**ЛАБОРАТОРНА РОБОТА № 3**

ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ ДАНИХ

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

**Хід роботи:**

**Посилання на репозиторій: https://github.com/AlexanderSydorchuk/AI-Lab3**

**Завдання 1.** Створення регресора однієї змінної

Побудувати регресійну модель на основі однієї змінної.

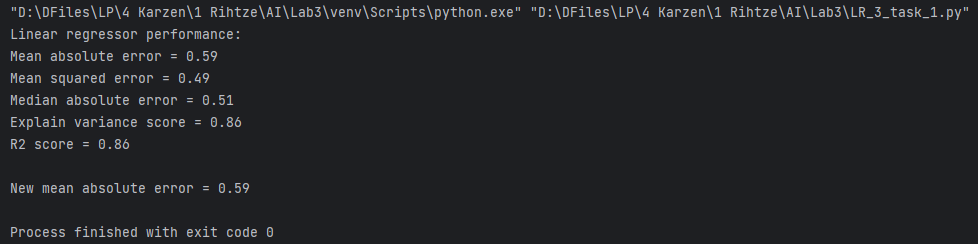
Використовувати файл вхідних даних: data\_singlevar\_regr.txt.

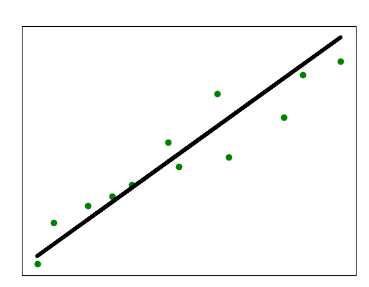
Лістинг програми:

import pickle  
import numpy as np  
from sklearn import linear\_model  
import sklearn.metrics as sm  
import matplotlib.pyplot as plt  
  
# Вхідний файл  
input\_file = 'data\_singlevar\_regr.txt'  
  
# Завантаження даних  
data = np.loadtxt(input\_file, delimiter=',')  
X, y = data[:, :-1], data[:, -1]  
  
num\_training = int(0.8 \* len(X))  
num\_test = len(X) - num\_training  
  
X\_train, y\_train = X[:num\_training], y[:num\_training]  
X\_test, y\_test = X[num\_training:], y[num\_training:]  
  
regressor = linear\_model.LinearRegression()  
regressor.fit(X\_train, y\_train)  
  
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("Linear regressor performance:")  
print("Mean absolute error =",  
 round(sm.mean\_absolute\_error(y\_test, y\_test\_pred), 2))  
print("Mean squared error =",  
 round(sm.mean\_squared\_error(y\_test, y\_test\_pred), 2))  
print("Median absolute error =",  
 round(sm.median\_absolute\_error(y\_test, y\_test\_pred), 2))  
print("Explain variance score =",  
 round(sm.explained\_variance\_score(y\_test, y\_test\_pred), 2))  
print("R2 score =", round(sm.r2\_score(y\_test, y\_test\_pred), 2))  
  
# Файл для збереження моделі  
output\_model\_file = 'model.pkl'  
with open(output\_model\_file, 'wb') as f:  
 pickle.dump(regressor, f)  
  
with open(output\_model\_file, 'rb') as f:  
 regressor\_model = pickle.load(f)  
  
y\_test\_pred\_new = regressor\_model.predict(X\_test)  
print("\nNew mean absolute error =",  
 round(sm.mean\_absolute\_error(y\_test, y\_test\_pred\_new), 2))

Результат виконання програми:





**Висновок:** модель для вихідних даних побудована правильно.Показники MAE, MSE вказують на середню якість, а показник R2 видає добрий результат.

**Завдання 2.** Побудувати регресійну модель на основі однієї змінної.

Використовувати вхідні дані відповідно свого варіанту, що визначається за

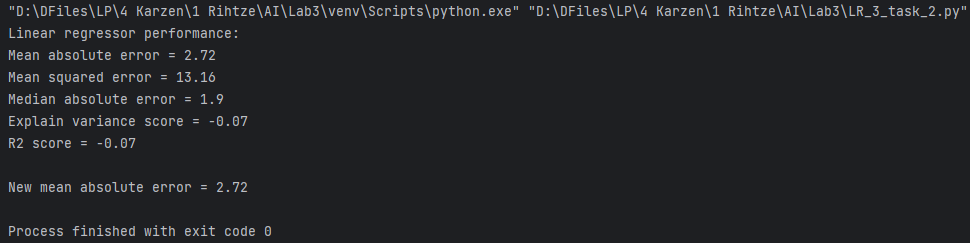
списком групи у журналі (таблиця 2.1).

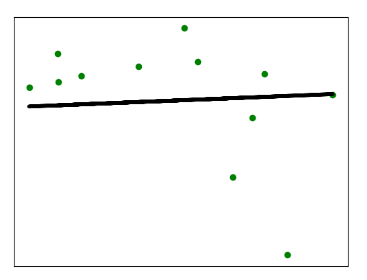


Лістинг програми:

import pickle  
import numpy as np  
from sklearn import linear\_model  
import sklearn.metrics as sm  
import matplotlib.pyplot as plt  
  
input\_file = 'data\_regr\_4.txt'  
  
data = np.loadtxt(input\_file, delimiter=',')  
X, y = data[:, :-1], data[:, -1]  
  
num\_training = int(0.8 \* len(X))  
num\_test = len(X) - num\_training  
  
X\_train, y\_train = X[:num\_training], y[:num\_training]  
X\_test, y\_test = X[num\_training:], y[num\_training:]  
  
regressor = linear\_model.LinearRegression()  
regressor.fit(X\_train, y\_train)  
  
y\_test\_pred = regressor.predict(X\_test)  
  
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("Linear regressor performance:")  
print("Mean absolute error =",  
 round(sm.mean\_absolute\_error(y\_test, y\_test\_pred), 2))  
print("Mean squared error =",  
 round(sm.mean\_squared\_error(y\_test, y\_test\_pred), 2))  
print("Median absolute error =",  
 round(sm.median\_absolute\_error(y\_test, y\_test\_pred), 2))  
print("Explain variance score =",  
 round(sm.explained\_variance\_score(y\_test, y\_test\_pred), 2))  
print("R2 score =", round(sm.r2\_score(y\_test, y\_test\_pred), 2))  
  
output\_model\_file = 'model2.pkl'  
with open(output\_model\_file, 'wb') as f:  
 pickle.dump(regressor, f)  
  
with open(output\_model\_file, 'rb') as f:  
 regressor\_model = pickle.load(f)  
y\_test\_pred\_new = regressor\_model.predict(X\_test)  
print("\nNew mean absolute error =",  
 round(sm.mean\_absolute\_error(y\_test, y\_test\_pred\_new), 2))

Результат виконання програми:



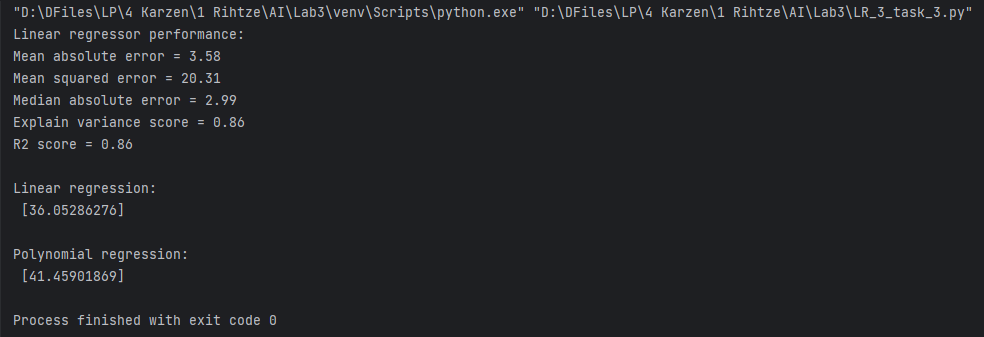


**Завдання 3.** Створення багатовимірного регресора. Використовувати файл вхідних даних: data\_multivar\_regr.txt, побудувати регресійну модель на основі багатьох змінних.

Лістинг програми:

from math import degrees  
import numpy as np  
from sklearn import linear\_model  
import sklearn.metrics as sm  
from sklearn.preprocessing import PolynomialFeatures  
  
input\_file = 'data\_multivar\_regr.txt'  
data = np.loadtxt(input\_file, delimiter=',')  
  
X, y = data[:, :-1], data[:, -1]  
  
num\_training = int(0.8 \* len(X))  
num\_test = len(X) - num\_training  
X\_train, y\_train = X[:num\_training], y[:num\_training]  
X\_test, y\_test = X[num\_training:], y[num\_training:]  
  
linear\_regressor = linear\_model.LinearRegression()  
linear\_regressor.fit(X\_train, y\_train)  
  
y\_test\_pred = linear\_regressor.predict(X\_test)  
  
print("Linear regressor performance:")  
print("Mean absolute error =",  
 round(sm.mean\_absolute\_error(y\_test, y\_test\_pred), 2))  
print("Mean squared error =",  
 round(sm.mean\_squared\_error(y\_test, y\_test\_pred), 2))  
print("Median absolute error =",  
 round(sm.median\_absolute\_error(y\_test, y\_test\_pred), 2))  
print("Explain variance score =",  
 round(sm.explained\_variance\_score(y\_test, y\_test\_pred), 2))  
print("R2 score =", round(sm.r2\_score(y\_test, y\_test\_pred), 2))  
  
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", linear\_regressor.predict(datapoint))  
print("\nPolynomial regression:\n", poly\_linear\_model.predict(poly\_datapoint))

Результат виконання програми:



**Висновок**: У порівнянні із лінійним регресором, поліноміальний peгpecop демонструє кращі результати, в якого значення вище.

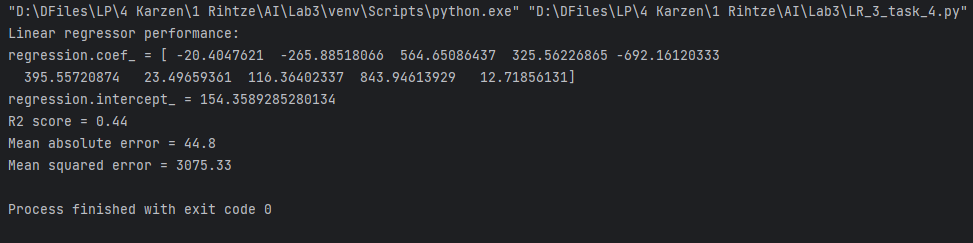
**Завдання 4.** Регресія багатьох змінних

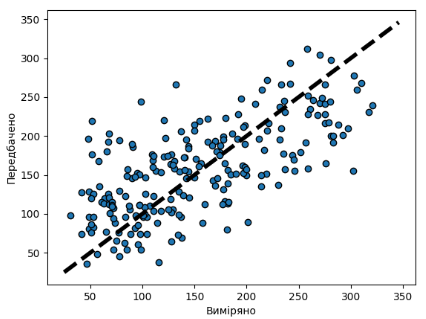
Розробіть лінійний регресор, використовуючи набір даних по діабету, який існує в sklearn.datasets.

Лістинг програми:

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  
from sklearn.model\_selection import train\_test\_split  
  
diabetes = datasets.load\_diabetes()  
X = diabetes.data  
y = diabetes.target  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.5, random\_state=0)  
  
regression = linear\_model.LinearRegression()  
regression.fit(X\_train, y\_train)  
y\_pred = regression.predict(X\_test)  
  
print("Linear regressor performance:")  
print("regression.coef\_ =", regression.coef\_)  
print("regression.intercept\_ =", regression.intercept\_)  
print("R2 score =", round(r2\_score(y\_test, y\_pred), 2))  
print("Mean absolute error =", round(mean\_absolute\_error(y\_test, y\_pred), 2))  
print("Mean squared error =", round(mean\_squared\_error(y\_test, y\_pred), 2))  
  
fig, ax = plt.subplots()  
ax.scatter(y\_test, y\_pred, edgecolors=(0, 0, 0))  
ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=4)  
ax.set\_xlabel('Виміряно')  
ax.set\_ylabel('Передбачено')  
plt.show()

Результат виконання програми:

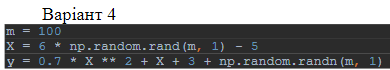




**Завдання 5.** Самостійна побудова регресії

Згенеруйте свої випадкові дані обравши за списком відповідно свій варіант (згідно табл. 2.2) та виведіть їх на графік. Побудуйте по них модель лінійної регресії, виведіть на графік. Побудуйте по них модель поліноміальної регресії, виведіть на графік. Оцініть її якість.

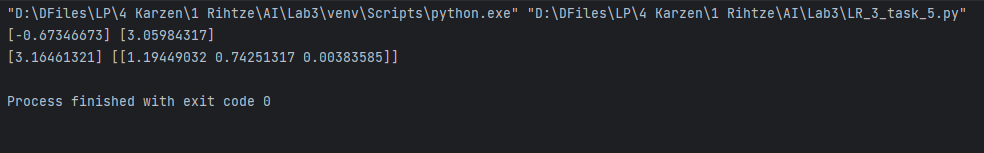


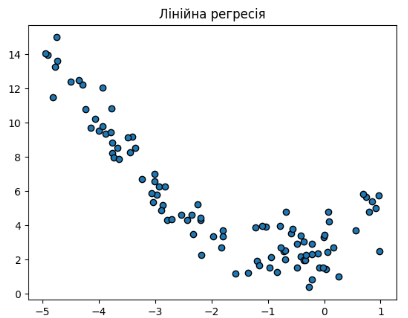


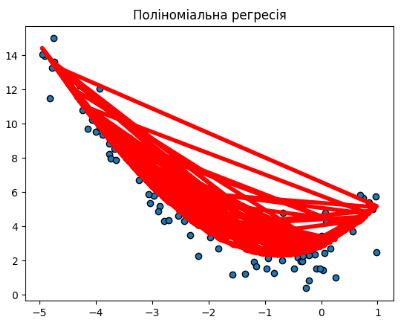
Лістинг програми:

import matplotlib.pyplot as plt  
import numpy as np  
from sklearn import datasets, linear\_model  
from sklearn.preprocessing import PolynomialFeatures  
  
m = 100  
X = 6 \* np.random.rand(m, 1) - 5  
y = 0.7 \* X \*\* 2 + X + 3 + np.random.randn(m, 1)  
  
fig, ax = plt.subplots()  
ax.scatter(X, y, edgecolors=(0, 0, 0))  
plt.title("Лінійна регресія")  
plt.show()  
  
print(X[1], y[1])  
  
poly\_features = PolynomialFeatures(degree=3, include\_bias=False)  
X\_poly = poly\_features.fit\_transform(np.array(X).reshape(-1, 1))  
  
linear\_regression = linear\_model.LinearRegression()  
linear\_regression.fit(X\_poly, y)  
print(linear\_regression.intercept\_, linear\_regression.coef\_)  
y\_pred = linear\_regression.predict(X\_poly)  
  
fig, ax = plt.subplots()  
ax.scatter(X, y, edgecolors=(0, 0, 0))  
plt.plot(X, y\_pred, color='red', linewidth=4)  
plt.title("Поліноміальна регресія")  
plt.show()

Результат виконання програми:







Модель:

Модель регресії з передбаченими коефіцієнтами:

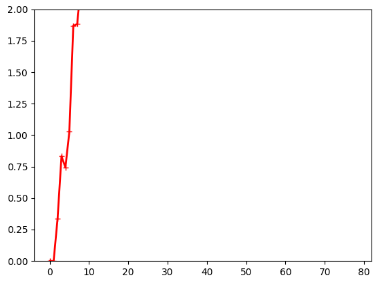
**Завдання 6.** Побудова кривих навчання

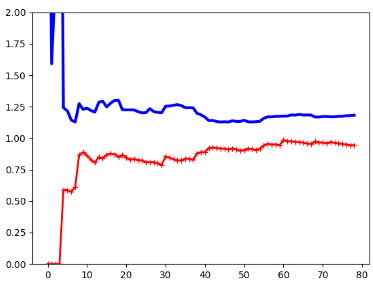
Побудуйте криві навчання для ваших даних у попередньому завданні.

Лістинг програми:

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  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import PolynomialFeatures  
from sklearn.pipeline import Pipeline  
  
m = 100  
X = 6 \* np.random.rand(m, 1) - 5  
y = 0.7 \* X \*\* 2 + X + 3 + np.random.randn(m, 1)  
  
  
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))  
 fig, ax = plt.subplots()  
 plt.ylim(0, 2)  
 ax.plot(np.sqrt(train\_errors), "r-+", linewidth=2, label='train')  
 ax.plot(np.sqrt(val\_errors), "b-", linewidth=3, label='val')  
 plt.show()  
  
  
linear\_regression = linear\_model.LinearRegression()  
plot\_learning\_curves(linear\_regression, np.array(X).reshape(-1, 1), y)  
  
polynomial\_regression = Pipeline([  
 ('poly\_features', PolynomialFeatures(degree=3, include\_bias=False)),  
 ('lin\_reg', linear\_model.LinearRegression()),  
])  
  
plot\_learning\_curves(polynomial\_regression, np.array(X).reshape(-1, 1), y)

Результат виконання програми:





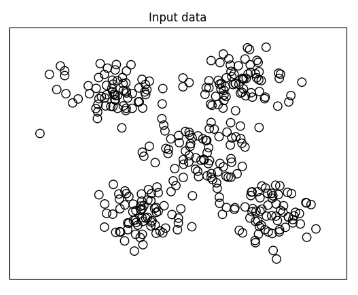
**Завдання 7.** Кластеризація даних за допомогою методу k-середніх.

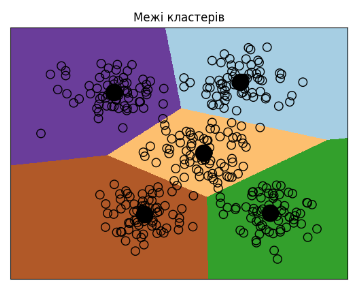
Провести кластеризацію даних методом k-середніх. Використовувати файл вхідних даних: data\_clustering.txt.

Лістинг програми:

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.cluster import KMeans  
  
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('Input data')  
plt.xlim(x\_min, x\_max)  
plt.ylim(y\_min, y\_max)  
plt.xticks(())  
plt.yticks(())  
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\_centers = kmeans.cluster\_centers\_  
plt.scatter(cluster\_centers[:, 0], cluster\_centers[:, 1],  
 marker='o', s=210, linewidths=4, color='black',  
 zorder=12, facecolors='black')  
  
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('Межі кластерів')  
plt.xlim(x\_min, x\_max)  
plt.ylim(y\_min, y\_max)  
plt.xticks(())  
plt.yticks(())  
plt.show()

Результат виконання програми:





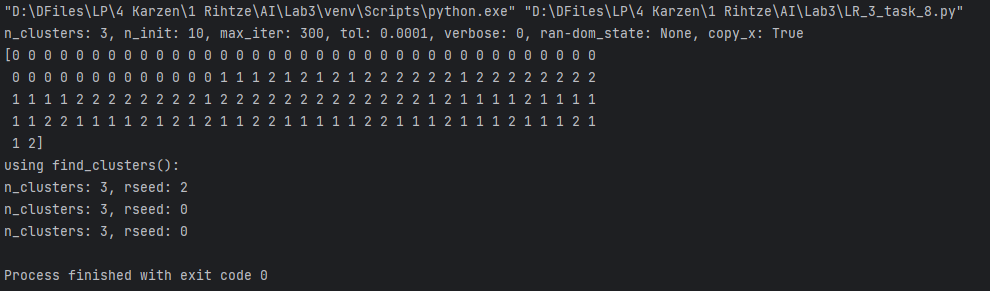
**Висновок:** метод k-середніх валідно працює, але лише за умови відомої кількісті кластерів.

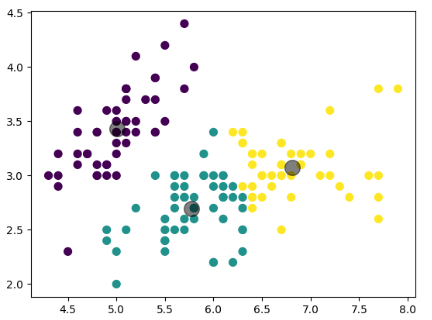
**Завдання 8.** Кластеризація K-середніх для набору даних Iris Виконайте кластеризацію K-середніх для набору даних Iris, який включає три типи (класи) квітів ірису (Setosa, Versicolour і Virginica) чотирма атрибутами: довжина чашолистка, ширина чашолистка, довжина пелюстки та ширина пелюстки. У цьому завданні використовуйте sklearn.cluster.KMeans для пошуку кластерів набору даних Iris.

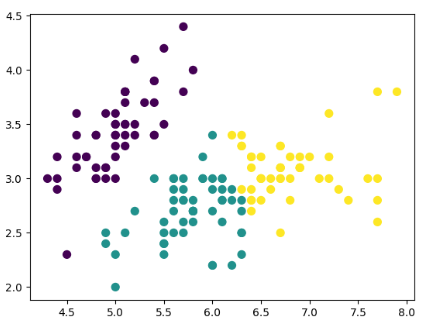
Лістинг програми:

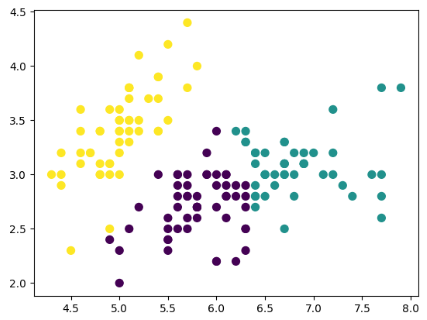
import matplotlib.pyplot as plt  
from sklearn import datasets  
from sklearn.cluster import KMeans  
from sklearn.metrics import pairwise\_distances\_argmin  
import numpy as np  
import warnings  
warnings.filterwarnings("ignore")  
  
iris = datasets.load\_iris()  
X = iris.data[:, :2]  
Y = iris.target  
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, ran-dom\_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()

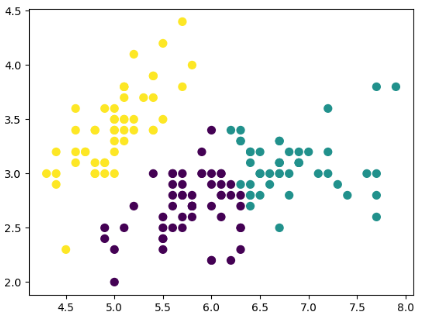
Результат виконання програми:











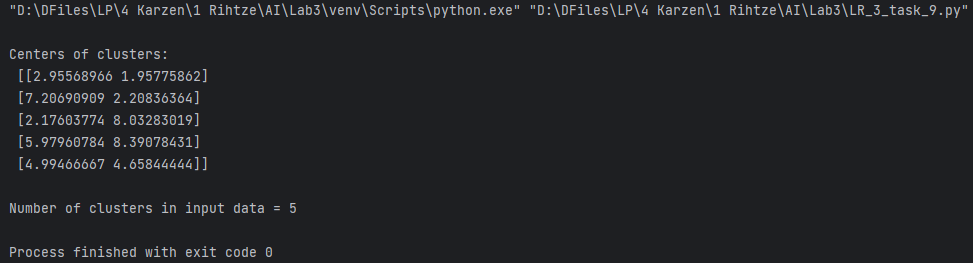
**Завдання 9.** Оцінка кількості кластерів з використанням методу

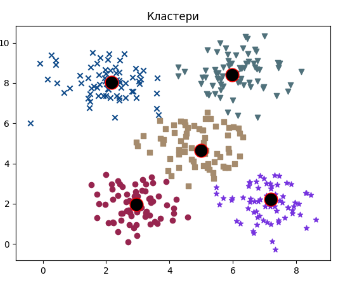
зсуву середнього

Лістинг програми:

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.cluster import MeanShift, estimate\_bandwidth  
from itertools import cycle  
  
# Завантаження даних  
X = np.loadtxt('data\_clustering.txt', delimiter=',')  
  
# Оцінка ширини вікна для X  
bandwidth\_X = estimate\_bandwidth(X, quantile=0.1, n\_samples=len(X))  
  
# Кластеризація даних методом зсуву середнього  
meanshift\_model = MeanShift(bandwidth=bandwidth\_X, bin\_seeding=True)  
meanshift\_model.fit(X)  
  
# Витягування центрів кластерів  
cluster\_centers = meanshift\_model.cluster\_centers\_  
print('\nCenters of clusters:\n', cluster\_centers)  
  
# Оцінка кількості кластерів  
labels = meanshift\_model.labels\_  
num\_clusters = len(np.unique(labels))  
print("\nNumber of clusters in input data =", num\_clusters)  
  
# Відображення на графіку точок та центрів кластерів  
plt.figure()  
markers = 'o\*xvs'  
for i, marker in zip(range(num\_clusters), markers):  
 # Відображення на графіку точок, що належать поточному кластеру  
 plt.scatter(X[labels == i, 0], X[labels == i, 1], marker=marker,  
 color=np.random.rand(3,))  
  
 # Відображення на графіку центру кластера  
 cluster\_center = cluster\_centers[i]  
 plt.plot(cluster\_center[0], cluster\_center[1], marker='o',  
 markerfacecolor='black', markeredgecolor='red',  
 markersize=15)  
  
plt.title('Кластери')  
plt.show()

Результат виконання програми:



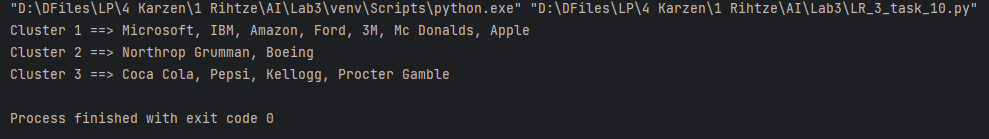


**Завдання 10.** Знаходження підгруп на фондовому ринку з використанням моделі поширення подібності

Лістинг програми:

import datetime  
import json  
import numpy as np  
from sklearn import covariance, cluster  
import yfinance as yf  
  
input\_file = 'company\_symbol\_mapping.json'  
with open(input\_file, 'r') as f:  
 company\_symbols\_map = json.loads(f.read())  
  
symbols, names = np.array(list(company\_symbols\_map.items())).T  
  
start\_date = datetime.datetime(2003, 7, 3)  
end\_date = datetime.datetime(2007, 5, 4)  
  
quotes = []  
for symbol in symbols:  
 quote = yf.Ticker(symbol).history(start=start\_date, end=end\_date)  
 quotes.append(quote)  
  
opening\_quotes = np.array([quote['Open'].values for quote in quotes]).astype(float)  
closing\_quotes = np.array([quote['Close'].values for quote in quotes]).astype(float)  
  
quotes\_diff = closing\_quotes - opening\_quotes  
  
X = quotes\_diff.copy().T  
X /= X.std(axis=0)  
  
edge\_model = covariance.GraphicalLassoCV()  
  
with np.errstate(invalid='ignore'):  
 edge\_model.fit(X)  
  
\_, labels = cluster.affinity\_propagation(edge\_model.covariance\_)  
num\_labels = labels.max()  
  
for i in range(num\_labels + 1):  
 cluster\_names = names[labels == i]  
 if len(cluster\_names) > 0:  
 print("Cluster", i+1, "==>", ', '.join(cluster\_names))

Результат виконання програми:



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