**ЛАБОРАТОРНА РОБОТА № 3**

**ДОСЛІДЖЕННЯ МЕТОДІВ РЕГРЕСІЇ**

***Мета роботи:*** використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити методи регресії даних у машинному навчанні.

Завдання 2.1

import pickle  
import numpy as np  
from sklearn import linear\_model  
import sklearn.metrics as sm  
import matplotlib.pyplot as plt  
  
# Вхідний файл, який містить дані  
input\_file = 'data\_singlevar\_regr.txt'  
  
# Завантаження даних  
data = np.loadtxt(input\_file, delimiter=',')  
X, y = data[:, :-1], data[:, -1]  
  
# Розбивка даних на навчальний та тестовий набори  
num\_training = int(0.8 \* len(X))  
num\_test = len(X) - num\_training  
  
# Тренувальні дані  
X\_train, y\_train = X[:num\_training], y[:num\_training]  
# Тестові дані  
X\_test, y\_test = X[num\_training:], y[num\_training:]  
  
# Створення об'єкта лінійного регресора  
regressor = linear\_model.LinearRegression()  
regressor.fit(X\_train, y\_train)  
  
# Прогнозування результату  
y\_test\_pred = regressor.predict(X\_test)  
  
# Побудова графіка  
plt.scatter(X\_test, y\_test, color='green')  
plt.plot(X\_test, y\_test\_pred, color='black', linewidth=4)  
plt.xticks(())  
plt.yticks(())  
plt.show()  
print("Продуктивність лінійної регресії:")  
print("Середня абсолютна похибка =", round(sm.mean\_absolute\_error(y\_test, y\_test\_pred), 2))  
print("Середня квадратична помилка =", round(sm.mean\_squared\_error(y\_test, y\_test\_pred), 2))  
print("Середня абсолютна помилка =", round(sm.median\_absolute\_error(y\_test, y\_test\_pred), 2))  
print("Пояснена оцінка дисперсії =", round(sm.explained\_variance\_score(y\_test,

y\_test\_pred), 2))  
print("R2 оцінка =", round(sm.r2\_score(y\_test, y\_test\_pred), 2))  
output\_model\_file = 'model.pkl'  
  
# Збереження моделі  
with open(output\_model\_file, 'wb') as f:  
 pickle.dump(regressor, f)  
  
# Завантаження моделі  
y\_test\_pred\_new = regressor.predict(X\_test)  
print("\nНова середня абсолютна помилка =", round(sm.mean\_absolute\_error(y\_test, y\_test\_pred\_new), 2))

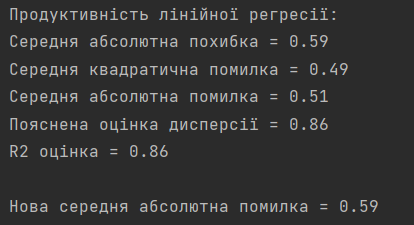


Рис.1 Результат виконання

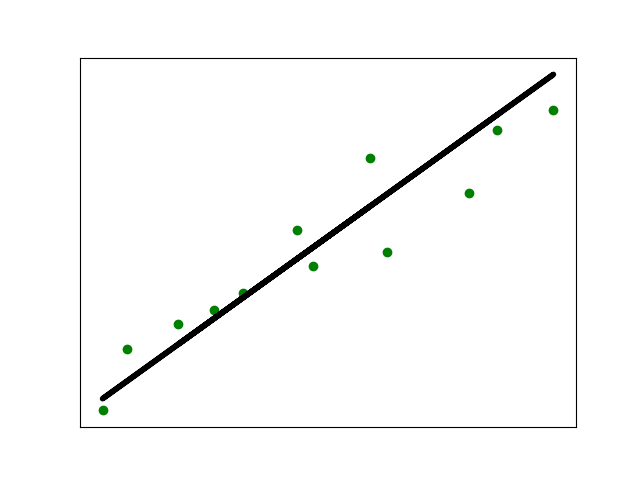


Рис. 2 Графік функції

Висновок: ми можемо використовувати цей спосіб для статистичного аналізу, який намагається показати зв’язок між двома змінними. Лінійна регресія може створити модель прогнозування за ніби-то випадковими даними, показуючи тенденцію в даних. Наприклад для цін або акцій.

Завдання 2.2

import pickle  
import numpy as np  
from sklearn import linear\_model  
import sklearn.metrics as sm  
import matplotlib.pyplot as plt  
  
# Вхідний файл, який містить дані  
input\_file = 'data\_regr\_4.txt'  
  
# Завантаження даних  
data = np.loadtxt(input\_file, delimiter=',')  
X, y = data[:, :-1], data[:, -1]  
  
# Розбивка даних на навчальний та тестовий набори  
num\_training = int(0.8 \* len(X))  
num\_test = len(X) - num\_training  
  
# Тренувальні дані  
X\_train, y\_train = X[:num\_training], y[:num\_training]  
# Тестові дані  
X\_test, y\_test = X[num\_training:], y[num\_training:]  
  
# Створення об'єкта лінійного регресора  
regressor = linear\_model.LinearRegression()  
regressor.fit(X\_train, y\_train)  
  
# Прогнозування результату  
y\_test\_pred = regressor.predict(X\_test)  
  
# Побудова графіка  
plt.scatter(X\_test, y\_test, color='green')  
plt.plot(X\_test, y\_test\_pred, color='black', linewidth=4)  
plt.xticks(())  
plt.yticks(())  
plt.show()  
  
print("Продуктивність лінійної регресії:")  
print("Середня абсолютна похибка =", round(sm.mean\_absolute\_error(y\_test, y\_test\_pred), 2))  
print("Середня квадратична помилка =", round(sm.mean\_squared\_error(y\_test, y\_test\_pred), 2))  
print("Середня абсолютна помилка =", round(sm.median\_absolute\_error(y\_test, y\_test\_pred), 2))  
print("Пояснена оцінка дисперсії =", round(sm.explained\_variance\_score(y\_test, y\_test\_pred), 2))  
print("R2 оцінка =", round(sm.r2\_score(y\_test, y\_test\_pred), 3))  
  
output\_model\_file = 'model.pkl'  
  
# Збереження моделі  
with open(output\_model\_file, 'wb') as f:  
 pickle.dump(regressor, f)  
  
# Завантаження моделі  
y\_test\_pred\_new = regressor.predict(X\_test)  
print("\nНова середня абсолютна помилка =", round(sm.mean\_absolute\_error(y\_test, y\_test\_pred\_new), 2))

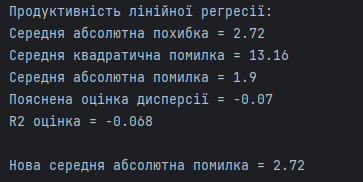


Рис.3 Результат виконання

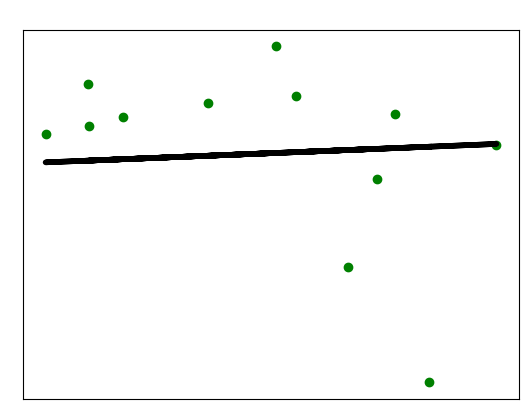


Рис. 4 Графік функції

Висновок: з графіка видно, що залишки розподілені не рівномірно щодо горизонтальної осі. Виходячи з R 2 оцінки можна зробити висновок, що продуктивність цієї моделі машинного навчання на основі регресії є поганою.

Завдання 2.3

import pickle  
import numpy as np  
from sklearn import linear\_model  
import sklearn.metrics as sm  
import matplotlib.pyplot as plt  
from sklearn.preprocessing import PolynomialFeatures  
  
# Вхідний файл, який містить дані  
input\_file = 'data\_multivar\_regr.txt'  
  
# Завантаження даних  
data = np.loadtxt(input\_file, delimiter=',')  
X, y = data[:, :-1], data[:, -1]  
  
# Розбивка даних на навчальний та тестовий набори  
num\_training = int(0.8 \* len(X))  
num\_test = len(X) - num\_training  
# Тренувальні дані  
X\_train, y\_train = X[:num\_training], y[:num\_training]  
# Тестові дані  
X\_test, y\_test = X[num\_training:], y[num\_training:]  
  
# Створення об'єкта лінійного регресора  
regressor = linear\_model.LinearRegression()  
regressor.fit(X\_train, y\_train)  
  
# Прогнозування результату  
y\_test\_pred = regressor.predict(X\_test)  
  
print("Продуктивність лінійної регресії:")  
print("Середня абсолютна похибка =", round(sm.mean\_absolute\_error(y\_test, y\_test\_pred), 2))  
print("Середня квадратична помилка =", round(sm.mean\_squared\_error(y\_test, y\_test\_pred), 2))  
print("Середня абсолютна помилка =", round(sm.median\_absolute\_error(y\_test, y\_test\_pred), 2))  
print("Пояснена оцінка дисперсії =", round(sm.explained\_variance\_score(y\_test, y\_test\_pred), 2))  
print("R2 оцінка =", round(sm.r2\_score(y\_test, y\_test\_pred), 3))  
  
output\_model\_file = 'model.pkl'  
  
# Збереження моделі  
with open(output\_model\_file, 'wb') as f:  
 pickle.dump(regressor, f)  
  
# Завантаження моделі  
y\_test\_pred\_new = regressor.predict(X\_test)  
print("\nНова середня абсолютна помилка =", round(sm.mean\_absolute\_error(y\_test, y\_test\_pred\_new), 2))  
# Поліноміальна регресія  
polynomial = PolynomialFeatures(degree=10)  
X\_train\_transformed = polynomial.fit\_transform(X\_train)  
  
datapoint = [[7.75, 6.35, 5.56]]  
poly\_datapoint = polynomial.fit\_transform(datapoint)  
  
poly\_linear\_model = linear\_model.LinearRegression()  
poly\_linear\_model.fit(X\_train\_transformed, y\_train)  
  
print("\nLinear regression:\n", regressor.predict(datapoint))  
print("\nPolynomial regression:\n", poly\_linear\_model.predict(poly\_datapoint))

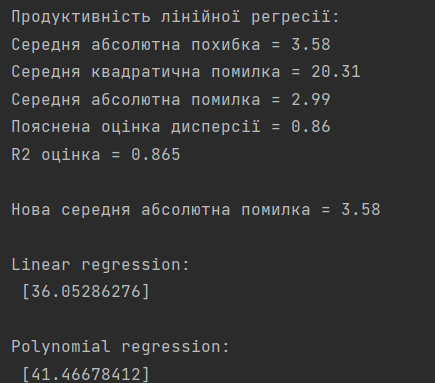


Рис. 5 Результат виконання

Для цього коду рекомендовано використовувати поліномінальну регресію з кількох причин. По-перше, поліноміальна регресія ефективна, коли залежність між вхідними та вихідними даними є складною. Результати показали, що поліноміальна регресія (з рівнем 10) передбачає значення ознаки для точки даних datapoint краще (41.46), ніж лінійна регресія (36.05). Отже, у даному випадку поліноміальна регресія є кращим вибором для точнішого передбачення результатів.

По-друге, використання поліноміальних ознак (у цьому коді ступеню 10) робить модель більш гнучкою та допомагає краще прогнозувати дані. Важливо правильно вибрати ступінь полінома та налаштувати складність моделі, щоб уникнути перенавчання та забезпечити краще відповідання вхідним даним.

Завдання 2.4

import matplotlib.pyplot as plt  
import numpy as np  
from sklearn import datasets, linear\_model  
from sklearn.metrics import mean\_squared\_error, r2\_score, mean\_absolute\_error  
  
# Load the diabetes dataset  
diabetes\_X, diabetes\_y = datasets.load\_diabetes(return\_X\_y=True)  
  
# Use only one feature  
diabetes\_X = diabetes\_X[:, np.newaxis, 2]  
# Split the data into training/testing sets  
diabetes\_X\_train = diabetes\_X[:-20]  
diabetes\_X\_test = diabetes\_X[-20:]  
  
# Split the targets into training/testing sets  
diabetes\_y\_train = diabetes\_y[:-20]  
diabetes\_y\_test = diabetes\_y[-20:]  
  
# Create linear regression object  
regr = linear\_model.LinearRegression()  
  
# Train the model using the training sets  
regr.fit(diabetes\_X\_train, diabetes\_y\_train)  
  
# Make predictions using the testing set  
diabetes\_y\_pred = regr.predict(diabetes\_X\_test)  
  
# The coefficients  
print("Regression coef: \n", regr.coef\_)  
print("Regression intercept: \n", regr.intercept\_)  
# Середня абсолютна похибка  
print("Mean absolute error :", round(mean\_absolute\_error(diabetes\_y\_test, diabetes\_y\_pred), 2))  
# The mean squared error  
print("Mean squared error: %.2f" % mean\_squared\_error(diabetes\_y\_test, diabetes\_y\_pred))  
  
# The coefficient of determination: 1 is perfect prediction  
print("R2 score: %.2f" % r2\_score(diabetes\_y\_test, diabetes\_y\_pred))  
  
fig, ax = plt.subplots()  
ax.scatter(diabetes\_y\_test, diabetes\_y\_pred, edgecolors=(0, 0, 0))  
ax.plot([diabetes\_y.min(), diabetes\_y.max()], [diabetes\_y.min(), diabetes\_y.max()], 'k--', lw=4)  
ax.set\_xlabel('Виміряно')  
ax.set\_ylabel('Передбачено')  
plt.show()

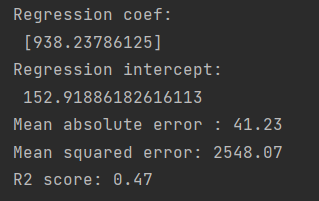


Рис. 6 Результат виконання

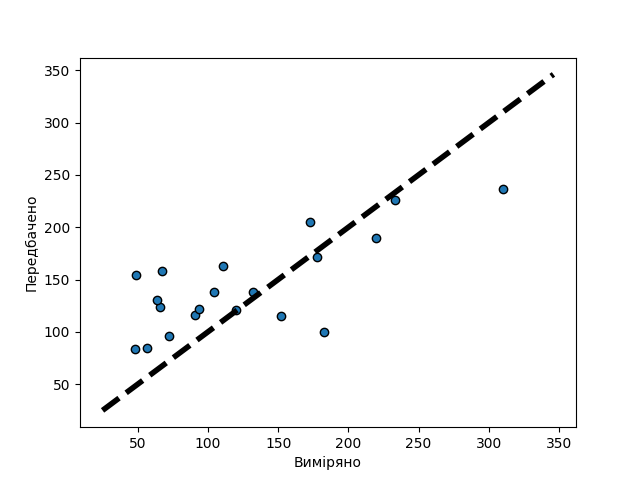
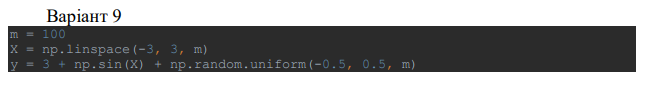


Рис. 7 Графік функції

Кожна точка на графіку представляє пару значень, де одна координата - фактичне значення, а інша - передбачене значення. Пунктирна лінія є опорною лінією, яка вказує на те, де могли б розташовуватися точки, якби передбачення моделі були абсолютно точними. Якщо точка на графіку знаходиться близько до діагональної лінії, це вказує на те, що передбачення моделі добре відповідають фактичним даним.

Завдання 2.5



import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.linear\_model import LinearRegression  
from sklearn.metrics import mean\_squared\_error, r2\_score  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import PolynomialFeatures  
  
m = 100  
X = np.linspace(-3, 3, m)  
y = 3 + np.sin(X) + np.random.uniform(-0.5, 0.5, m)  
  
X = X.reshape(-1, 1)  
Y = y.reshape(-1, 1)  
  
lin = LinearRegression()  
lin.fit(X, y)  
  
poly = PolynomialFeatures(degree=2)  
X\_poly = poly.fit\_transform(X)  
poly.fit(X\_poly, y)  
lin2 = LinearRegression()  
lin2.fit(X\_poly, y)  
  
Y\_NEW = lin2.predict(X\_poly)  
r2 = r2\_score(Y, Y\_NEW)  
  
print('R2: ', r2)  
  
# Visualising the Linear Regression results  
plt.scatter(X, y, color='blue')  
plt.plot(X, lin.predict(X), color='red')  
plt.title('Linear Regression')  
plt.show()  
  
# Visualising the Polynomial Regression results  
plt.scatter(X, y, color='blue')  
plt.plot(X, lin2.predict(poly.fit\_transform(X)), color='red')  
plt.title('Polynomial Regression')  
plt.show()



Рис. 8 Результат виконання

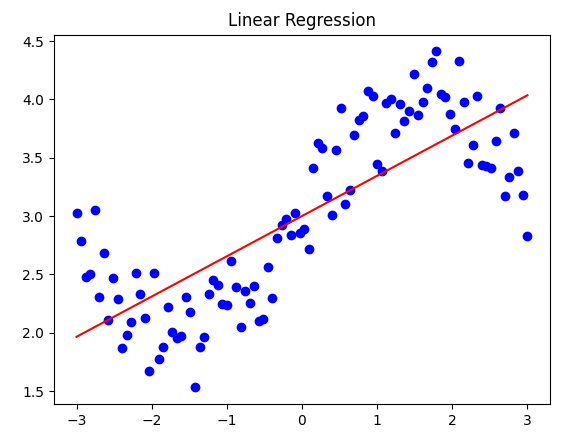


Рис. 9 Лінійна регресія

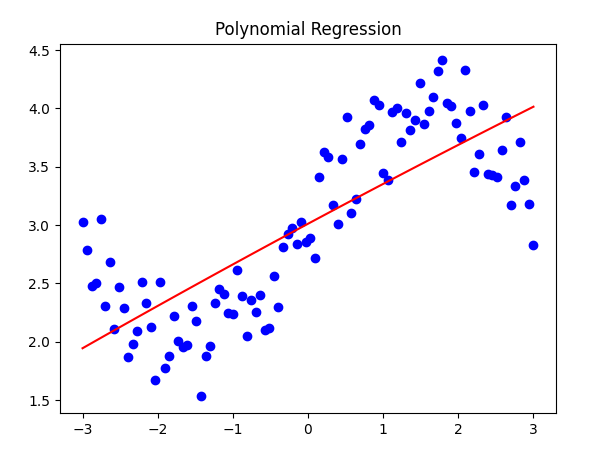


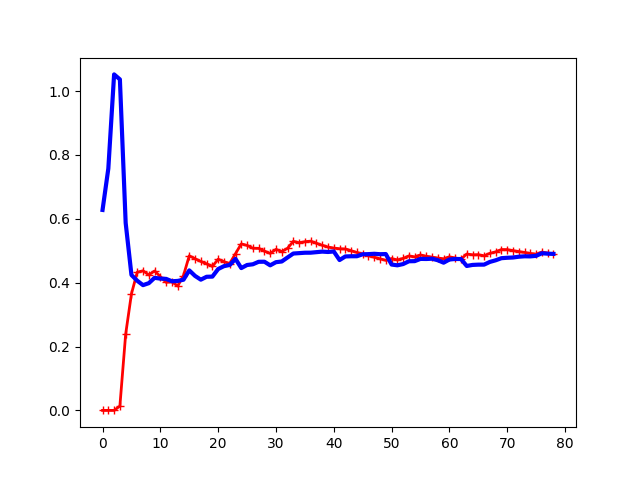
Рис. 10 Поліноміальна регресія

Завдання 2.6

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.linear\_model import LinearRegression  
from sklearn.metrics import mean\_squared\_error, r2\_score  
from sklearn.model\_selection import train\_test\_split  
from sklearn.pipeline import Pipeline  
from sklearn.preprocessing import PolynomialFeatures  
  
def plot\_learning\_curves(model, X, y):  
 X\_train, X\_val, y\_train, y\_val = train\_test\_split(X, y, test\_size=0.2)  
 train\_errors, val\_errors = [], []  
 for m in range(1, len(X\_train)):  
 model.fit(X\_train[:m], y\_train[:m])  
 y\_train\_predict = model.predict(X\_train[:m])  
 y\_val\_predict = model.predict(X\_val)  
 train\_errors.append(mean\_squared\_error(y\_train\_predict, y\_train[:m]))  
 val\_errors.append(mean\_squared\_error(y\_val\_predict, y\_val))  
 plt.plot(np.sqrt(train\_errors), "r-+", linewidth=2, label='train')  
 plt.plot(np.sqrt(val\_errors), "b-", linewidth=3, label='val')  
 plt.show()  
  
m = 100  
X = np.linspace(-3, 3, m)  
y = np.sin(X) + np.random.uniform(-0.5, 0.5, m)  
X = X.reshape(-1, 1)  
Y = y.reshape(-1, 1)  
lin = LinearRegression()  
lin.fit(X, y)  
  
poly = PolynomialFeatures(degree=2, include\_bias=False)  
X\_poly = poly.fit\_transform(X)  
  
poly.fit(X\_poly, y)  
lin2 = LinearRegression()  
lin2.fit(X\_poly, y)  
  
Y\_NEW = lin2.predict(X\_poly)  
r2 = r2\_score(Y, Y\_NEW)  
  
print('R2: ', r2)  
  
polynomial\_regg = Pipeline([("poly\_features", PolynomialFeatures(degree=2, include\_bias=False)),  
 ("lin\_reg", LinearRegression()),])  
plot\_learning\_curves(polynomial\_regg, X, y)

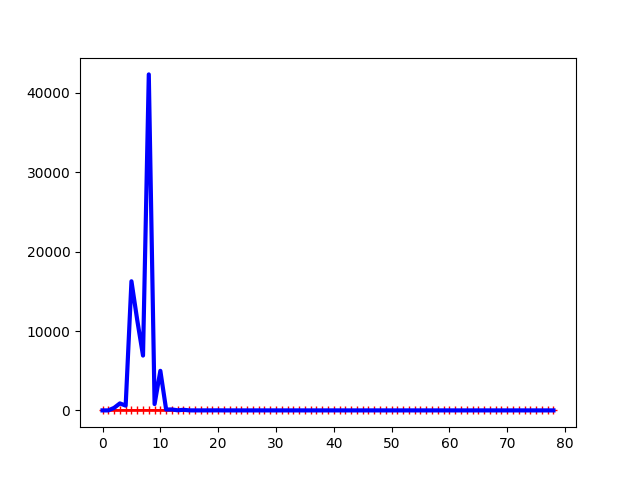


Рис. 11 Результат виконання



0,2

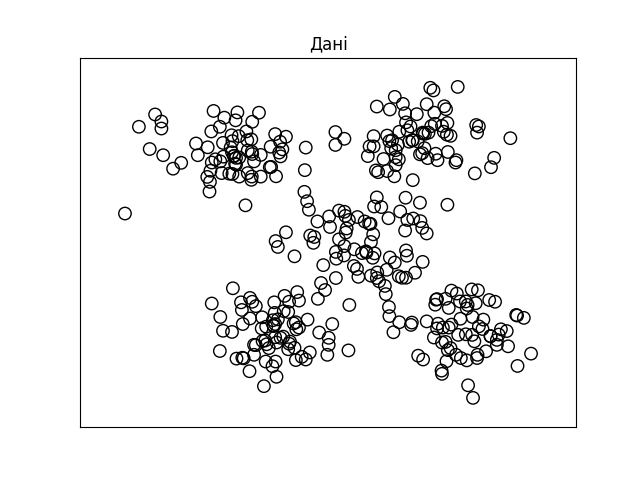




1,0

Завдання 2.7

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.cluster import KMeans  
from sklearn import metrics  
  
X = np.loadtxt('data\_clustering.txt', delimiter=',')  
  
num\_clusters = 5  
  
plt.figure()  
plt.scatter(X[:, 0], X[:, 1], marker='o',  
 facecolors='none', edgecolors='black', s=80)  
  
x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max()+1  
y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max()+1  
  
plt.title('Дані')  
plt.xlim(x\_min, x\_max)  
plt.ylim(y\_min, y\_max)  
plt.xticks(())  
plt.yticks(())  
plt.show()  
  
kmeans = KMeans(init='k-means++', n\_clusters=num\_clusters, n\_init=10)  
kmeans.fit(X)  
step\_size = 0.01  
  
x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max()+1  
y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max()+1  
  
x\_vals, y\_vals = np.meshgrid(  
 np.arange(x\_min, x\_max, step\_size), np.arange(y\_min, y\_max, step\_size))  
output = kmeans.predict(np.c\_[x\_vals.ravel(), y\_vals.ravel()])  
  
output = output.reshape(x\_vals.shape)  
plt.figure()  
plt.clf()  
plt.imshow(output, interpolation='nearest', extent=(x\_vals.min(), x\_vals.max(  
), y\_vals.min(), y\_vals.max()), cmap=plt.cm.Paired, aspect='auto', origin='lower')  
  
plt.scatter(X[:, 0], X[:, 1], marker='o',  
 facecolors='none', edgecolors='black', s=80)  
  
cluster\_center = kmeans.cluster\_centers\_  
plt.scatter(cluster\_center[:, 0], cluster\_center[:, 1], marker='o',  
 s=210, linewidths=4, color='black', zorder=12, facecolors='none')  
  
x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max()+1  
y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max()+1  
  
plt.title('Границі')  
plt.xlim(x\_min, x\_max)  
plt.ylim(y\_min, y\_max)  
plt.xticks(())  
plt.yticks(())  
plt.show()



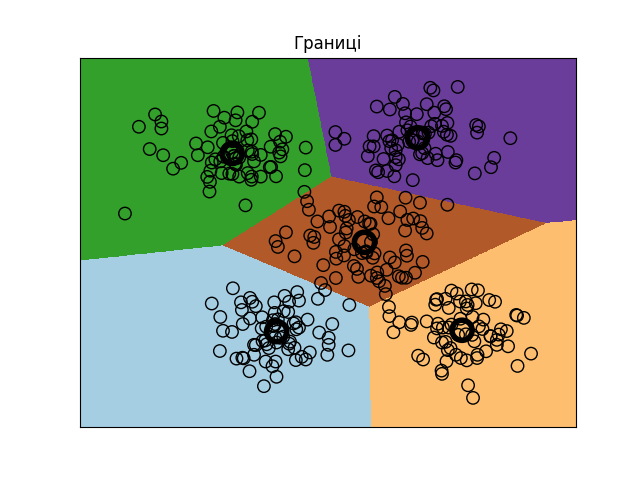
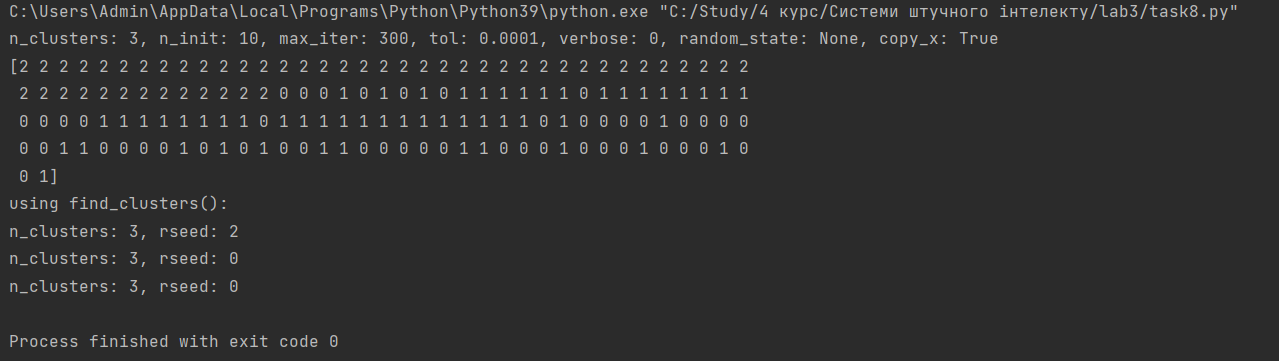
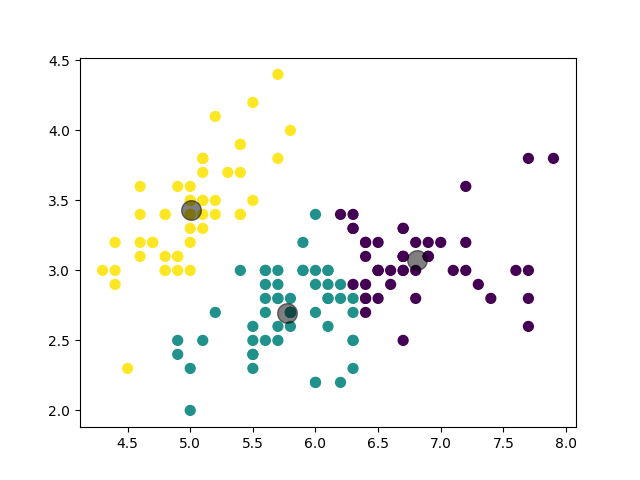


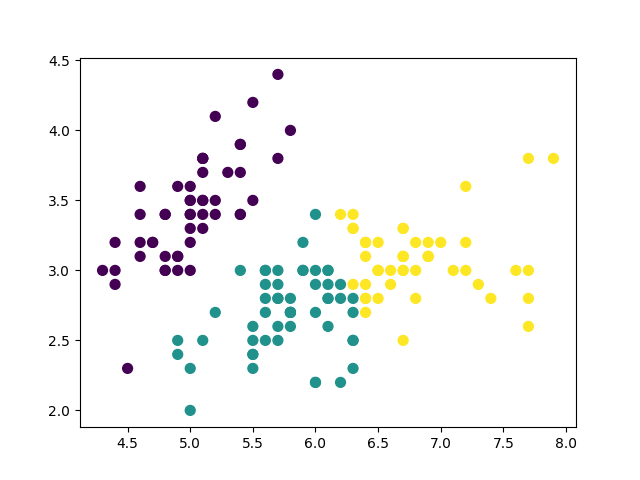
Рис. 12 Результат виконання

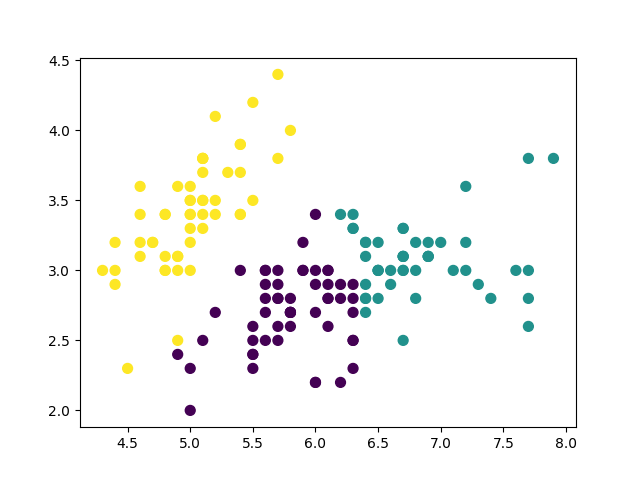
Завдання 2.8

import matplotlib.pyplot as plt  
from sklearn import datasets  
from sklearn.cluster import KMeans  
from sklearn.metrics import pairwise\_distances\_argmin  
import numpy as np  
  
iris = datasets.load\_iris()  
X = iris.data[:, :2]  
Y = iris.target  
  
kmeans = KMeans(n\_clusters=Y.max() + 1, init='k-means++', n\_init=10, max\_iter=300,  
 tol=0.0001, verbose=0, random\_state=None, copy\_x=True)  
kmeans.fit(X)  
y\_pred = kmeans.predict(X)  
  
print("n\_clusters: 3, n\_init: 10, max\_iter: 300, tol: 0.0001, verbose: 0, random\_state: None, copy\_x: True")  
print(y\_pred)  
plt.figure()  
plt.scatter(X[:, 0], X[:, 1], c=y\_pred, s=50, cmap='viridis')  
centers = kmeans.cluster\_centers\_  
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5)  
plt.show()  
  
  
def find\_clusters(X, n\_clusters, rseed=2):  
 rng = np.random.RandomState(rseed)  
 i = rng.permutation(X.shape[0])[:n\_clusters]  
 centers = X[i]  
  
 while True:  
 labels = pairwise\_distances\_argmin(X, centers)  
 new\_centers = np.array([X[labels == i].mean(0) for i in range(n\_clusters)])  
 if np.all(centers == new\_centers):  
 break  
 centers = new\_centers  
 return centers, labels  
  
  
print("using find\_clusters():")  
centers, labels = find\_clusters(X, 3)  
print("n\_clusters: 3, rseed: 2")  
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')  
plt.show()  
  
centers, labels = find\_clusters(X, 3, rseed=0)  
print("n\_clusters: 3, rseed: 0")  
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')  
plt.show()  
  
labels = KMeans(3, random\_state=0).fit\_predict(X)  
print("n\_clusters: 3, rseed: 0")  
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')  
plt.show()









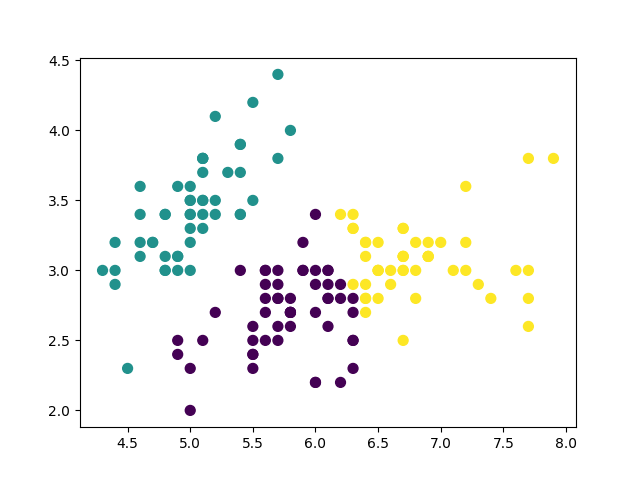


Рис. 13 Результат виконання

Код ілюструє приклади кластеризації даних за допомогою моделі KMeans з різними параметрами. Аналізуючи результати кластеризації при зміні параметрів і методів ініціалізації, можна визначити, як ці зміни впливають на результати. Це дозволяє визначити найкращий метод кластеризації для конкретної задачі та оптимізувати аналіз даних.

Завдання 2.9

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.cluster import MeanShift, estimate\_bandwidth  
from itertools import cycle  
  
X = np.loadtxt('data\_clustering.txt', delimiter=',')  
bandwidth = estimate\_bandwidth(X, quantile=0.2, n\_samples=500)  
ms = MeanShift(bandwidth=bandwidth, bin\_seeding=True)  
ms.fit(X)  
  
cluster\_centers = ms.cluster\_centers\_  
labels = ms.labels\_  
  
print("cluster\_centers:\n", cluster\_centers)  
print("labels:\n", labels)  
  
plt.figure()  
markers = cycle('o\*sv')  
colors = cycle('bgrcmyk')  
for i, marker in zip(range(len(cluster\_centers)), markers):  
 plt.scatter(X[labels == i, 0], X[labels == i, 1], marker=marker, color=next(colors), s=50, label='cluster ' + str(i))  
 cluster\_center = cluster\_centers[i]  
 plt.plot(cluster\_center[0], cluster\_center[1], marker='o', markerfacecolor='k', markeredgecolor='k', markersize=15)  
plt.title(f'Estimated number of clusters: {len(cluster\_centers)}')  
plt.show()

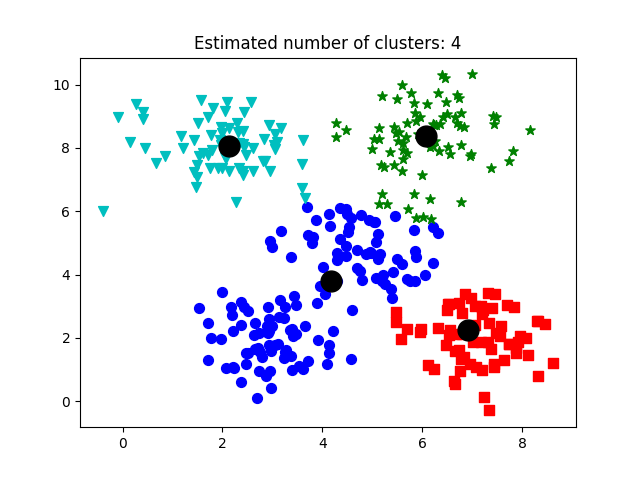


Рис. 14 Результат виконання

Наданий код ілюструє використання алгоритму MeanShift для кластеризації. У виводі вказані центри кластерів та їх кількість. Цей алгоритм дозволяє автоматично визначати кількість кластерів, що є корисним у випадках, коли заздалегідь невідомо, скільки кластерів слід шукати.

***Висновки:*** використовуючи спеціалізовані бібліотеки та мову програмування Python дослідив методи регресії даних у машинному навчанні.

GitHub: https://github.com/unravee1/AI\_labs