**ЛАБОРАТОРНА РОБОТА № 3**

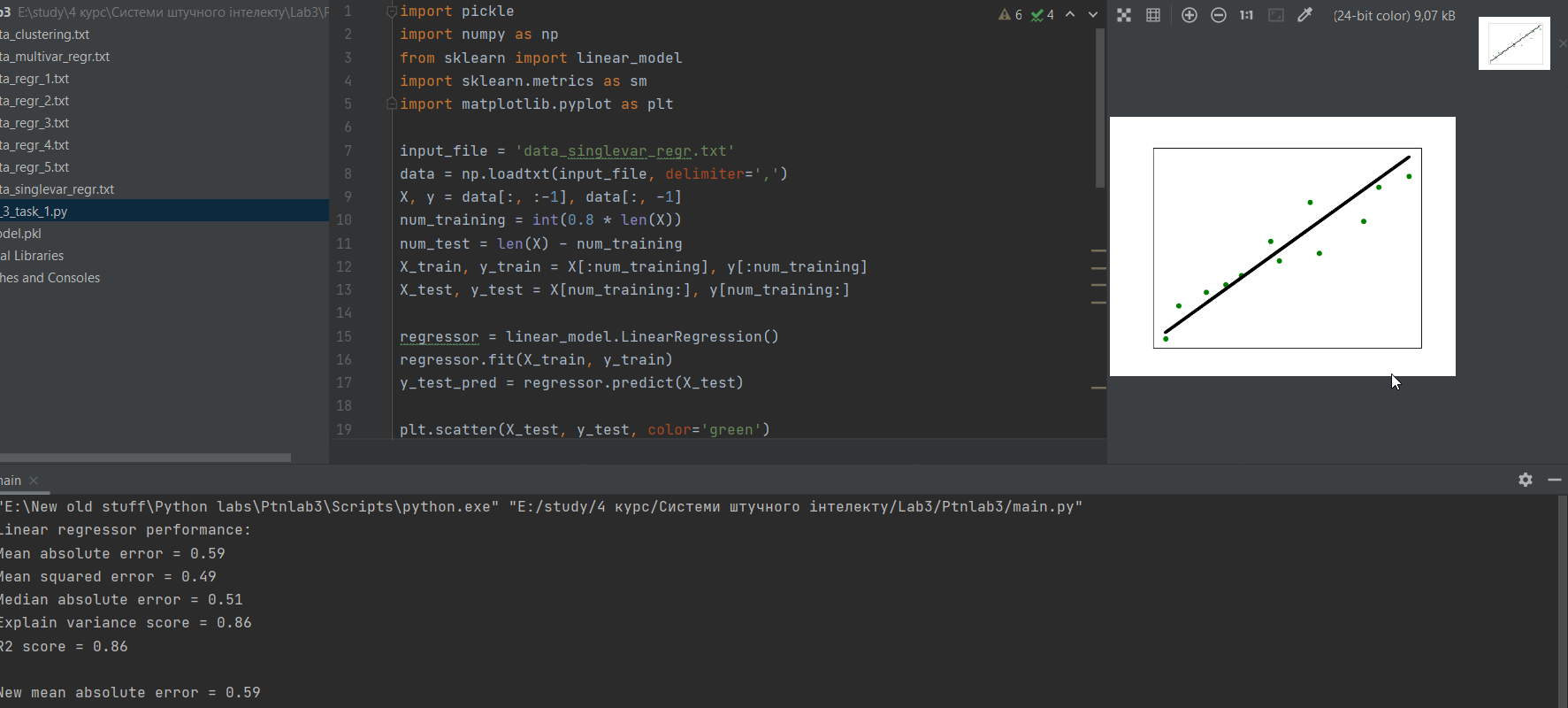
**Тема**: ДОСЛІДЖЕННЯ МЕТОДІВ РЕГРЕСІЇ ТА НЕКОНТРОЬОВАНОГО НАВЧАННЯ.

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

**Хід роботи:**

**Task1**

****

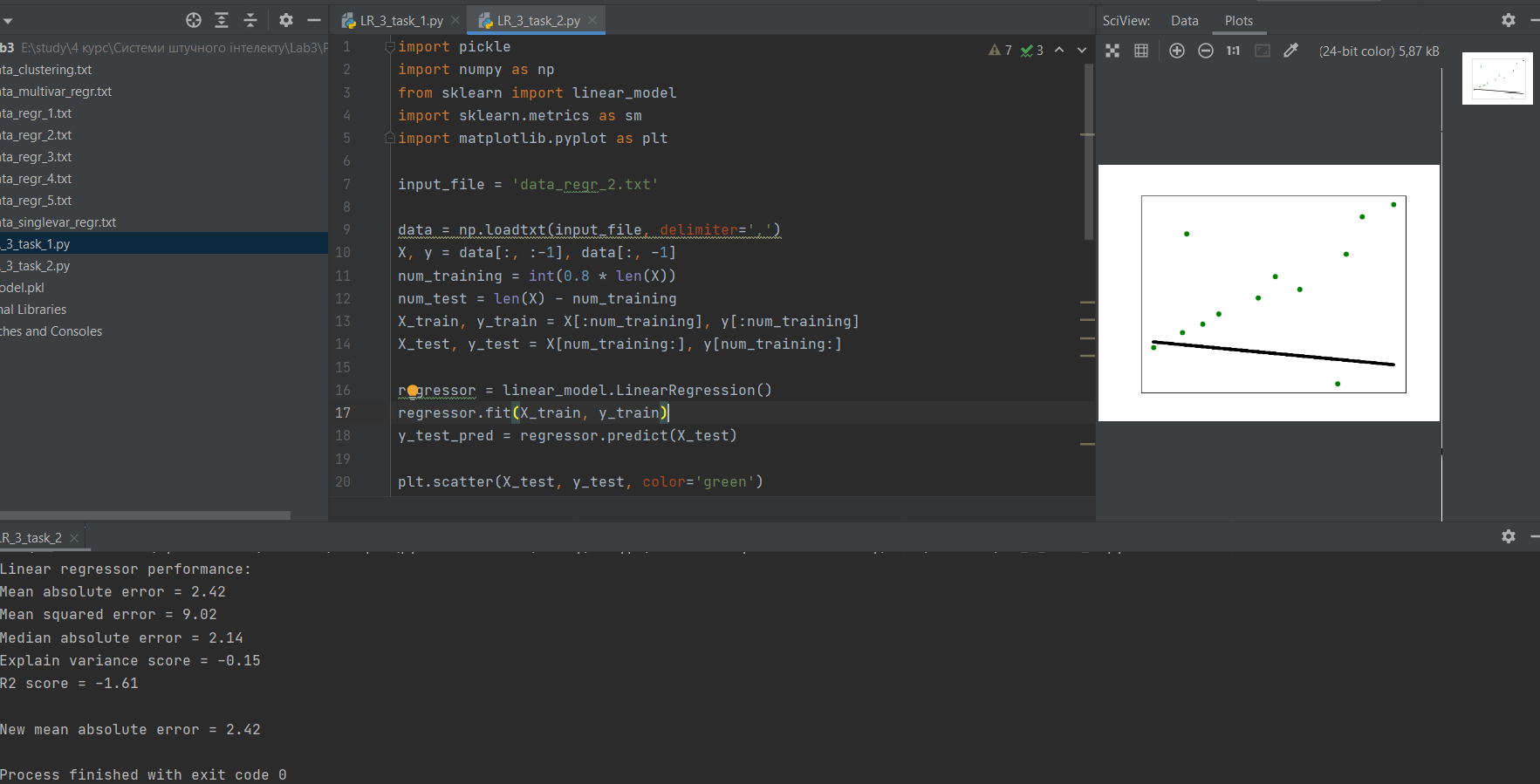
****

Використання лінійної регресії є простим, але неефективним через узагальненість та неточність.

import pickle  
import numpy as np  
from sklearn import linear\_model  
import sklearn.metrics as sm  
import matplotlib.pyplot as plt  
  
input\_file = 'data\_singlevar\_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:]  
  
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 = 'model.pkl'  
  
with open(output\_model\_file, 'wb') as f:  
 pickle.dump(regressor, f)  
  
y\_test\_pred\_new = regressor.predict(X\_test)  
print("\nNew mean absolute error =",  
round(sm.mean\_absolute\_error(y\_test, y\_test\_pred\_new), 2))

**Task2**

****

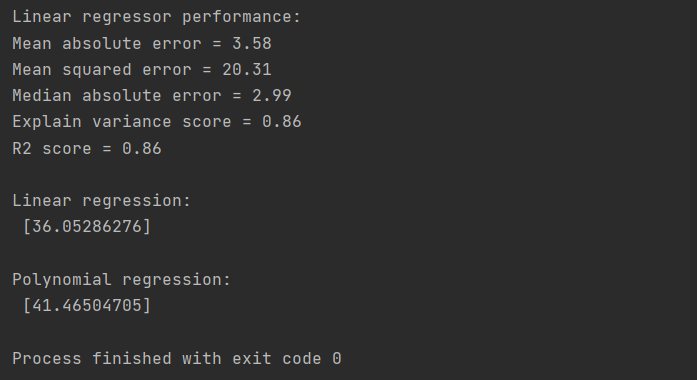
****

За малої кількості даних використання будь-яких алгоритмів є неефективним, особливо алгоритму лінійної регресії.

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\_2.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 = 'model.pkl'  
  
with open(output\_model\_file, 'wb') as f:  
 pickle.dump(regressor, f)  
  
y\_test\_pred\_new = regressor.predict(X\_test)  
print("\nNew mean absolute error =",  
round(sm.mean\_absolute\_error(y\_test, y\_test\_pred\_new), 2))

**Task3**

****

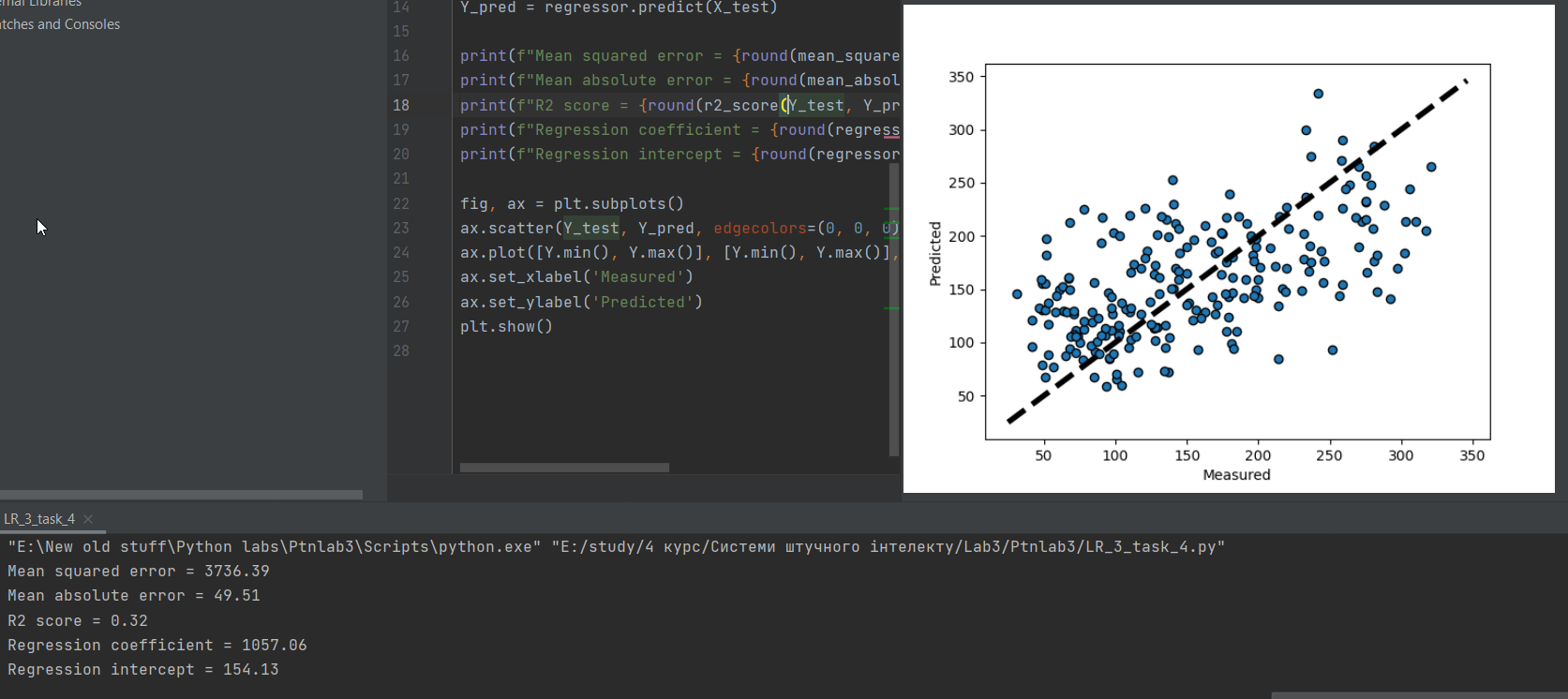
****

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:]  
  
regressor = linear\_model.LinearRegression()  
regressor.fit(X\_train, y\_train)  
  
y\_test\_pred = 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", regressor.predict(datapoint))  
print("\nPolynomial regression:\n", poly\_linear\_model.predict(poly\_datapoint))

Порівняно з лінійним регресором поліноміальний peгpecop забезпечує отримання результату, ближчого до значення 41.35. Тобто дає кращі результати

**Task4**



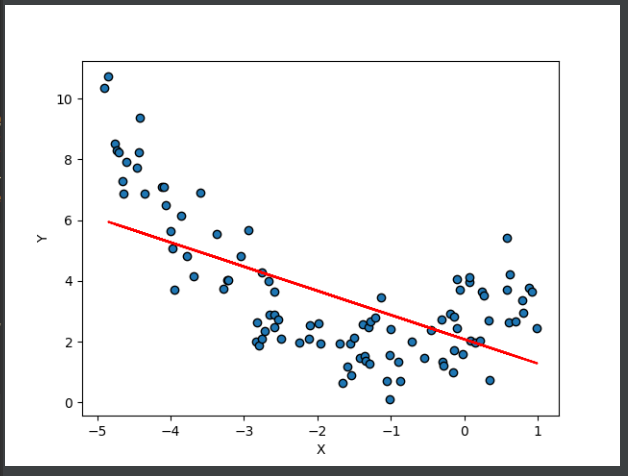
****

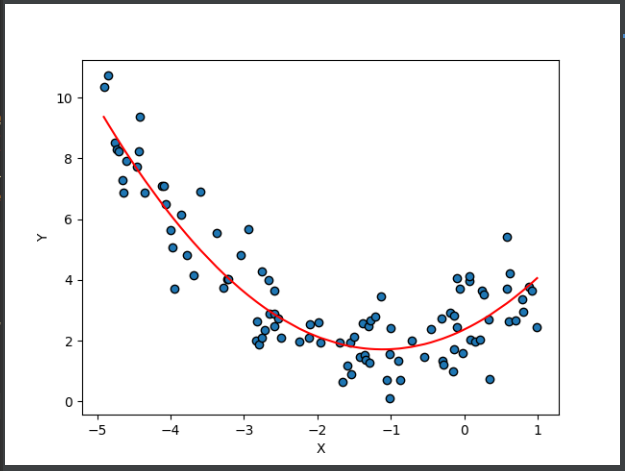
import matplotlib.pyplot as plt  
import numpy as np  
from sklearn import linear\_model, datasets  
from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score  
from sklearn.model\_selection import train\_test\_split  
  
diabetes = datasets.load\_diabetes()  
X = diabetes.data[:, np.newaxis, 2]  
Y = diabetes.target  
  
X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.5, random\_state=0)  
regressor = linear\_model.LinearRegression()  
regressor.fit(X\_train, Y\_train)  
Y\_pred = regressor.predict(X\_test)  
  
print(f"Mean squared error = {round(mean\_squared\_error(Y\_test, Y\_pred), 2)}")  
print(f"Mean absolute error = {round(mean\_absolute\_error(Y\_test, Y\_pred), 2)}")  
print(f"R2 score = {round(r2\_score(Y\_test, Y\_pred), 2)}")  
print(f"Regression coefficient = {round(regressor.coef\_[0], 2)}")  
print(f"Regression intercept = {round(regressor.intercept\_, 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('Measured')  
ax.set\_ylabel('Predicted')  
plt.show()

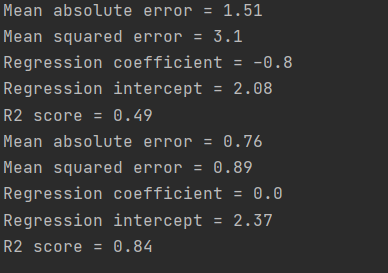
Використання лінійної регресії в даному випадку не є ефективним через велике розповсюдження даних.

**Task5**

****

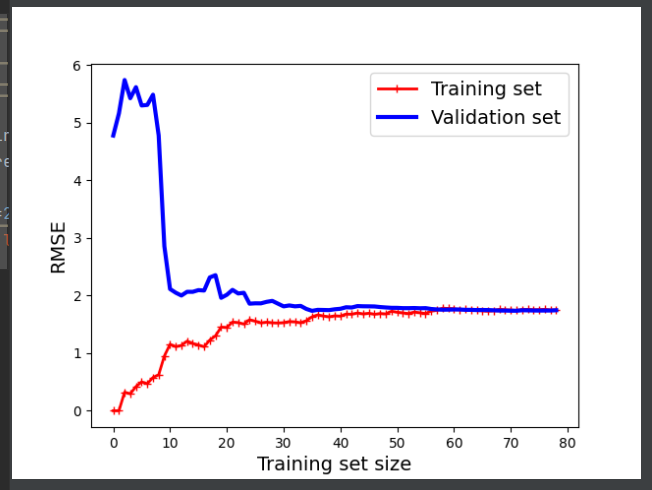
****

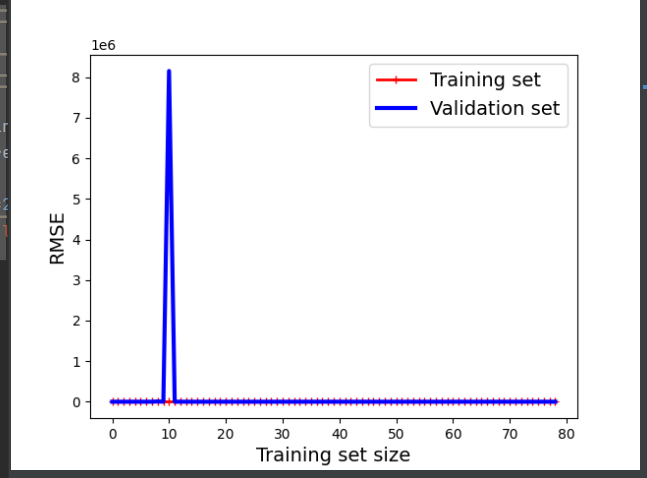
****

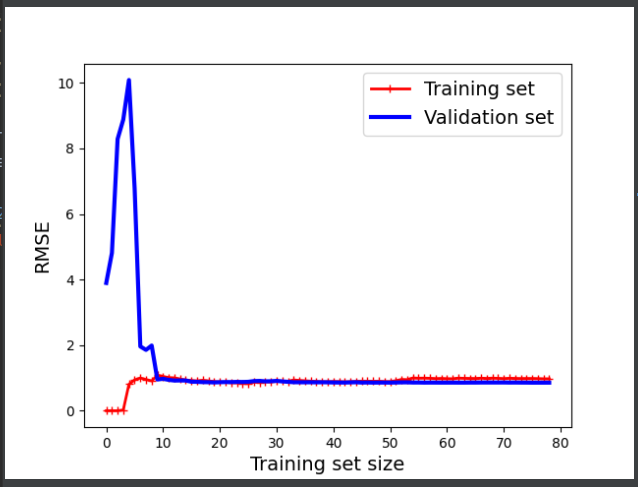
****

import matplotlib.pyplot as plt  
import numpy as np  
from sklearn import linear\_model  
from sklearn.preprocessing import PolynomialFeatures  
from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score  
from sklearn.model\_selection import train\_test\_split  
  
m = 100  
X = 6 \* np.random.rand(m, 1) - 5  
Y = 0.5 \* X\*\*2 + X + 2 + np.random.randn(m, 1)  
  
indices = np.argsort(X, axis=0)  
X = X[indices].reshape(-1, 1)  
Y = Y[indices].reshape(-1, 1)  
  
X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.5, random\_state=0)  
regressor = linear\_model.LinearRegression()  
regressor.fit(X\_train, Y\_train)  
Y\_pred = regressor.predict(X\_test)  
  
print(f"Mean absolute error = {round(mean\_absolute\_error(Y\_test, Y\_pred), 2)}")  
print(f"Mean squared error = {round(mean\_squared\_error(Y\_test, Y\_pred), 2)}")  
print(f"Regression coefficient = {round(regressor.coef\_[0][0], 2)}")  
print(f"Regression intercept = {round(regressor.intercept\_[0], 2)}")  
print(f"R2 score = {round(r2\_score(Y\_test, Y\_pred), 2)}")  
  
plt.scatter(X, Y, edgecolors=(0, 0, 0))  
plt.plot(X\_test, Y\_pred, color="red")  
plt.xlabel('X')  
plt.ylabel('Y')  
plt.show()  
  
poly = PolynomialFeatures(degree=2)  
X\_poly = poly.fit\_transform(X)  
regressor = linear\_model.LinearRegression()  
regressor.fit(X\_poly, Y)  
Y\_pred = regressor.predict(X\_poly)  
  
print(f"Mean absolute error = {round(mean\_absolute\_error(Y, Y\_pred), 2)}")  
print(f"Mean squared error = {round(mean\_squared\_error(Y, Y\_pred), 2)}")  
print(f"Regression coefficient = {round(regressor.coef\_[0][0], 2)}")  
print(f"Regression intercept = {round(regressor.intercept\_[0], 2)}")  
print(f"R2 score = {round(r2\_score(Y, Y\_pred), 2)}")  
  
plt.scatter(X, Y, edgecolors=(0, 0, 0))  
plt.plot(X, Y\_pred, color="red")  
plt.xlabel('X')  
plt.ylabel('Y')  
plt.show()

**Task6**

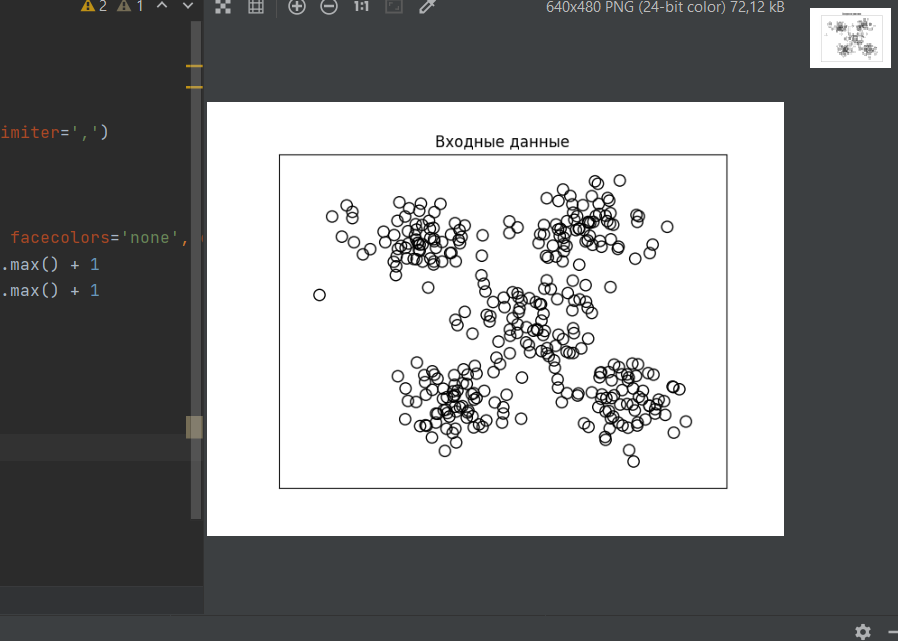
****

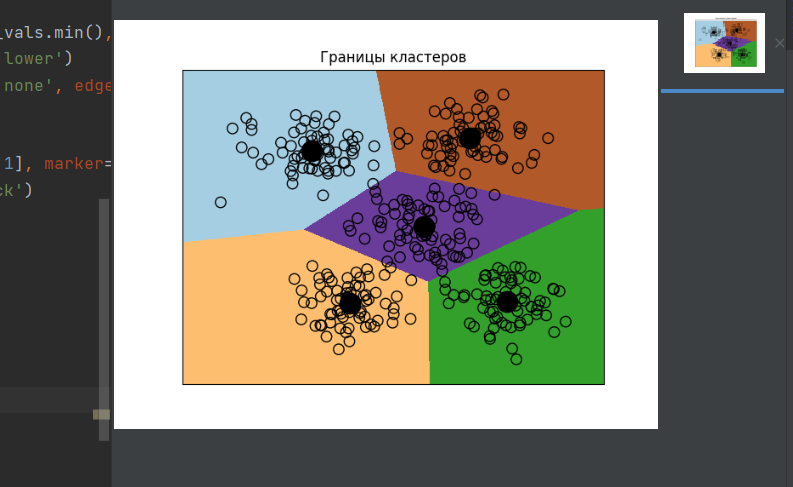
****

****

**Task7**

****

****

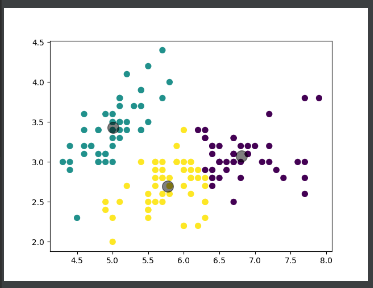
****

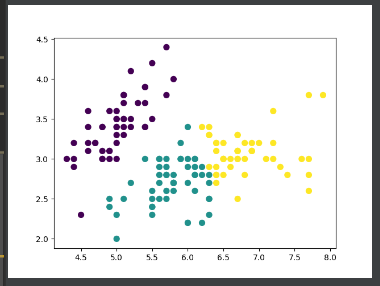
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.cluster import KMeans  
from sklearn import metrics  
  
X = np.loadtxt('data\_clustering.txt', delimiter=',')  
num\_clusters = 5  
  
plt.figure()  
plt.scatter(X[:, 0], X[:, 1], marker='o', facecolors='none', edgecolors='black', s=80)  
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('Входные данные')  
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\_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('Границы кластеров')  
plt.xlim(x\_min, x\_max)  
plt.ylim(y\_min, y\_max)  
plt.xticks(())  
plt.yticks(())  
plt.show()

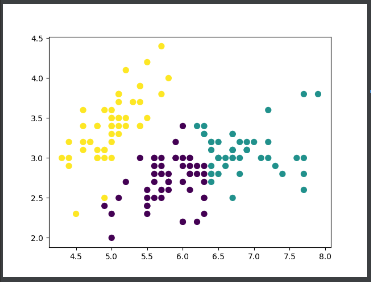
Використання методу K-середніх дозволяє ефективно класифікувати дані без допомоги вчителя, а за використання K-середніх++ знаходження центрів залишається за алгоритмом.

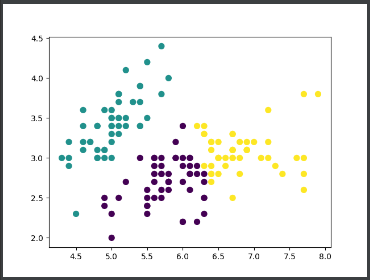
**Task8**

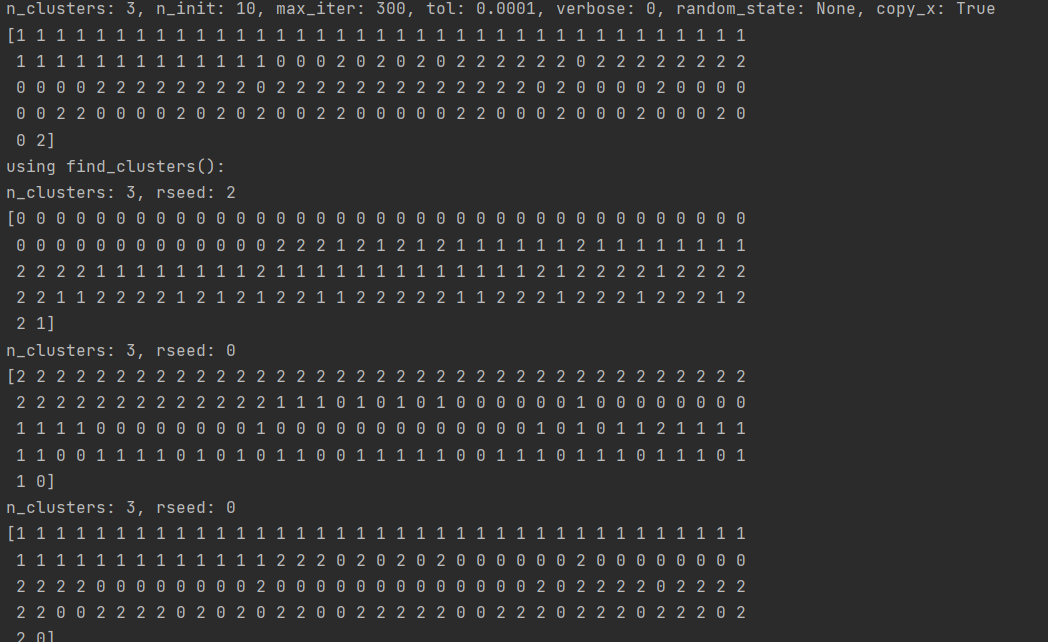
****

****

****

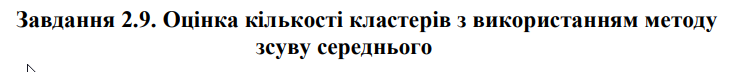
****

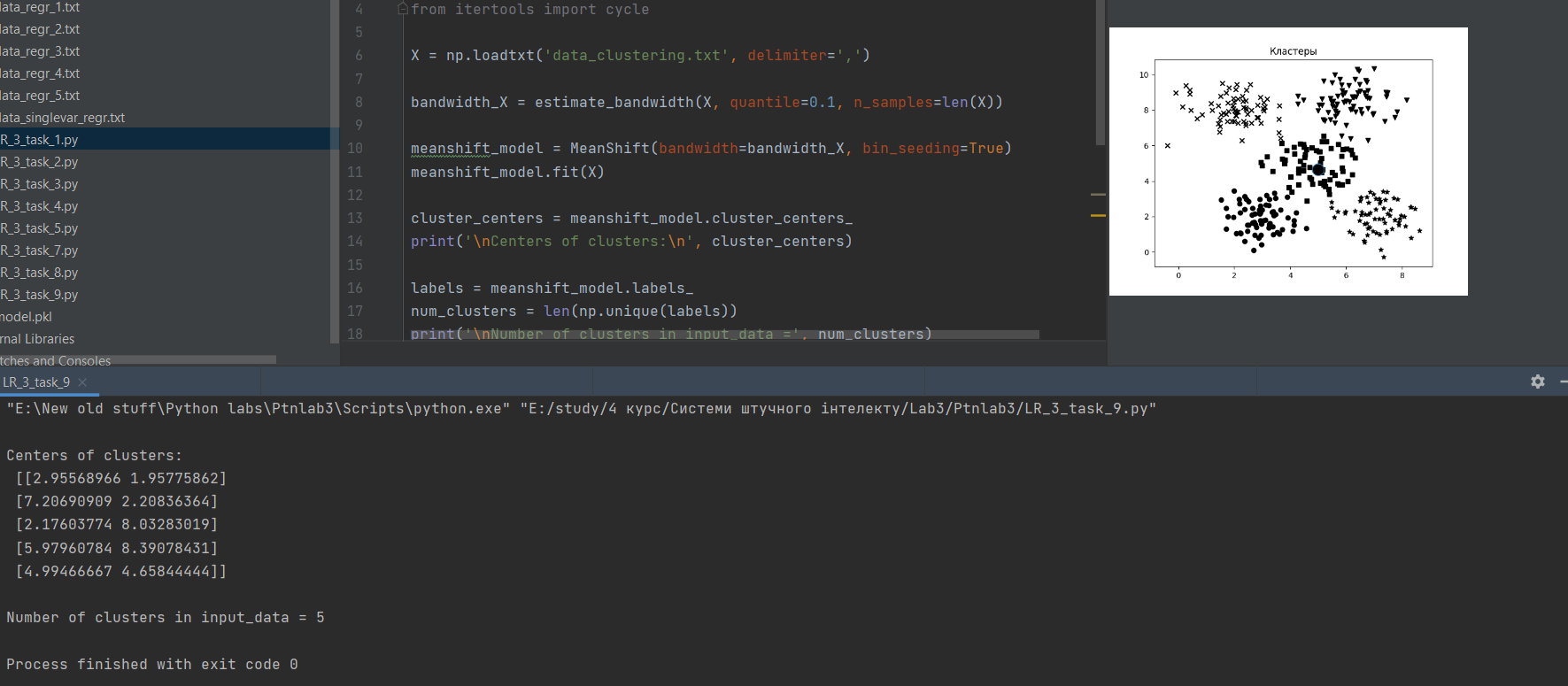
****

****

import matplotlib.pyplot as plt  
from sklearn import datasets  
from sklearn.svm import SVC  
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(f"n\_clusters: {Y.max() + 1}, 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(f"n\_clusters: {3}, rseed: {2}")  
print(labels)  
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')  
plt.show()  
  
centers, labels = find\_clusters(X, 3, rseed=0)  
print(f"n\_clusters: {3}, rseed: {0}")  
print(labels)  
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')  
plt.show()  
  
labels = KMeans(3, random\_state=0).fit\_predict(X)  
print(f"n\_clusters: {3}, rseed: {0}")  
print(labels)  
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')  
plt.show()

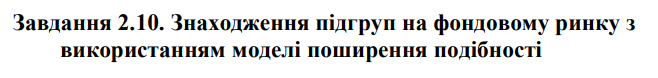
**Task9**

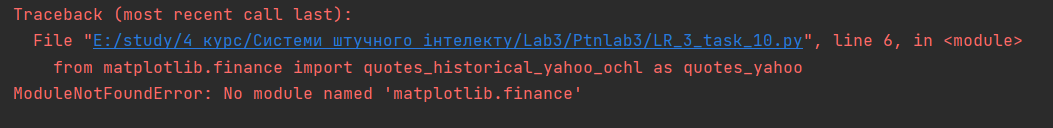
****

****

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\_X = estimate\_bandwidth(X, quantile=0.1, n\_samples=len(X))  
  
meanshift\_model = MeanShift(bandwidth=bandwidth\_X, bin\_seeding=True)  
meanshift\_model.fit(X)  
  
cluster\_centers = meanshift\_model.cluster\_centers\_  
print('\nCenters of clusters:\n', cluster\_centers)  
  
labels = meanshift\_model.labels\_  
num\_clusters = len(np.unique(labels))  
print('\nNumber of clusters in input\_data =', num\_clusters)  
  
plt.figure()  
markers = 'o\*xvs'  
for i, marker in zip(range(num\_clusters), markers):  
 plt.scatter(X[labels==i, 0], X[labels==i, 1], marker=marker, color='black')  
  
cluster\_center = cluster\_centers[i]  
plt.plot(cluster\_center[0], cluster\_center[1], marker='o',  
 markerfacecolor='black', markersize=15)  
plt.title('Кластеры')  
plt.show()

**Task10**

****

****

import datetime  
import json  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn import covariance, cluster  
from matplotlib.finance import quotes\_historical\_yahoo\_ochl as quotes\_yahoo  
  
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 = [quotes\_yahoo(symbol, start\_date, end\_date, asobject=True) for symbol in symbols]  
  
opening\_quotes = np.array([quote.open for quote in quotes]).astype(np.float)  
closing\_quotes = np.array([quote.close for quote in quotes]).astype(np.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):  
 print(“Cluster”, i+1, “🡺”, ‘,’.join(names[labels == i]))

**Під час виконання коду сталася помилка. Модуля matplotlib.finance не було знайдено.**

**https://gitlab.com/2019-2023/ipz19-3/lysovyi-maksym/AI**

**Висновок:** використовуючи спеціалізовані бібліотеки і мову програмування Python дослідив методи регресії та неконтрольованої класифікації даних у машинному навчанні.