**ЛАБОРАТОРНА РОБОТА № 3**

**ДОСЛІДЖЕННЯ МЕТОДІВ РЕГРЕСІЇ ТА НЕКОНТРОЬОВАНОГО НАВЧАННЯ**

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

**ЧАСТИНА 1. ДОСЛІДЖЕННЯ МЕТОДІВ РЕГРЕСІЇ**

**1. ТЕОРЕТИЧНІ ВІДОМОСТІ**

Теоретичні відомості подані на лекціях. Також доцільно вивчити матеріал поданий в літературі:

Джоши Пратик. Искусственный интеллект с примерами на Python. : Пер. с англ. - СПб. : ООО "Диалектика", 2019. - 448 с. - Парал. тит. англ. ISBN 978-5-907114-41-8 (рус.)

Можна використовувати Google Colab або Jupiter Notebook.

Регресія - це процес оцінки того, як співвідносяться між собою вхідні та вихідні змінні. Слід зазначити, що вихідні змінні можуть мати значення з безперервного ряду дійсних чисел. Отже, існує безліч результуючих можливостей. Це різко контрастує з процесом класифікації, у якому кількість вихідних класів фіксовано.

У регресії передбачається, що вихідні змінні залежить від вхідних, і завдання полягає у з'ясуванні співвідношення між ними. Звідси вхідні змінні називаються незалежними змінними (або предикторами), а вихідні – залежними (або критеріальними змінними). При цьому не потрібно, щоб вхідні змінні були незалежними один від одного. Існує безліч ситуацій, коли між вхідними змінними існує кореляція.

Регресійний аналіз дозволяє з'ясувати, як змінюється значення вихідний змінної, коли змінюємо лише частина вхідних змінних, залишаючи інші вхідні змінні фіксованими. У разі лінійної регресії передбачається, що вхідні та вихідні змінні пов'язані між собою лінійною залежністю. Це накладає обмеження на нашу процедуру моделювання, але прискорює її та робить більш ефективною.

Іноді лінійної регресії виявляється недостатньо для пояснення співвідношень між вхідними та вихідними змінними. У подібних випадках ми використовуємо поліноміальну регресію, в якій вхідні та вихідні змінні пов'язані між собою поліноміальною залежністю.

З обчислювальної точки зору такий підхід складніший, але забезпечує більш високу точність. Вибір виду регресії для виявлення зазначених відношень визначається видом конкретної задачі. Регресію часто використовують для прогнозування цін, економічних показників та інше.

**2. ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ ТА МЕТОДИЧНІ**

**РЕКОМЕНДАЦІЇ ДО ЙОГО ВИКОНАННЯ**

**Завдання 2.1. Створення регресора однієї змінної**

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("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'

# Збереження моделі

with open(output\_model\_file, 'wb') as f:

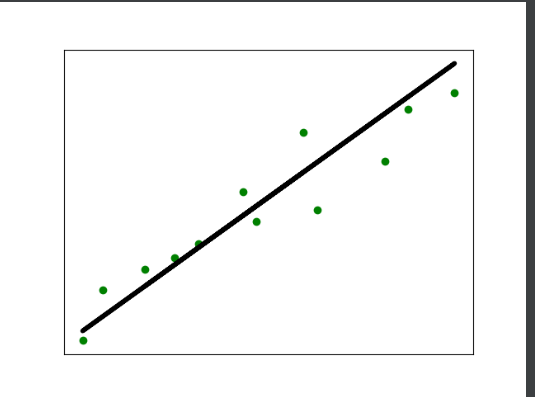
pickle.dump(regressor, f)

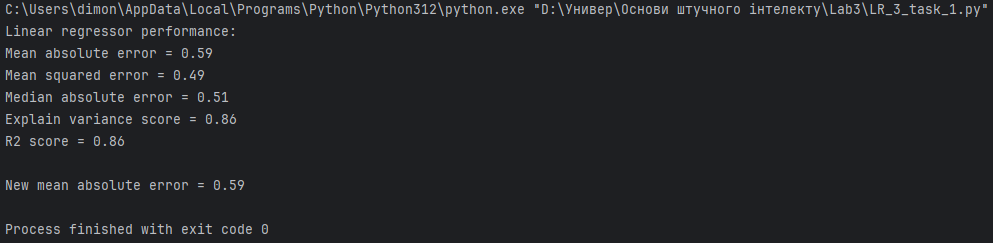
# Завантаження моделі

y\_test\_pred\_new = regressor.predict(X\_test)

print("\nNew mean absolute error =",

round(sm.mean\_absolute\_error(y\_test, y\_test\_pred\_new), 2))





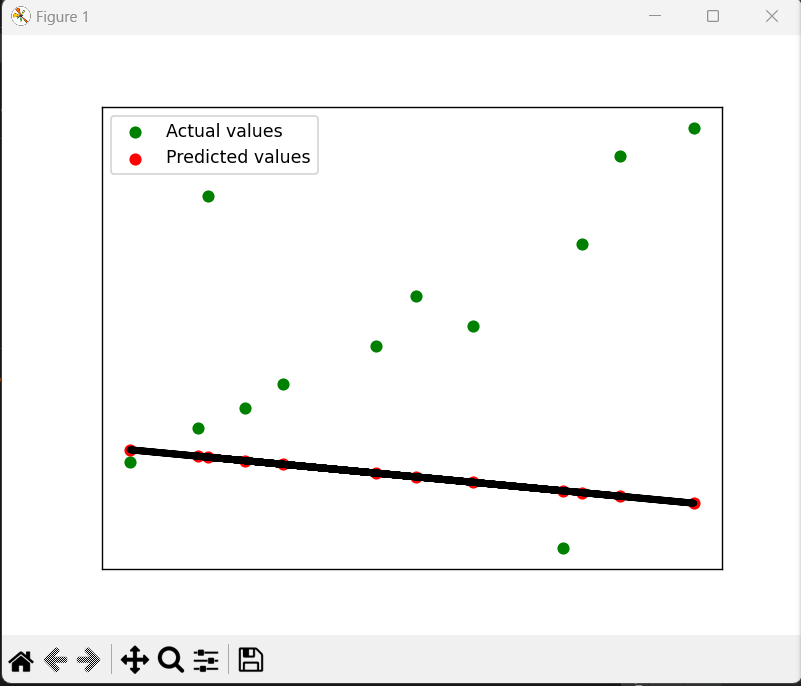
Висновок: Модель виявляє високу точність у прогнозуванні, оскільки значення середньоквадратичної помилки (MSE) дуже низьке, вказуючи на добру адаптацію до спостережуваних даних. Здатність моделі відтворювати спостережувані дані також є високою. Крім того, нова середня абсолютна помилка також дорівнює 0.59, що свідчить про стабільну точність моделі при прогнозуванні нових даних.

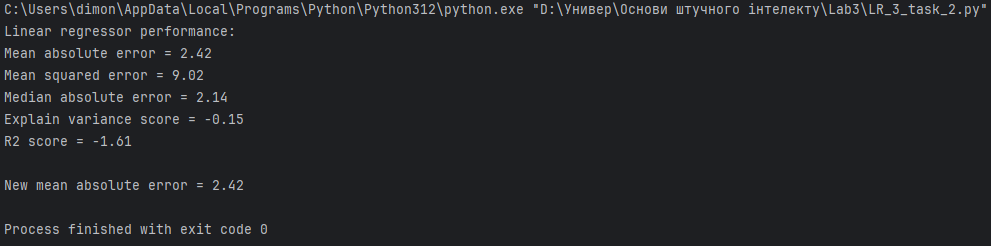
**Завдання 2.2. Передбачення за допомогою регресії однієї змінної**

|  |  |
| --- | --- |
| № за списком | 2 |
| № варіанту | 2 |

import pickle  
import matplotlib  
matplotlib.use('TkAgg') # або інший бекенд, який працює на вашій системі  
import matplotlib.pyplot as plt  
import numpy as np  
from sklearn import linear\_model  
import sklearn.metrics as sm  
  
  
# Вхідний файл, який містить дані  
input\_file = 'data\_regr\_2.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 на тестових даних  
y\_test\_pred = regressor.predict(X\_test)  
  
# Побудова графіка  
plt.scatter(X\_test, y\_test, color='green', label='Actual values')  
plt.scatter(X\_test, y\_test\_pred, color='red', label='Predicted values')  
plt.plot(X\_test, y\_test\_pred, color='black', linewidth=4)  
plt.legend()  
plt.xticks(())  
plt.yticks(())  
plt.show()  
  
# Роздрукування результатів  
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'  
  
# 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("\nNew mean absolute error =", round(sm.mean\_absolute\_error(y\_test, y\_test\_pred\_new), 2))





Висновок: У даному випадку можна визначити, що регресія є невдалим вибором для нашої задачі. Вона проявляє значні помилки у прогнозах та не здатна належним чином пояснити змінність у ваших даних. Модель не ефективно виконує свої функції і потребує подальшої оптимізації або розгляду інших типів моделей.

**Завдання 2.3. Створення багатовимірного регресора**

import numpy as np

from sklearn import linear\_model

import sklearn.metrics as sm

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("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))

# Поліноміальна регресія

polynomial = PolynomialFeatures(degree=10)

X\_train\_transformed = polynomial.fit\_transform(X\_train)

datapoint = [[7.75, 6.35, 5.56]]

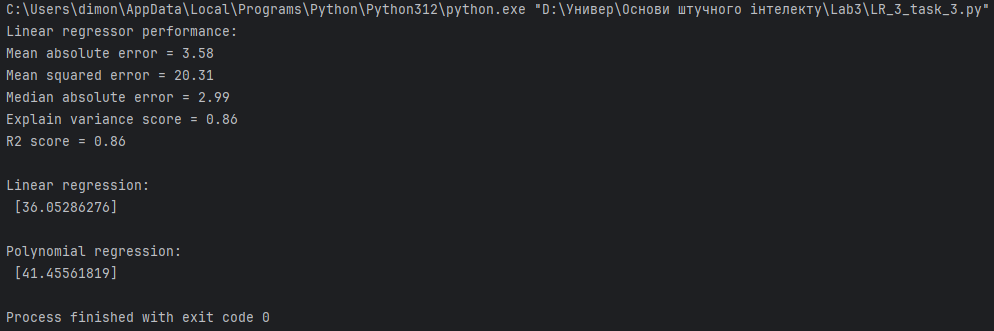
poly\_datapoint = polynomial.fit\_transform(datapoint)

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))



**Завдання 2.4. Регресія багатьох змінних**

import matplotlib.pyplot as plt

import numpy as np

from sklearn import datasets, linear\_model

from sklearn.metrics import mean\_squared\_error, r2\_score

from sklearn.metrics import mean\_absolute\_error

from sklearn.model\_selection import train\_test\_split

diabetes = datasets.load\_diabetes()

X = diabetes.data

y = diabetes.target

Xtrain, Xtest, ytrain, ytest = train\_test\_split(X, y, test\_size=0.5, random\_state=0)

regr = linear\_model.LinearRegression()

regr.fit(Xtrain, ytrain)

ypred = regr.predict(Xtest)

print("regr coef =", np.around(regr.coef\_,2))

print("regr intercept =", np.around(regr.intercept\_,2))

print("R2 score =", np.around(r2\_score(ytest,ypred), 2))

print("Mean absolute error =", np.around(mean\_absolute\_error(ytest,ypred), 2))

print("Mean squared error =", np.around(mean\_squared\_error(ytest,ypred), 2))

fig, ax = plt.subplots()

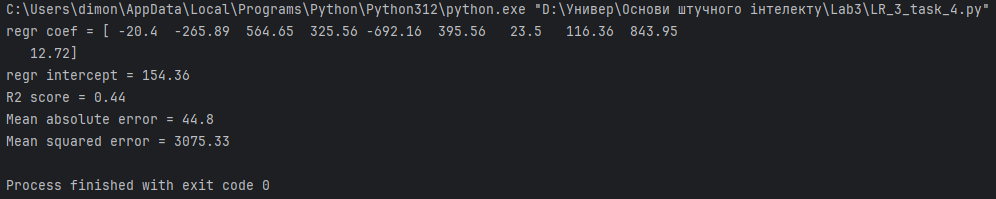
ax.scatter(ytest, ypred, edgecolors = (0, 0, 0))

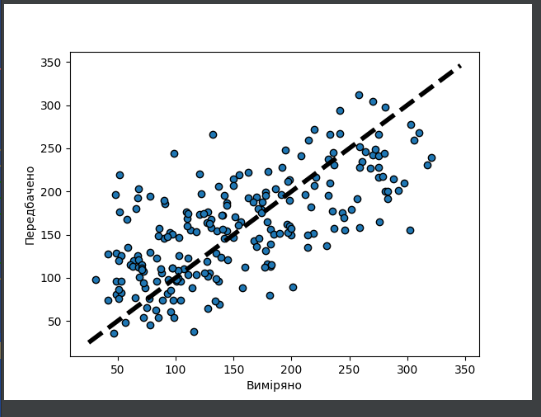
ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw = 4)

ax.set\_xlabel('Виміряно')

ax.set\_ylabel('Передбачено')

plt.show()





Висновок: За отриманими результатами можна зробити висновок, що модель обмежено в здатності пояснювати варіацію цільової змінної, а її прогнози характеризуються середньою похибкою приблизно 44.8 одиниці. Можливо, для покращення якості прогнозу слід розглянути використання інших факторів або вибір інших моделей.

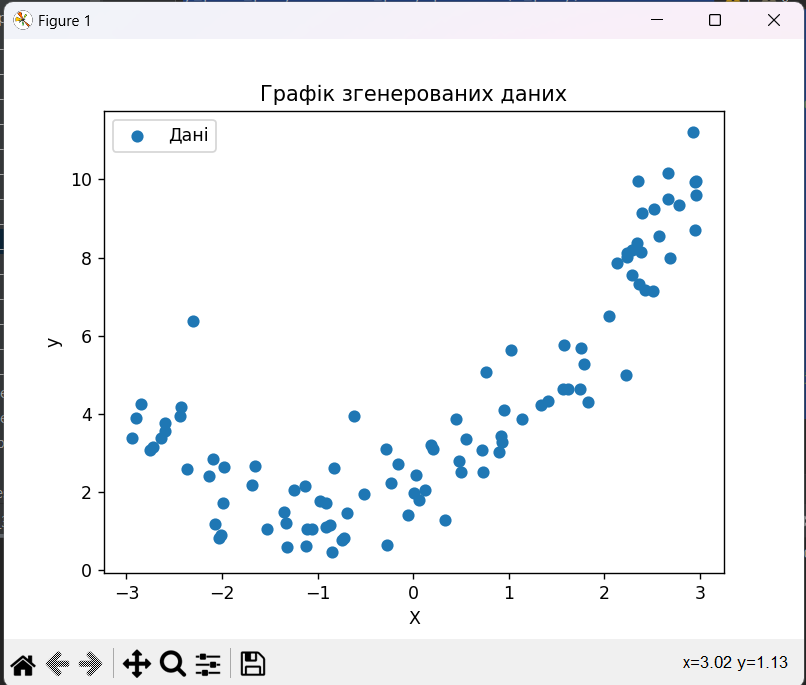
**Завдання 2.5. Самостійна побудова регресії**

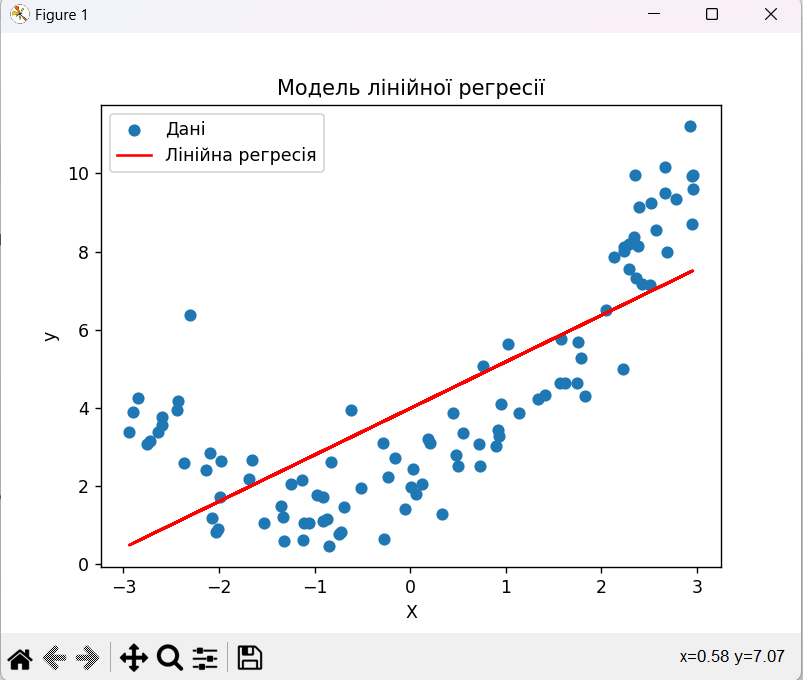
|  |  |
| --- | --- |
| № за списком | 2 |
| № варіанту | 2 |

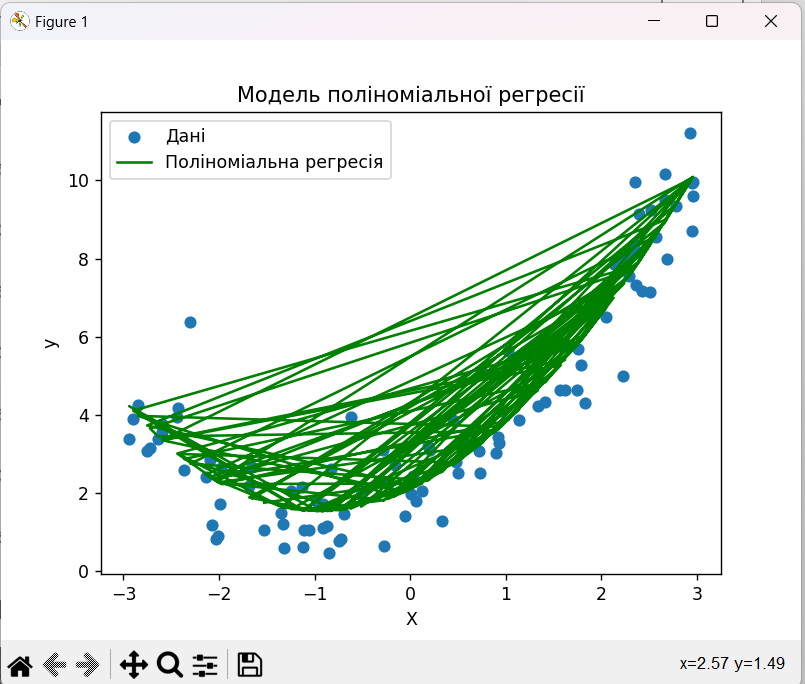
Варіант 2

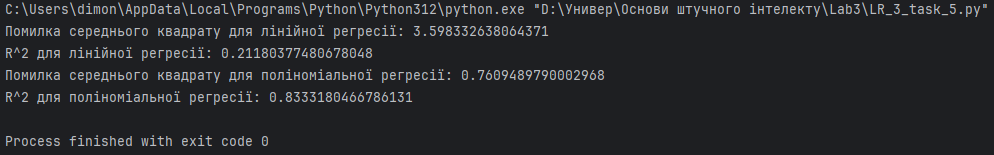
m = 100 X = np.linspace(-3, 3, m) y = 3 + np.sin(X) + np.random.uniform(-0.5, 0.5, m)

import matplotlib  
matplotlib.use('TkAgg') # Змініть 'TkAgg' на інший бекенд за потреби  
import matplotlib.pyplot as plt  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.linear\_model import LinearRegression  
from sklearn.preprocessing import PolynomialFeatures  
from sklearn.metrics import mean\_squared\_error, r2\_score  
  
# Згенеруємо дані  
m = 100  
X = 6 \* np.random.rand(m, 1) - 3  
y = 0.6 \* X \*\* 2 + X + 2 + np.random.randn(m, 1)  
  
# Побудуємо графік  
plt.scatter(X, y, label='Дані')  
plt.xlabel('X')  
plt.ylabel('y')  
plt.legend()  
plt.title('Графік згенерованих даних')  
plt.show()  
  
# Побудуємо модель лінійної регресії  
X = X.reshape(-1, 1)  
model\_linear = LinearRegression()  
model\_linear.fit(X, y)  
y\_pred\_linear = model\_linear.predict(X)  
  
# Побудуємо графік моделі лінійної регресії  
plt.scatter(X, y, label='Дані')  
plt.plot(X, y\_pred\_linear, color='red', label='Лінійна регресія')  
plt.xlabel('X')  
plt.ylabel('y')  
plt.legend()  
plt.title('Модель лінійної регресії')  
plt.show()  
  
# Оцінимо якість моделі лінійної регресії  
mse\_linear = mean\_squared\_error(y, y\_pred\_linear)  
r2\_linear = r2\_score(y, y\_pred\_linear)  
print(f'Помилка середнього квадрату для лінійної регресії: {mse\_linear}')  
print(f'R^2 для лінійної регресії: {r2\_linear}')  
  
# Побудуємо модель поліноміальної регресії (наприклад, поліном 3-го ступеня)  
poly = PolynomialFeatures(degree=3)  
X\_poly = poly.fit\_transform(X)  
model\_poly = LinearRegression()  
model\_poly.fit(X\_poly, y)  
y\_pred\_poly = model\_poly.predict(X\_poly)  
  
# Побудуємо графік моделі поліноміальної регресії  
plt.scatter(X, y, label='Дані')  
plt.plot(X, y\_pred\_poly, color='green', label='Поліноміальна регресія')  
plt.xlabel('X')  
plt.ylabel('y')  
plt.legend()  
plt.title('Модель поліноміальної регресії')  
plt.show()  
  
# Оцінимо якість моделі поліноміальної регресії  
mse\_poly = mean\_squared\_error(y, y\_pred\_poly)  
r2\_poly = r2\_score(y, y\_pred\_poly)  
print(f'Помилка середнього квадрату для поліноміальної регресії: {mse\_poly}')  
print(f'R^2 для поліноміальної регресії: {r2\_poly}')









**Завдання 2.6. Побудова кривих навчання**

import numpy as np

from matplotlib import pyplot as plt

from sklearn.metrics import mean\_squared\_error

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

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")

# Згенеруємо дані

m = 100

X = np.linspace(-3, 3, m)

y = 3 + np.sin(X) + np.random.uniform(-0.5, 0.5, m)

lin\_reg = LinearRegression()

plot\_learning\_curves(lin\_reg, X.reshape(-1, 1), y)

polynomial\_regression = Pipeline([

("poly\_features", PolynomialFeatures(degree=10,include\_bias=False)),

("lin\_reg", LinearRegression()),

])

plot\_learning\_curves(polynomial\_regression, X.reshape(-1, 1), y)

# Розділення даних на навчальний та валідаційний набори

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Перетворення ознак для поліноміальної регресії 2-го ступеня

poly = PolynomialFeatures(degree=2)

X\_train\_poly = poly.fit\_transform(X\_train.reshape(-1, 1))

X\_val\_poly = poly.transform(X\_val.reshape(-1, 1))

# Навчання поліноміальної моделі

model = LinearRegression()

model.fit(X\_train\_poly, y\_train)

# Передбачення на навчальному та валідаційному наборах

y\_train\_pred = model.predict(X\_train\_poly)

y\_val\_pred = model.predict(X\_val\_poly)

# Обчислення середньоквадратичної помилки

train\_error = mean\_squared\_error(y\_train, y\_train\_pred)

val\_error = mean\_squared\_error(y\_val, y\_val\_pred)

# Візуалізація кривої навчання

plt.figure(figsize=(10, 6))

plt.plot(X\_train, y\_train, 'b.', label='Навчальні дані')

plt.plot(X\_val, y\_val, 'r.', label='Валідаційні дані')

plt.plot(X\_train, y\_train\_pred, 'g-', label='Прогноз на навчальних даних')

plt.plot(X\_val, y\_val\_pred, 'y-', label='Прогноз на валідаційних даних')

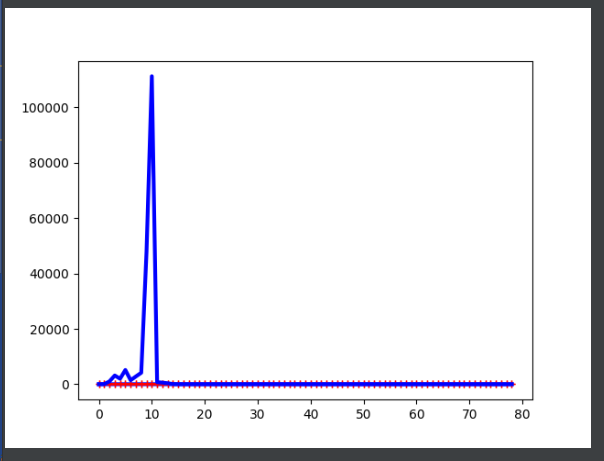
plt.title(f'Поліноміальна регресія 2-го ступеня\nTrain Error: {train\_error:.2f}, Validation Error: {val\_error:.2f}')

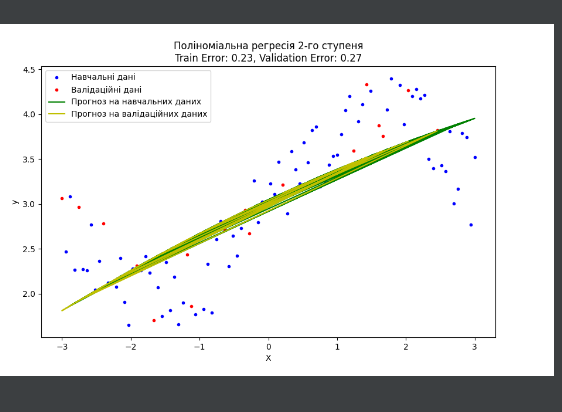
plt.xlabel('X')

plt.ylabel('y')

plt.legend()

plt.show()





ЧАСТИНА 2. ДОСЛІДЖЕННЯ МЕТОДІВ НЕКОНТРОЬОВАНОГО НАВЧАННЯ

**1. ТЕОРЕТИЧНІ ВІДОМОСТІ**

Термін навчання без вчителя (unsupervised learning) відноситься до процесу побудови моделі машинного навчання, що не вимагає залучення розмічених тренувальних даних. Машинне навчання без вчителя знаходить застосування у багатьох галузях, включаючи сегментування ринку, торгівля акціями, обробка природної мови, машинний зір та ін.

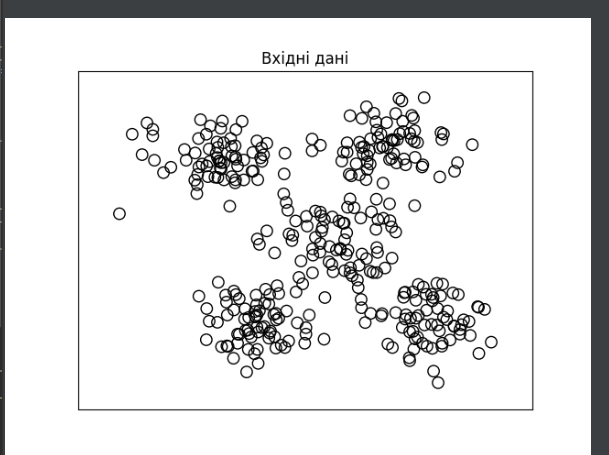
У попередніх ЛР ми мали справу з даними, з якими асоціювалися позначки (маркери). У разі помічених навчальних даних алгоритми вчаться класифікувати по цих мітках.

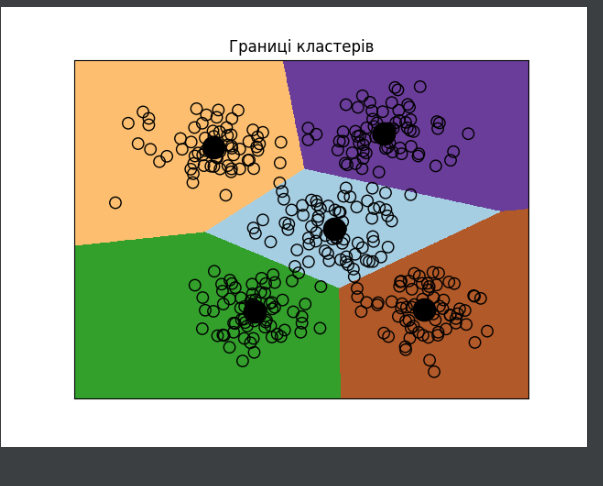
Алгоритми навчання без вчителя намагаються будувати моделі, які здатні знаходити підгрупи в заданому наборі даних, використовуючи різні метрики подібності.

Розглянемо, як формулюється завдання навчання, якщо воно проводиться без учителя. Коли у нас є набір даних, які не асоціюються з будь-якими мітками, ми припускаємо, що ці дані генеруються під впливом прихованих змінних, що управляють їх розподілом. У такому разі процес навчання може наслідувати якусь ієрархічну схему, використовуючи на початковому етапі індивідуальні точки даних. Далі можна створювати більш глибокі рівні представлення даних.

**2. ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ ТА МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ЙОГО ВИКОНАННЯ**

**Завдання 2.7. Кластеризація даних за допомогою методу k-середніх**





**Завдання 2.8. Кластеризація K-середніх для набору даних Iris**

import sklearn

from sklearn.datasets import load\_iris

from sklearn.cluster import KMeans

import numpy as np

import matplotlib.pyplot as plt

# Завантаження даних Iris

iris = load\_iris()

X = iris['data']

y = iris['target']

sklearn.cluster.KMeans(n\_clusters = 8, init = 'k-means++',n\_init = 10, max\_iter =

300, tol = 0.0001 , verbose = 0, random\_state =

None, copy\_x = True, algorithm = 'auto')

# Кластеризація з використанням K-Means

kmeans = KMeans(n\_clusters=5, n\_init=10)

kmeans.fit(X)

y\_kmeans = kmeans.predict(X)

# Візуалізація результатів кластеризації

plt.scatter(X[:, 0], X[:, 1], c=y\_kmeans, s=50, cmap='viridis')

centers = kmeans.cluster\_centers\_

plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5)

plt.show()

# Оголошення функції 'find\_clusters', яка буде використовуватися для пошуку кластерів і візуалізації їх

def find\_clusters(X, n\_clusters, rseed = 2):

# Створення генератора випадкових чисел з фіксованим насінням

rng = np.random.RandomState(rseed)

i = rng.permutation(X.shape[0])[:n\_clusters]

centers = X[i]

while True:

labels = pairwise\_distances\_argmin(X, centers)

# Обчислення нових центрів кластерів

new\_centers = np.array([X[labels == i].mean(0)

for i in range(n\_clusters)])

# Перевірка умови зупинки алгоритму

if np.all(centers == new\_centers):

break

centers = new\_centers

return centers, labels

centers, labels = find\_clusters(X, 3)

plt.scatter(X[:, 0], X[:, 1], c=labels,

s=50, cmap= 'viridis');

#Виклик функції find\_clusters для X з 3 кластерами та зі зміненим random\_state

centers, labels = find\_clusters(X, 3, rseed=0)

plt.scatter(X[:, 0], X[:, 1], c=labels,

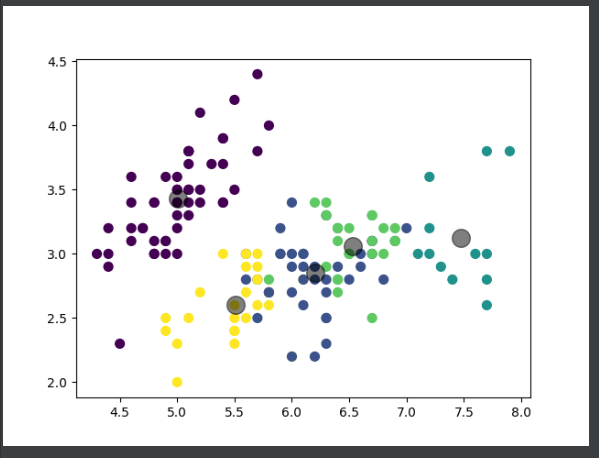
s=50, cmap= 'viridis');

#Використання K-Means моделі для кластеризації X з 3 кластерами і візуалізація результатів

labels = KMeans(3, random\_state=0).fit\_predict(X)

plt.scatter(X[:, 0], X[:, 1], c=labels,

s=50, cmap= 'viridis');



**Завдання 2.9. Оцінка кількості кластерів з використанням методу зсуву середнього**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import MeanShift,estimate\_bandwidth

from itertools import cycle

# Зчитуємо дані з файлу

X = np.loadtxt('data\_clustering.txt', delimiter=',')

# Оцінка ширини вікна для Х

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):

#Відобреження на графіку точок, які належать даному кластеру

plt.scatter(X[labels==i, 0], X[labels==i, 1], marker=marker,

color='black')

# Відображення на графіку центру кластера

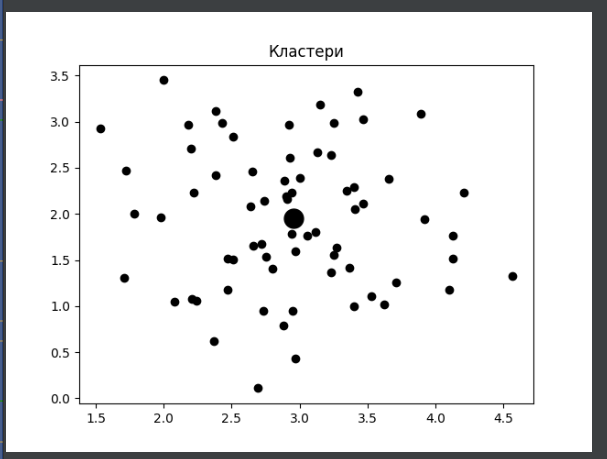
cluster\_center = cluster\_centers[i]

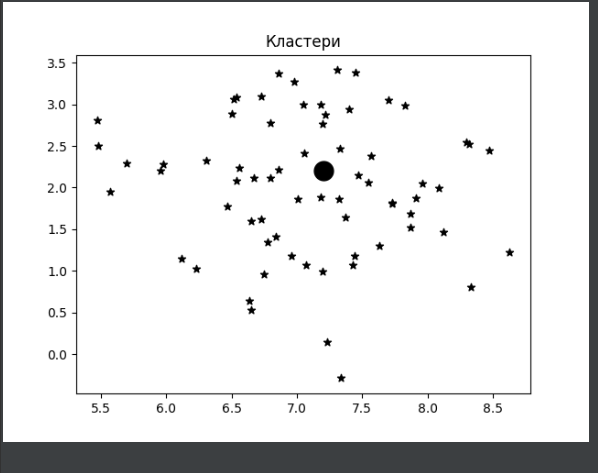
plt.plot(cluster\_center[0], cluster\_center[1], marker='o',

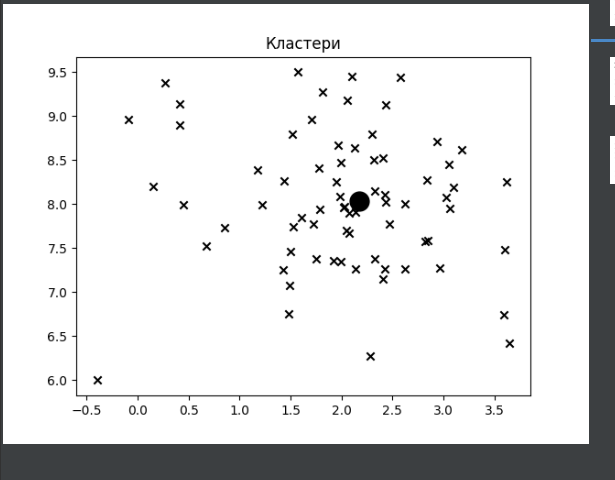
markerfacecolor='black', markeredgecolor='black',markersize=15)

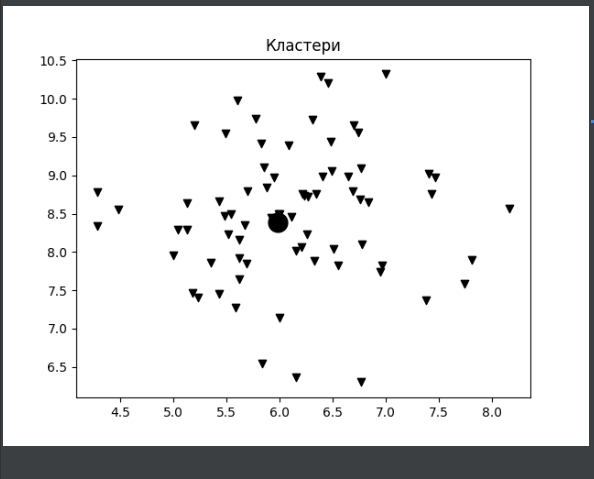
plt.title('Кластери')

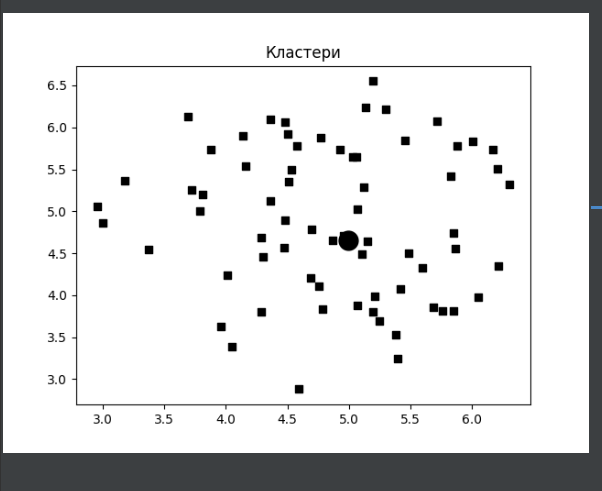
plt.show()











**Завдання 2.10. Знаходження підгруп на фондовому ринку з використанням моделі поширення подібності**

import datetime

import json

import numpy as np

from sklearn import covariance, cluster

import yfinance as yf

# Вхідний файл із символічними позначеннями компаній

input\_file = "company\_symbol\_mapping.json"

# Завантаження прив'язок символів компаній до їх повних назв

with open(input\_file, "r") as f:

company\_symbols\_map = json.loads(f.read())

symbols, names = np.array(list(company\_symbols\_map.items())).T

# Визначення архівних даних котирувань

start\_date = "2003-07-03"

end\_date = "2007-05-04"

# Завантаження архівних даних котирувань

quotes = []

valid\_symbols = []

for symbol in symbols:

try:

data = yf.download(symbol, start=start\_date, end=end\_date)

if not data.empty:

quotes.append(data)

valid\_symbols.append(symbol)

except Exception as e:

print(f"Failed to download data for {symbol}: {e}")

# Перевірка чи є валідні дані

if not quotes:

print(

"No valid data available for any symbol. Check your symbol mapping and data availability."

)

else:

# Оновлення символів на дійсні

symbols = valid\_symbols

# Вилучення котирувань, що відповідають відкриттю та закриттю біржі

opening\_quotes = np.array([quote["Open"].values for quote in quotes]).T

closing\_quotes = np.array([quote["Close"].values for quote in quotes]).T

# Обчислення різниці між двома видами котирувань

quotes\_diff = closing\_quotes - opening\_quotes

# Нормалізація даних

X = quotes\_diff.copy()

X /= X.std(axis=0)

# Створення моделі графа

edge\_model = covariance.GraphicalLassoCV()

# Навчання моделі

with np.errstate(invalid="ignore"):

edge\_model.fit(X)

# Створення моделі кластеризації на основі поширення подібності

\_, labels = cluster.affinity\_propagation(edge\_model.covariance\_)

num\_labels = labels.max()

# Виведення результатів

print("\nClustering of stocks based on difference in opening and closing quotes:\n")

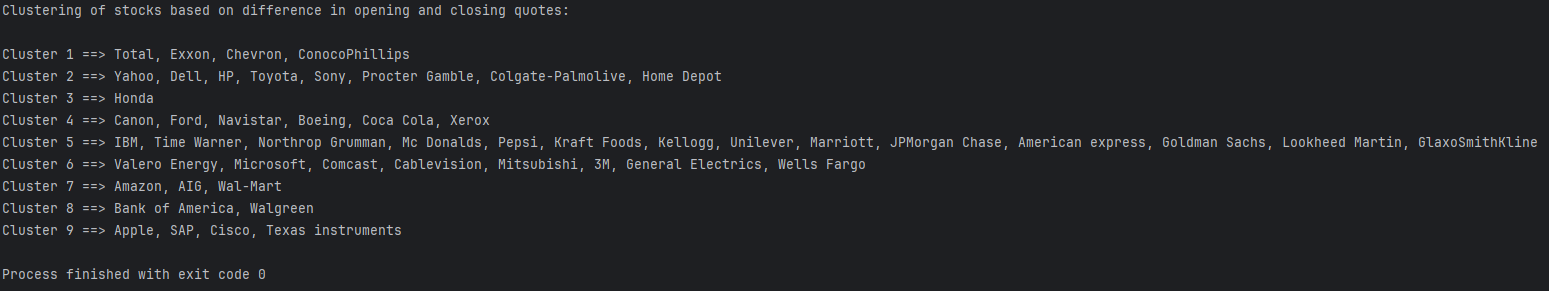
for i in range(num\_labels + 1):

cluster\_indices = np.where(labels == i)[0]

cluster\_names = names[cluster\_indices]

if len(cluster\_names) > 0:

print("Cluster", i + 1, "==>", ", ".join(cluster\_names))



Посилання на Git:

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