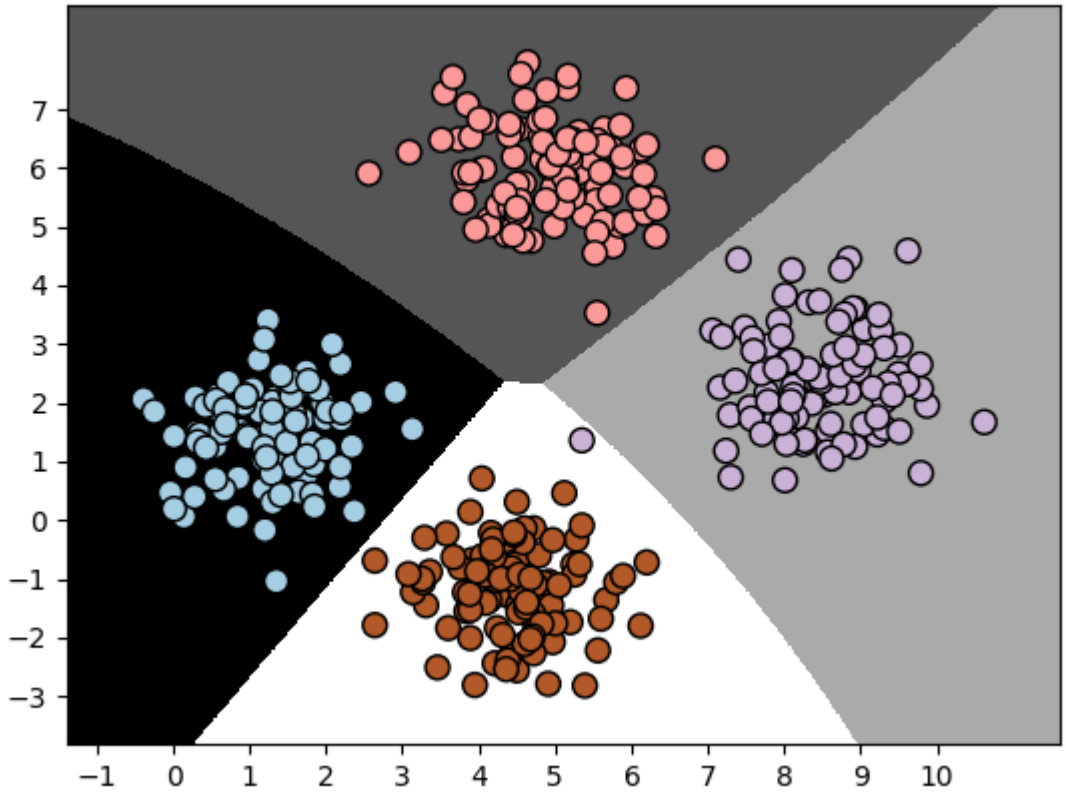
Завдання 2.4: Класифікація наївним байєсовським класифікатором.

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:

```
ys.path.extend(['C:\\Users\\franc\\OneDrive\\Desktop\\labsOAI\\lab1', 'C:/Users/franc/OneDrive/Desktop/labsOAI/lab1'])

ython 3.10.1 (tags/v3.10.1:2cd268a, Dec  6 2021, 19:10:37) [MSC v.1929 64 bit (AMD64)]
ccuracy of Naive Bayes classifier = 99.75 %
```



Лістинг програми після розбиття даних на навчальний та тестовий набори:

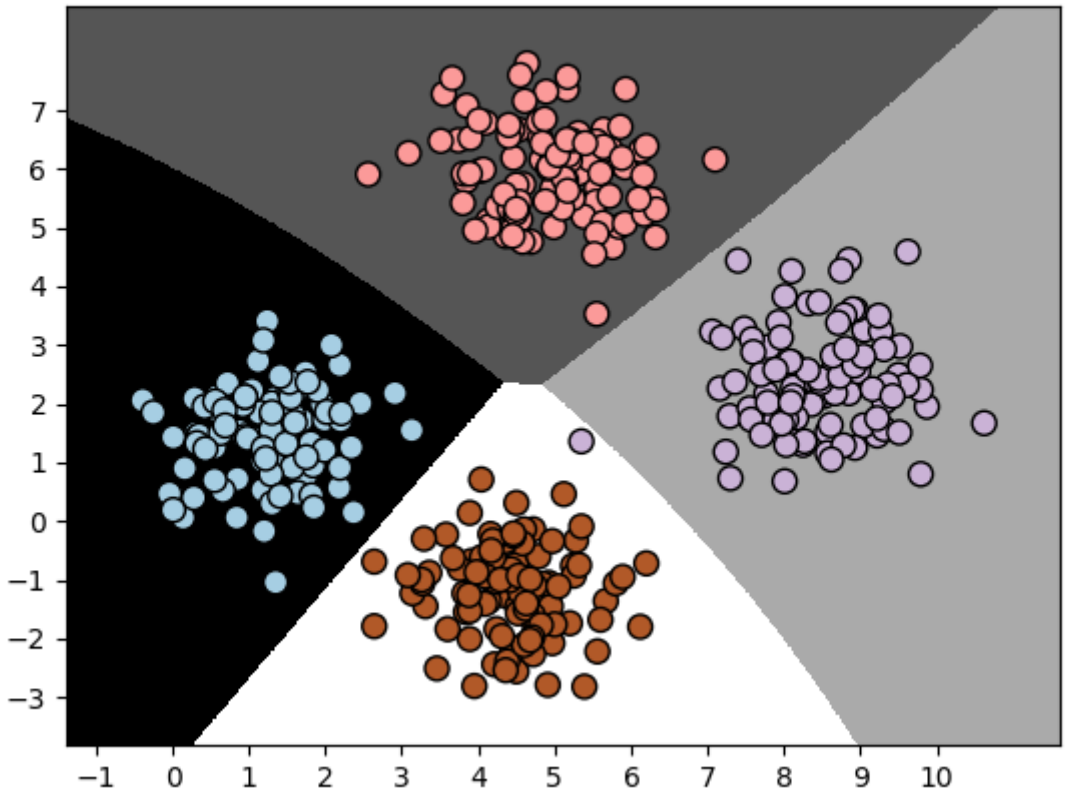
```
import pandas as pd
import numpy as np
df = pd.read_csv('data_metrics.csv')
df.head()
thresh = 0.5
df['predicted_RF'] = (df.model_RF >= 0.5).astype('int')
df['predicted_LR'] = (df.model_LR >= 0.5).astype('int')
df.head()
from sklearn.metrics import confusion_matrix, recall_score
confusion_matrix(df.actual_label.values, 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)
```

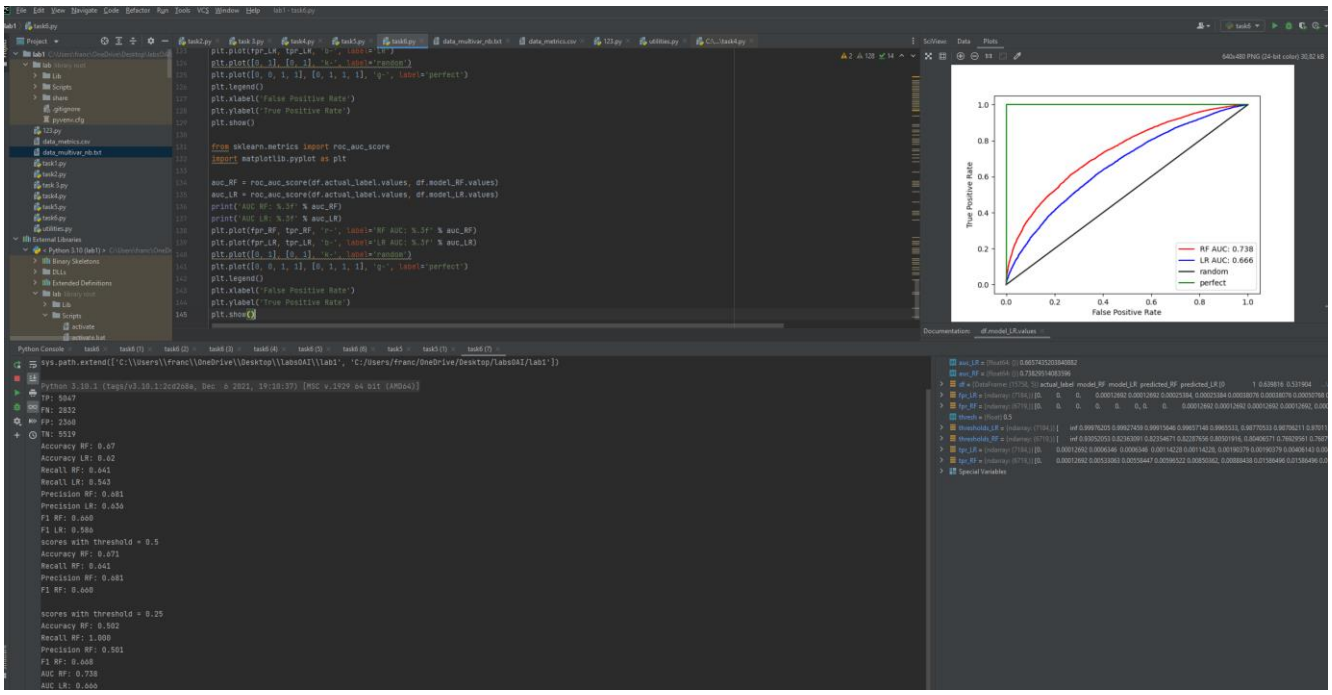
```
def ivanytskyi_precision_score(y_true, y_pred):
    TP, FN, FP, TN = find_conf_matrix_values(y_true, y_pred)
    precision = TP / (TP + FP)
    return precision


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:

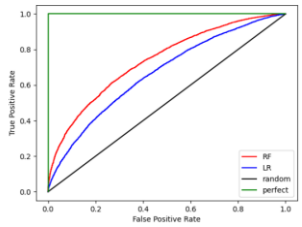


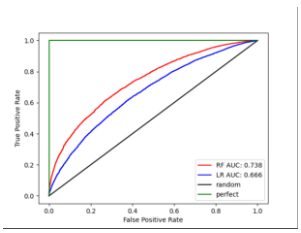
Завдання №2.7

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 кросвалідацію, 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.