

Міністерство освіти і науки України  
Західноукраїнський національний університет

ЗВІТ  
З МОДУЛЬНОЇ РОБОТИ №1  
із дисципліни «Обчислювальний інтелект»  
на тему «Дослідження побудови класифікатора та регресора методом k-  
найближчих сусідів (k-nn)»

Виконав:  
Студент: Рябий В.В.  
Групи: КНм-11

Тернопіль 2024

## Зміст

Вступ.....	3
Хід роботи .....	4
Завдання 1. Створення KNN – класифікатора у Python .....	4
Завдання 2. Створення KNN – регресора у Python .....	10
Висновок .....	15

## Вступ

Дослідження побудови класифікатора та регресора методом k-найближчих сусідів (k-nn) є важливим етапом у вивченні обчислювального інтелекту. Метод k-nn є одним із найпростіших та найефективніших алгоритмів машинного навчання для класифікації та регресії, він заснований на концепції використання схожості між прикладами даних.

Метою даної роботи є вивчення можливостей аналізу даних з використанням класифікатора та регресора методом k-найближчих сусідів (k-nn). У ході роботи буде створено k-nn класифікатор та регресор у середовищі програмування Python.

Під час виконання ми створимо класифікатор KNN на мові програмування Python, завантажимо базу параметрів, виконаємо операції над даними і виберемо величину K для найкращих показників якості класифікації у тестовій вибірці. Також розробимо KNN – регресора у Python і виберемо величину K для найкращих показників якості регресії у тестовій вибірці, здійснимо візуалізацію отриманих рішень.

## Хід роботи

Мета роботи: Вивчити можливості аналізу даних з використанням класифкатора та регресора методом k-найближчих сусідів (k-nn).

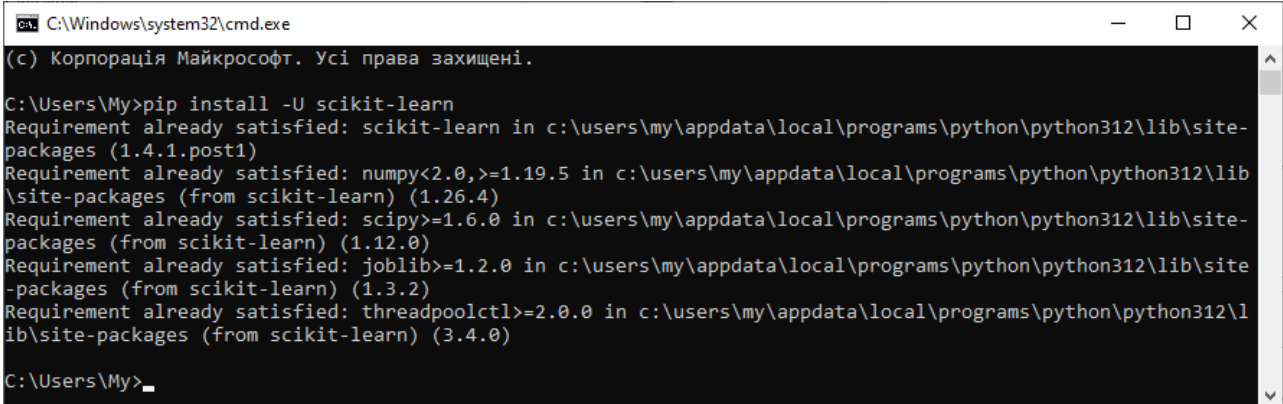
Завдання 1. Створення KNN – класифікатора у Python

Розробка програмної реалізації Python, яка забезпечує виконання наступних кроків:

Крок 1. Завантажити базу параметрів квітів iris dataset

Iris dataset є одним з вбудованих наборів даних у бібліотеці scikit-learn у Python. Ви можете встановити цю бібліотеку, використовуючи рір:

```
pip install -U scikit-learn
```



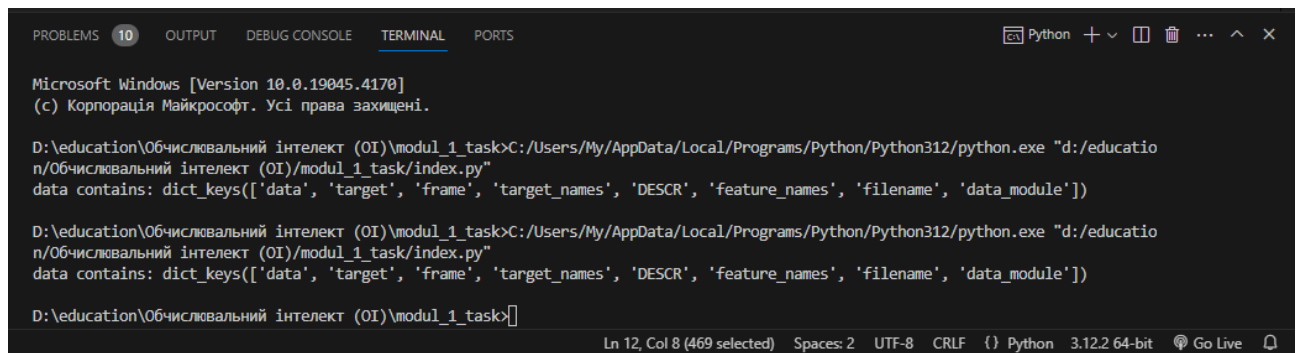
```
C:\Users\My>pip install -U scikit-learn
Requirement already satisfied: scikit-learn in c:\users\my\appdata\local\programs\python\python312\lib\site-packages (1.4.1.post1)
Requirement already satisfied: numpy<2.0,>=1.19.5 in c:\users\my\appdata\local\programs\python\python312\lib\site-packages (from scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in c:\users\my\appdata\local\programs\python\python312\lib\site-packages (from scikit-learn) (1.12.0)
Requirement already satisfied: joblib>=1.2.0 in c:\users\my\appdata\local\programs\python\python312\lib\site-packages (from scikit-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\my\appdata\local\programs\python\python312\lib\site-packages (from scikit-learn) (3.4.0)
C:\Users\My>
```

Рисунок 1.1 Завантаження Iris dataset

Для перевірки чи успішно ми встановили бібліотеку можна скористатися наступним кодом, супутно перевіряючи, які ключі містить Iris.

```
import pandas as pd
import numpy as np
np.random.seed = 2021
from sklearn.datasets import load_iris
iris = load_iris()
print ('data contains:',iris.keys())
X, y, labels, feature_names = iris.data, iris.target, iris.target_names,
iris['feature_names']
df_iris= pd.DataFrame(X, columns= feature_names)
df_iris['label'] = y
features_dict = {k:v for k,v in enumerate(labels)}
df_iris['label_names'] = df_iris.label.apply(lambda x: features_dict[x])
df_iris
```

Результат виконання коду і вивід даних у терміналі зображено на рисунку 1.2.



```
Microsoft Windows [Version 10.0.19045.4170]
(c) Корпорація Майкрософт. Усі права захищені.

D:\education\Обчислювальний інтелект (OI)\modul_1_task>C:/Users/My/AppData/Local/Programs/Python/Python312/python.exe "d:/education/Обчислювальний інтелект (OI)\modul_1_task/index.py"
data contains: dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])

D:\education\Обчислювальний інтелект (OI)\modul_1_task>C:/Users/My/AppData/Local/Programs/Python/Python312/python.exe "d:/education/Обчислювальний інтелект (OI)\modul_1_task/index.py"
data contains: dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])

D:\education\Обчислювальний інтелект (OI)\modul_1_task>
```

Рисунок 1.2 Перевірка ключів Iris

## Крок 2. Перемішати записи у завантаженій базі

Для перемішування записів у базі Iris dataset використовуємо функцію `shuffle` з бібліотеки `numpy`. Ось як це можна зробити:

```
import numpy as np
from sklearn.datasets import load_iris

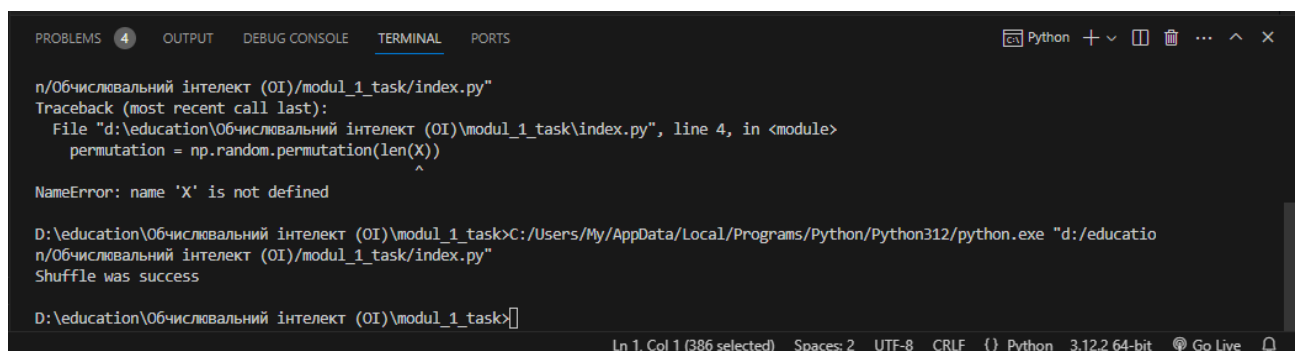
# Завантажуємо Iris dataset
iris = load_iris()

# Отримуємо дані та мітки класів
X = iris.data # Ознаки
y = iris.target # Мітки класів

# Перемішуємо дані та мітки класів одночасно
permutation = np.random.permutation(len(X))
X_shuffled = X[permutation]
y_shuffled = y[permutation]

print('Shuffle was success')
```

У цьому коді `permutation` - це випадкова перестановка індексів довжини даних. Потім ми застосовуємо цю перестановку до ознак (`X`) та міток класів (`y`), щоб перемішати їх у відповідності з цією перестановкою. Тепер `X_shuffled` та `y_shuffled` містять перемішані дані та мітки класів відповідно.



```
n/Обчислювальний інтелект (OI)\modul_1_task/index.py"
Traceback (most recent call last):
  File "d:\education\Обчислювальний інтелект (OI)\modul_1_task\index.py", line 4, in <module>
    permutation = np.random.permutation(len(X))
                                         ^
NameError: name 'X' is not defined

D:\education\Обчислювальний інтелект (OI)\modul_1_task>C:/Users/My/AppData/Local/Programs/Python/Python312/python.exe "d:/education/Обчислювальний інтелект (OI)\modul_1_task/index.py"
Shuffle was success

D:\education\Обчислювальний інтелект (OI)\modul_1_task>
```

Рисунок 1.3 Перемішані записи у завантаженій базі

### Крок 3. Нормалізувати параметри квітів ірису

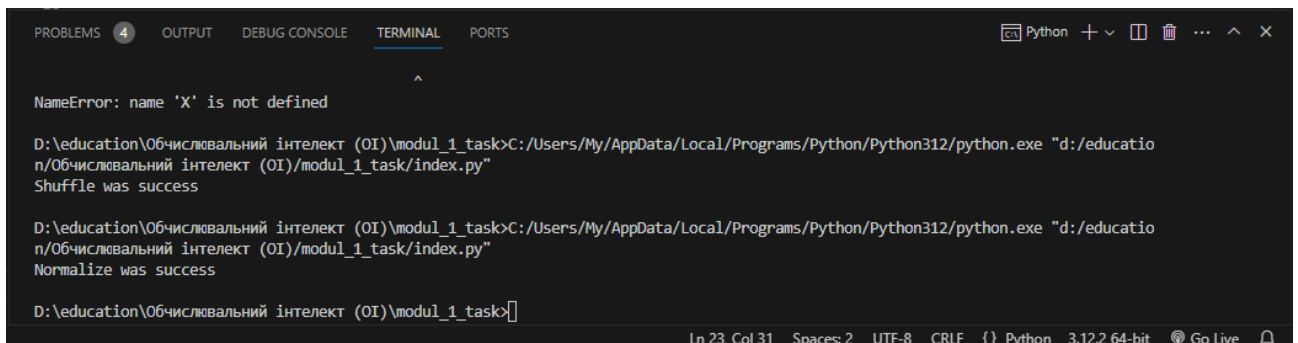
Для нормалізації параметрів квітів ірису можна скористатися стандартною нормалізацією, яка полягає в відніманні середнього значення і поділі на стандартне відхилення кожного параметра. Ось як це можна зробити за допомогою бібліотеки scikit-learn:

```
from sklearn.preprocessing import StandardScaler

# Ініціалізуємо нормалізатор
scaler = StandardScaler()

# Проводимо нормалізацію для ознак
X_normalized = scaler.fit_transform(X_shuffled)
```

Допишемо вже існуючий код додавши до нього вище описані рядки. `X_shuffled` - це перемішані ознаки, що були отримані на попередньому кроці. `StandardScaler` нормалізує кожен параметр таким чином, щоб його середнє значення стало рівним 0, а стандартне відхилення - 1. `fit_transform` метод використовується для підрахунку середнього та стандартного відхилення параметрів і одночасної нормалізації даних.



```
PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python + - [ ] [ ] ... ^ x

NameError: name 'X' is not defined
^
D:\education\Обчислювальний інтелект (OI)\modul_1_task>C:/Users/My/AppData/Local/Programs/Python/Python312/python.exe "d:/education/Обчислювальний інтелект (OI)/modul_1_task/index.py"
Shuffle was success
D:\education\Обчислювальний інтелект (OI)\modul_1_task>C:/Users/My/AppData/Local/Programs/Python/Python312/python.exe "d:/education/Обчислювальний інтелект (OI)/modul_1_task/index.py"
Normalize was success
D:\education\Обчислювальний інтелект (OI)\modul_1_task>[ ]

Ln 23, Col 31 Spaces: 2 UTF-8 CRLF { } Python 3.12.2 64-bit Go Live
```

Рисунок 1.4 Результат нормалізації по X

### Крок 4. Розділити існуючі записи на навчальну і тестові вибірки

Об'єднуємо вище описані фрагменти коду і отримуємо програму, яка реалізує перемішування записів, нормалізацію параметрів та розділення на навчальну та тестову вибірки:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np

# Завантаження даних Iris
iris = load_iris()
```

```

X = iris.data
y = iris.target

# Перемішування записів
indices = np.random.permutation(len(X))
X_shuffled = X[indices]
y_shuffled = y[indices]

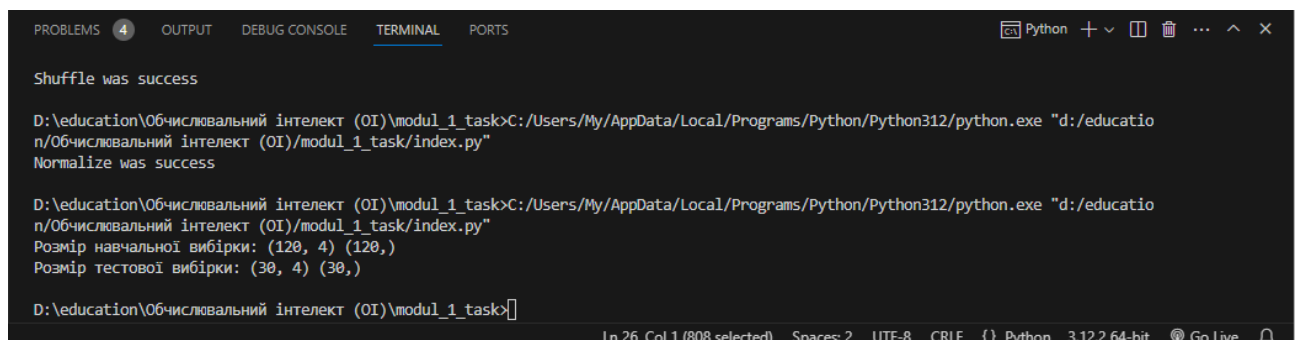
# Нормалізація параметрів
scaler = StandardScaler()
X_normalized = scaler.fit_transform(X_shuffled)

# Розділення на навчальну та тестову вибірки
X_train, X_test, y_train, y_test = train_test_split(X_normalized, y_shuffled,
test_size=0.2, random_state=42)

# Вивід розмірів навчальної та тестової вибірок
print("Розмір навчальної вибірки:", X_train.shape, y_train.shape)
print("Розмір тестової вибірки:", X_test.shape, y_test.shape)

```

У цьому коді: Ми завантажуюмо набір даних Iris з `sklearn.datasets`. Перемішуємо записи та відповідні мітки класів. Нормалізуємо параметри квітів ірису з використанням `StandardScaler`. Розділяємо дані на навчальну та тестову вибірки, використовуючи `train_test_split`. Після виконання цього коду ви отримаєте навчальні та тестові вибірки `X_train`, `X_test`, `y_train`, `y_test`.



```

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python + v [Icons] ... ^ X

Shuffle was success

D:\education\Обчислювальний інтелект (OI)\modul_1_task>C:/Users/My/AppData/Local/Programs/Python/Python312/python.exe "d:/education/Обчислювальний інтелект (OI)/modul_1_task/index.py"
Normalize was success

D:\education\Обчислювальний інтелект (OI)\modul_1_task>C:/Users/My/AppData/Local/Programs/Python/Python312/python.exe "d:/education/Обчислювальний інтелект (OI)/modul_1_task/index.py"
Розмір навчальної вибірки: (120, 4) (120,)
Розмір тестової вибірки: (30, 4) (30,)

D:\education\Обчислювальний інтелект (OI)\modul_1_task>
Ln 26, Col 1 (808 selected) Spaces: 2 UTF-8 CRLF {} Python 3.12.2 64-bit Go Live

```

Рисунок 1.5 Створення на навчальної і тестової вибірки

## Крок 5. Навчити KNN-класифікатор з різними значеннями K

Щоб, навчити KNN-класифікатор з різними значеннями K потрібно доповнити існуючий код наступними рядками:

```

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Створення списку різних значень K
k_values = [1, 3, 5, 7, 9]

```

```
# Цикл для навчання та оцінки класифікатора для кожного значення K
for k in k_values:
    # Створення KNN-класифікатора з поточним значенням K
    knn_classifier = KNeighborsClassifier(n_neighbors=k)

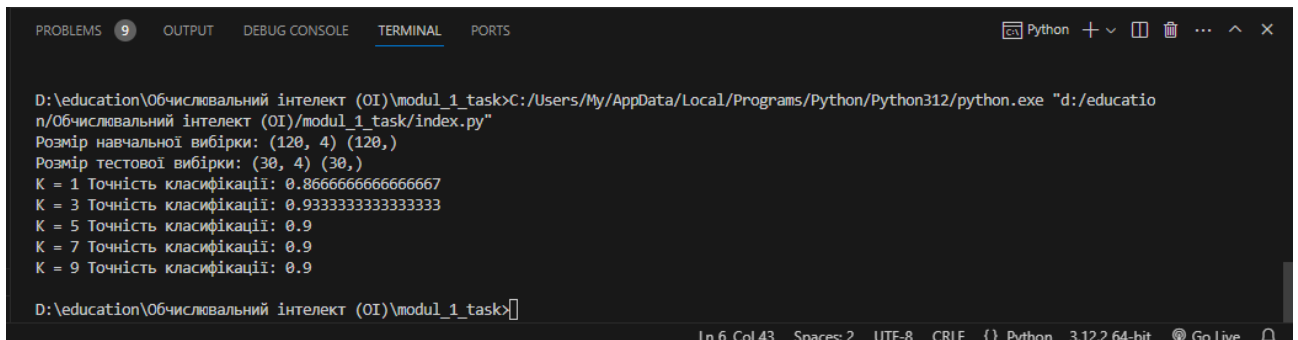
    # Навчання класифікатора на навчальних даних
    knn_classifier.fit(X_train, y_train)

    # Прогнозування класів для тестових даних
    y_pred = knn_classifier.predict(X_test)

    # Оцінка точності класифікації
    accuracy = accuracy_score(y_test, y_pred)

    # Виведення результатів
    print("K =", k, "Точність класифікації:", accuracy)
```

У цьому коді ми спочатку створюємо список різних значень K, які хочемо випробувати. Потім ми використовуємо цикл для навчання та оцінки класифікатора для кожного значення K. Для кожного значення K ми створюємо KNN-класифікатор, навчаємо його на навчальних даних, прогнозуємо класи для тестових даних, обчислюємо точність класифікації та виводимо результати.



```
D:\education\Обчислювальний інтелект (OI)\modul_1_task>C:/Users/My/AppData/Local/Programs/Python/Python312/python.exe "d:/education/Обчислювальний інтелект (OI)/modul_1_task/index.py"
Розмір навчальної вибірки: (120, 4) (120,)
Розмір тестової вибірки: (30, 4) (30,)
K = 1 Точність класифікації: 0.8666666666666667
K = 3 Точність класифікації: 0.9333333333333333
K = 5 Точність класифікації: 0.9
K = 7 Точність класифікації: 0.9
K = 9 Точність класифікації: 0.9
D:\education\Обчислювальний інтелект (OI)\modul_1_task>
```

Рисунок 1.6 KNN-класифікатор з різними значеннями K

Крок 6. Вибрати величину K для найкращих показників якості класифікацій у тестовій вибірці

Щоб вибрати найкращу величину K для найкращих показників якості класифікації у тестовій вибірці, ми можемо обирати ту величину K, яка має найвищу точність класифікації на тестовій вибірці. Ось як це можна зробити:

```
best_accuracy = 0
best_k = 0

for k in k_values:
    knn_classifier = KNeighborsClassifier(n_neighbors=k)
    knn_classifier.fit(X_train, y_train)
    y_pred = knn_classifier.predict(X_test)
```



```

accuracy = accuracy_score(y_test, y_pred)

if accuracy > best_accuracy:
    best_accuracy = accuracy
    best_k = k

print("Найкраща величина K:", best_k)
print("Точність класифікації для найкращої величини K:", best_accuracy)

```

У цьому коді ми перебираємо різні значення K і зберігаємо найкращу точність та відповідну величину K. По закінченню циклу ми виводимо найкращу величину K та відповідну точність класифікації.

```

D:\education\Обчислювальний інтелект (OI)\modul_1_task>C:/Users/My/AppData/Local/Programs/Python/Python312/python.exe "d:/education/Обчислювальний інтелект (OI)/modul_1_task/index.py"
Розмір навчальної вибірки: (120, 4) (120,)
Розмір тестової вибірки: (30, 4) (30,)
K = 1 Точність класифікації: 0.9
K = 3 Точність класифікації: 0.9
K = 5 Точність класифікації: 0.8666666666666667
K = 7 Точність класифікації: 0.9333333333333333
K = 9 Точність класифікації: 0.9333333333333333
Найкраща величина K: 7
Точність класифікації для найкращої величини K: 0.9333333333333333
D:\education\Обчислювальний інтелект (OI)\modul_1_task>

```

Рисунок 1.7 Вивід найкращої величини K та точність класифікації.

В цьому розділі успішно створено KNN – класифікатора у Python. Для роботи було завантажено iris dataset. Виконано покрокове доповнення коду програми для реалізації завдання.

## Завдання 2. Створення KNN – регресора у Python

Розробити програмну реалізацію Python, яка забезпечує виконання наступних кроків:

Крок 1. Згенерувати випадковий набір даних в діапазоні 1000 значень

Давайте розпочнемо з генерації випадкового набору даних в діапазоні з 1000 значень. Для цього ми можемо скористатися бібліотекою NumPy. Ось код, щоб здійснити цей перший крок:

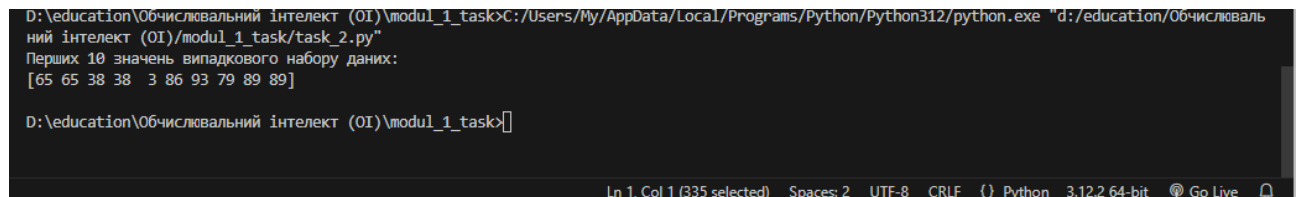
```
import numpy as np

# Задаємо діапазон значень
lower_bound = 0
upper_bound = 100

# Генеруємо випадковий набір даних з розмірністю 1000
random_data = np.random.randint(lower_bound, upper_bound, size=1000)

print("Перших 10 значень випадкового набору даних:")
print(random_data[:10]) # Виведемо перші 10 значень для перевірки
```

Цей код згенерує випадковий набір даних у діапазоні від 0 до 100 з розмірністю 1000. Відповідний вивід покаже перші 10 значень цього набору даних.



```
D:\education\Обчислювальний інтелект (OI)\modul_1_task\task_2.py
Перших 10 значень випадкового набору даних:
[65 65 38 38 3 86 93 79 89 89]
```

Рисунок 2.1 Виконання першого кроку

Крок 2. Нормалізувати значення.

Для цього ми можемо використати стандартне відхилення та середнє значення. Ось як ми можемо зробити це:

```
import numpy as np

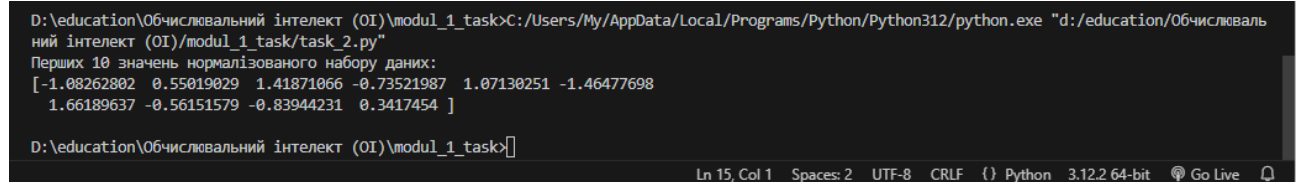
# Задаємо діапазон значень
lower_bound = 0
upper_bound = 100

# Генеруємо випадковий набір даних з розмірністю 1000
random_data = np.random.randint(lower_bound, upper_bound, size=1000)

# Нормалізуємо значення
normalized_data = (random_data - np.mean(random_data)) / np.std(random_data)
```

```
print("Перших 10 значень нормалізованого набору даних:")
print(normalized_data[:10]) # Виведемо перші 10 значень для перевірки
```

Цей код використовує стандартне відхилення та середнє значення для нормалізації нашого випадкового набору даних. Він виведе перші 10 нормалізованих значень для перевірки.

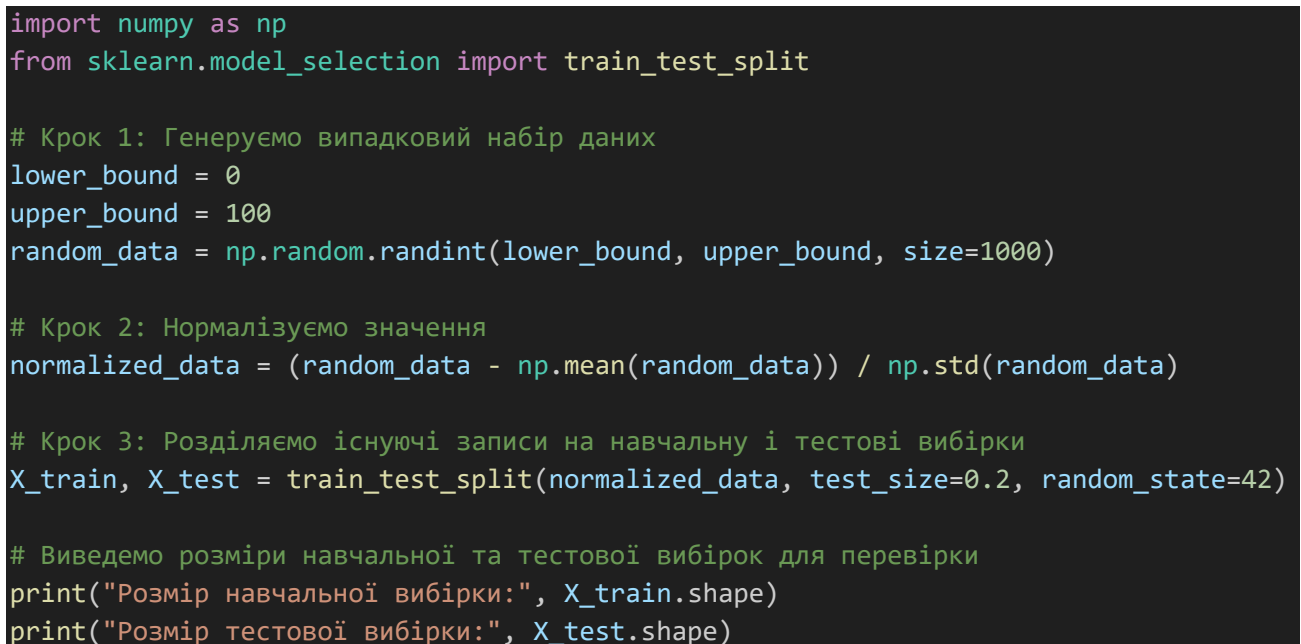


```
D:\education\Обчислювальний інтелект (OI)\modul_1_task>C:/Users/My/AppData/Local/Programs/Python/Python312/python.exe "d:/education/Обчислювальний інтелект (OI)/modul_1_task/task_2.py"
Перших 10 значень нормалізованого набору даних:
[-1.08262802  0.55019029  1.41871066 -0.73521987  1.07130251 -1.46477698
 1.66189637 -0.56151579 -0.83944231  0.3417454 ]
D:\education\Обчислювальний інтелект (OI)\modul_1_task>
```

Рисунок 2.2 Нормалізовані значення

### Крок 3. Розділити існуючі записи на навчальну і тестові вибірки

Для розділення існуючих записів на навчальну і тестові вибірки ми можемо скористатися функцією `train_test_split` з бібліотеки `sklearn`. Ось як ми можемо зробити це:



```
import numpy as np
from sklearn.model_selection import train_test_split

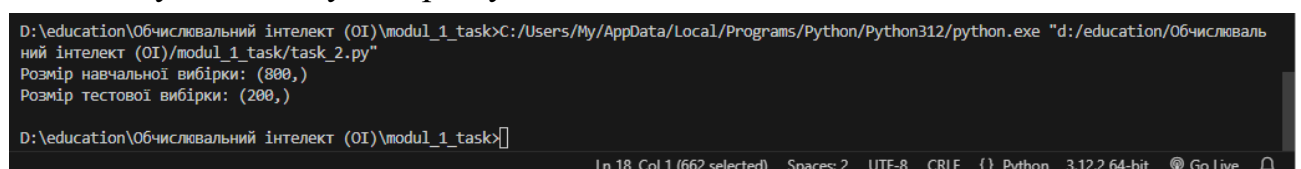
# Крок 1: Генеруємо випадковий набір даних
lower_bound = 0
upper_bound = 100
random_data = np.random.randint(lower_bound, upper_bound, size=1000)

# Крок 2: Нормалізуємо значення
normalized_data = (random_data - np.mean(random_data)) / np.std(random_data)

# Крок 3: Розділяємо існуючі записи на навчальну і тестові вибірки
X_train, X_test = train_test_split(normalized_data, test_size=0.2, random_state=42)

# Виведемо розміри навчальної та тестової вибірок для перевірки
print("Розмір навчальної вибірки:", X_train.shape)
print("Розмір тестової вибірки:", X_test.shape)
```

У цьому коді ми використовуємо бібліотеку `train_test_split` з `sklearn.model_selection` для розділення нашого нормалізованого набору даних на навчальну та тестову вибірки у співвідношенні 80/20 відповідно.



```
D:\education\Обчислювальний інтелект (OI)\modul_1_task>C:/Users/My/AppData/Local/Programs/Python/Python312/python.exe "d:/education/Обчислювальний інтелект (OI)/modul_1_task/task_2.py"
Розмір навчальної вибірки: (800,)
Розмір тестової вибірки: (200,)
D:\education\Обчислювальний інтелект (OI)\modul_1_task>
```

Рисунок 2.3 Виконання коду

#### Крок 4. Навчити KNN-регресор з різними значеннями K

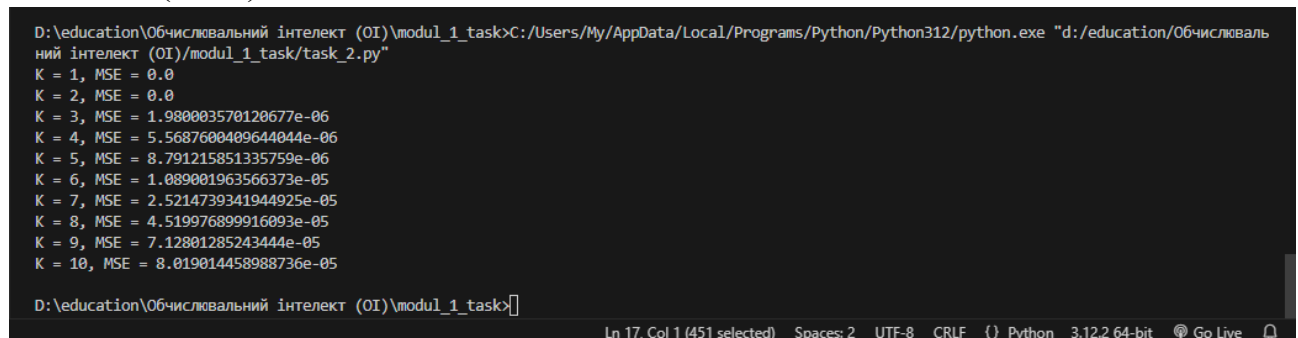
Для цього кроку потрібно включити в код нижче зазначені бібліотки:

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
```

Та сам код доповнити кількома рядками:

```
# Крок 4: Навчаємо KNN-регресор з різними значеннями K
for k in range(1, 11):
    knn_regressor = KNeighborsRegressor(n_neighbors=k)
    knn_regressor.fit(X_train.reshape(-1, 1), X_train) # Навчання на навчальних
даних
    predictions = knn_regressor.predict(X_test.reshape(-1, 1)) # Прогнозування на
тестових даних
    mse = mean_squared_error(X_test, predictions) # Розрахунок
середньоквадратичної помилки
    print(f"K = {k}, MSE = {mse}")
```

У цьому фрагменті ми навчаємо KNN-регресор з різними значеннями K від 1 до 10 та оцінюємо якість прогнозів за допомогою середньоквадратичної помилки (MSE).



```
D:\education\Обчислювальний інтелект (OI)\modul_1_task>C:/Users/My/AppData/Local/Programs/Python/Python312/python.exe "d:/education/Обчислювальний інтелект (OI)/modul_1_task/task_2.py"
K = 1, MSE = 0.0
K = 2, MSE = 0.0
K = 3, MSE = 1.980003570120677e-06
K = 4, MSE = 5.5687600409644044e-06
K = 5, MSE = 8.791215851335759e-06
K = 6, MSE = 1.089001963566373e-05
K = 7, MSE = 2.5214739341944925e-05
K = 8, MSE = 4.519976899916093e-05
K = 9, MSE = 7.12801285243444e-05
K = 10, MSE = 8.019014458988736e-05
D:\education\Обчислювальний інтелект (OI)\modul_1_task>
```

Рисунок 2.4 Результат Обчислень

Крок 5. Вибрати величину K для найкращих показників якості регресії у тестовій вибірці

Для вибору оптимального значення K за найкращими показниками якості регресії у тестовій вибірці ми будемо порівнювати середньоквадратичну помилку (MSE) для різних значень K. Потім виберемо те значення K, для якого MSE найменше. Ось оновлений фрагмент код з для кроку №5:

```
best_k = None
best_mse = float('inf')
for k in range(1, 11):
    knn_regressor = KNeighborsRegressor(n_neighbors=k)
    knn_regressor.fit(X_train.reshape(-1, 1), X_train) # Навчання на навчальних
даних
    predictions = knn_regressor.predict(X_test.reshape(-1, 1)) # Прогнозування на
тестових даних
```

```

    mse = mean_squared_error(X_test, predictions) # Розрахунок
середньоквадратичної помилки
    if mse < best_mse:
        best_mse = mse
        best_k = k

print(f"Найкраще значення K: {best_k} з MSE = {best_mse}")

```

У цьому коді ми проходимо по різним значенням K від 1 до 10, навчаємо модель за кожним значенням K, оцінюємо якість за допомогою MSE і обираємо те значення K, для якого MSE найменше.

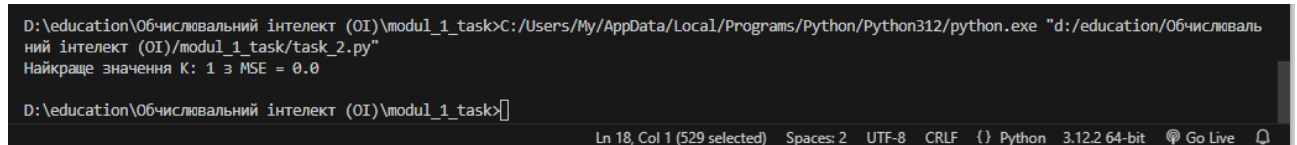


Рисунок 2.5 Реалізація кроку №5

### Крок 6. Здійснити візуалізації отриманих рішень

У цьому кроці потрібно використати бібліотеку Matplotlib для візуалізації отриманих результатів. Також ось повний код програми для вирішення завдання №2:

```

import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error

# Крок 1: Генеруємо випадковий набір даних
lower_bound = 0
upper_bound = 100
random_data = np.random.randint(lower_bound, upper_bound, size=1000)

# Крок 2: Нормалізуємо значення
normalized_data = (random_data - np.mean(random_data)) / np.std(random_data)

# Крок 3: Розділяємо існуючі записи на навчальну і тестові вибірки
X_train, X_test = train_test_split(normalized_data, test_size=0.2, random_state=42)

# Крок 4: Навчаємо KNN-регресор з різними значеннями K та оцінюємо якість
best_k = None
best_mse = float('inf')
mse_values = []
for k in range(1, 11):
    knn_regressor = KNeighborsRegressor(n_neighbors=k)
    knn_regressor.fit(X_train.reshape(-1, 1), X_train) # Навчання на навчальних
даних
    predictions = knn_regressor.predict(X_test.reshape(-1, 1)) # Прогнозування на
тестових даних

```

```

    mse = mean_squared_error(X_test, predictions) # Розрахунок
середньоквадратичної помилки
    mse_values.append(mse)
    if mse < best_mse:
        best_mse = mse
        best_k = k

print(f"Найкраще значення K: {best_k} з MSE = {best_mse}")

# Крок 6: Візуалізація отриманих результатів
plt.figure(figsize=(10, 6))
plt.plot(range(1, 11), mse_values, marker='o', linestyle='-', color='b')
plt.title('Залежність MSE від значення K')
plt.xlabel('Значення K')
plt.ylabel('MSE')
plt.xticks(range(1, 11))
plt.grid(True)
plt.show()

```

У цьому коді ми створюємо графік, на якому відображаємо залежність середньоквадратичної помилки (MSE) від значення K. Таким чином, ми можемо візуально порівняти якість регресії для різних значень K.

```

D:\education\Обчислювальний інтелект (OI)\modul_1_task>C:/Users/My/AppData/Local/Programs/Python/Python312/python.exe "d:/education/Обчислювальний інтелект (OI)/modul_1_task/task_2.py"
Найкраще значення K: 1 з MSE = 0.0

```

Рисунок 2.6 Результат запуску коду

Також відповідно до завдання візуалізації коду було створено графік залежностей.

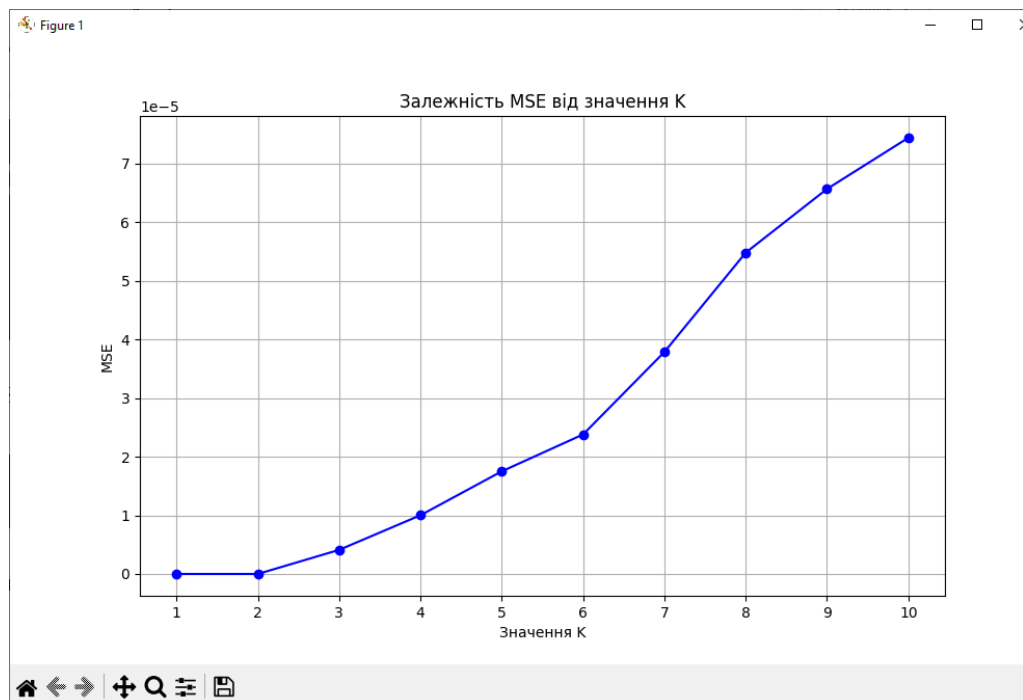


Рисунок 2.7 Візуалізації Результатів

## Висновок

У цій роботі було побудовано класифікатора та регресора методом k-найближчих сусідів (k-nn, як доказ важливості етапу у вивченні обчислювального інтелекту. Метод k-nn є одним із найпростіших та найефективніших алгоритмів машинного навчання для класифікації та регресії, він заснований на концепції використання схожості між прикладами даних.

Досягнуто мети даної роботи, а саме вивчено можливостей аналізу даних з використанням класифікатора та регресора методом k-найближчих сусідів (k-nn). У ході роботи було створено k-nn класифікатор та регресор у середовищі програмування Python.

Під час виконання ми створили класифікатор KNN на мові програмування Python, заважили базу параметрів, виконали операції над даними і вибрано величину K для найкращих показників якості класифікації у тестовій вибірці. Також розробили KNN – регресора у Python і обрано величину K для найкращих показників якості регресії у тестовій вибірці, здійснено візуалізацію отриманих рішень.