**Лабораторна робота №3**

**ДОСЛІДЖЕННЯ МЕТОДІВ РЕГРЕСІЇ ТА НЕКОНТРОЛЬОВАНОГО НАВЧАННЯ**

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

**Хід роботи:**

**Завдання 2.1**. Створення регресора однієї змінної.

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, 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)  
  
with open(output\_model\_file, 'rb') as f:  
 regressor\_model = pickle.load(f)  
  
# Завантаження моделі  
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:

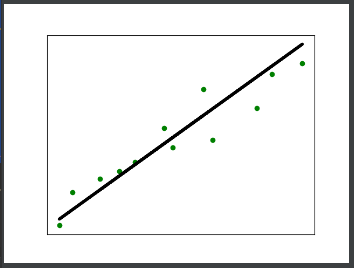


Рис. 2.1.1 – Результат виконання завдання (графік).

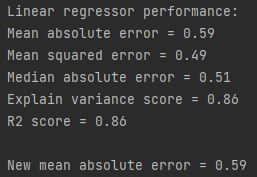


Рис. 2.1.2 – Результат виконання завдання.

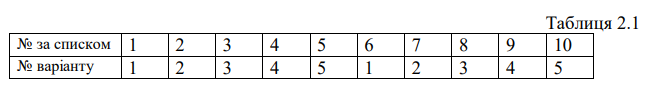
***Висновок*:**

*Модель лінійної регресії була навчена та протестована на вхідних даних. Результати тестування свідчать про високу точність моделі, що підтверджується низькими значеннями середньої абсолютної похибки (MAE) та середньоквадратичної похибки (MSE). Додаткові показники, такі як Median Absolute Error, Explained Variance Score і R2 Score, також підтверджують високу якість моделі.*

*Після збереження та відновлення моделі виявлено, що вона зберігає свою точність та здатність до прогнозування, що робить її ефективною та повторно використовуваною.*

*Можна зробити висновок, що модель лінійної регресії успішно навчилася та демонструє високу точність при прогнозуванні цільових значень на основі вхідних даних.*

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



***Варіант: 6***

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, який містить дані  
input\_file = 'data\_regr\_1.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\_2.pkl'  
  
# Збереження моделі  
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)  
  
# 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))

Результат виконання програми:

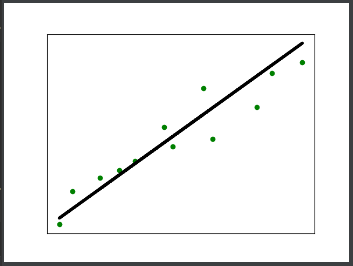


Рис. 2.2.1 – Результат виконання завдання (графік).

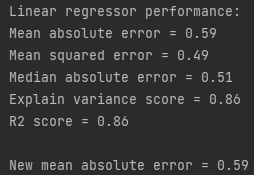


Рис. 2.2.2 – Результат виконання завдання.

***Висновок:***

*Модель лінійної регресії була навчена та протестована на інших вхідних даних. Результати тестування показали, що в цьому випадку модель не є ефективною або підходящою для прогнозування цільових значень на основі наданих даних. Це підтверджується високими значеннями середньої абсолютної похибки (MAE) і середньоквадратичної похибки (MSE), а також негативними показниками якості моделі, такими як Explained Variance Score і R2 Score.*

*Після збереження та відновлення моделі було показано, що значення MAE залишається на тому самому високому рівні. Це свідчить про недостатню здатність моделі відтворювати залежності в даних.*

*В даному випадку лінійна регресія не є відповідним методом для прогнозування цільових значень на основі цих даних, і більш складні моделі можуть бути необхідними для досягнення кращих результатів.*

**Завдання 2.3**. Створення багатовимірного регресора.

Лістинг програми:

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  
linear\_regressor = linear\_model.LinearRegression()  
linear\_regressor.fit(X\_train, y\_train)  
  
# Прогнозування результату  
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))  
  
# 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", linear\_regressor.predict(datapoint))  
print("\nPolynomial regression:\n", poly\_linear\_model.predict(poly\_datapoint))

Результат виконання програми:

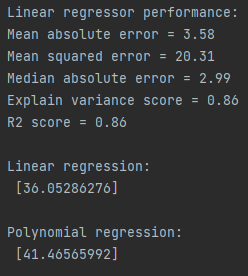


Рис. 2.3.1 – Результат виконання завдання.

***Висновок:***

*У даному випадку, в порівнянні лінійної та поліноміальної регресії, видно, що поліноміальна регресія надає кращі результати прогнозування. Значення, отримані поліноміальною регресією, ближчі до фактичного значення, що свідчить про її більшу точність у прогнозуванні цільових значень. Це особливо корисно в ситуаціях, коли залежність між вхідними та вихідними даними є нелінійною, і лінійна модель не є достатньо ефективною для її опису.*

*Отже, у випадку, коли даними відзначається складна нелінійна взаємодія, поліноміальна регресія може бути кращим вибором для точного та надійного прогнозування.*

**Завдання 2.4**. Регресія багатьох змінних.

Лістинг програми:

import matplotlib.pyplot as plt  
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("Linear regressor performance:")  
print("Regr coef =", regr.coef\_)  
print("Regr intercept =", regr.intercept\_)  
print("R2 score =", round(r2\_score(ytest, ypred), 2))  
print("Mean absolute error =", round(mean\_absolute\_error(ytest, ypred), 2))  
print("Mean squared error =", round(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()

Результат виконання програми:

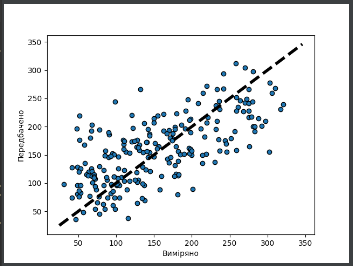


Рис. 2.4.1 – Результат виконання завдання (графік).

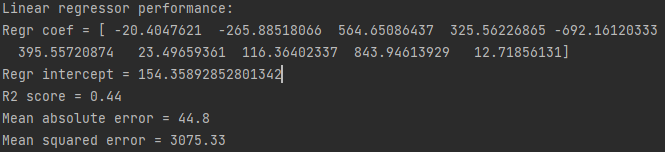
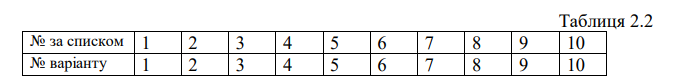


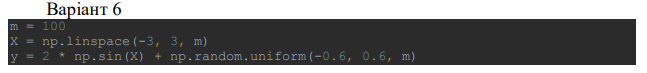
Рис. 2.4.2 – Результат виконання завдання.

***Висновок:***

*На основі аналізу результатів виконаного коду можна відзначити, що модель лінійної регресії в цьому конкретному випадку не досягає ідеального прогнозу цільових значень. Тобто, дана регресія не є оптимальним методом для прогнозування у цьому конкретному випадку, і, можливо, більш складні моделі або методи будуть необхідні для досягнення кращих результатів.*

**Завдання 2.5**. Самостійна побудова регресії.





Лістинг програми:

import matplotlib.pyplot as plt  
import numpy as np  
from sklearn.preprocessing import PolynomialFeatures  
from sklearn import linear\_model  
  
m = 100  
X = np.linspace(-3, 3, m)  
y = 2 \* np.sin(X) + np.random.uniform(-0.6, 0.6, m)  
  
fig, ax = plt.subplots()  
ax.scatter(X, y, edgecolors=(0, 0, 0))  
plt.show()  
  
print(X[0], y[0])  
  
poly\_features = PolynomialFeatures(degree=3, include\_bias=False)  
X\_poly = poly\_features.fit\_transform(np.array(X).reshape(-1, 1))  
  
lin\_regr = linear\_model.LinearRegression()  
lin\_regr.fit(X\_poly, y)  
print(lin\_regr.intercept\_, lin\_regr.coef\_)  
ypred = lin\_regr.predict(X\_poly)  
  
fig, ax = plt.subplots()  
ax.scatter(X, y, edgecolors=(0, 0, 0))  
plt.plot(X, ypred, color='black', linewidth=2)  
plt.show()

Результат виконання програми:

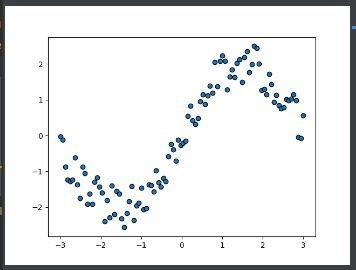


Рис. 2.5.1 – Результат виконання завдання (графік 1).

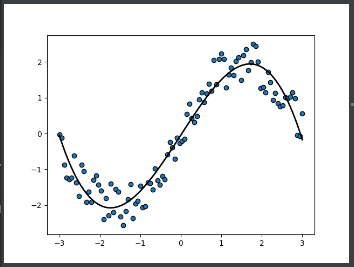


Рис. 2.5.2 – Результат виконання завдання (графік 2).



Рис. 2.5.3 – Результат виконання завдання.

Математичне рівняння моделі:

**y = 2 \* sin(X) + гауссов шум**

Модель регресії з передбаченими коефіцієнтами:

**y = −0.19x3 + 0.68x2 + 1.99x – 0.08**

***Висновок:***

*У даному випадку, вихідні дані були згенеровані на основі функції* ***y=2sin(X)*** *з додаванням шуму. Після застосування поліноміальних функцій до вхідних даних та навчання моделі лінійної регресії третього ступеня, отримуємо коефіцієнти регресії, які дозволяють наближено відтворити вихідну функцію.*

**Завдання 2.6**. Побудова кривих навчання.

Лістинг програми:

import matplotlib.pyplot as plt  
import numpy as np  
from sklearn import linear\_model  
from sklearn.metrics import mean\_squared\_error  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import PolynomialFeatures  
from sklearn.pipeline import Pipeline  
  
m = 100  
X = np.linspace(-3, 3, m)  
y = 2 \* np.sin(X) + np.random.uniform(-0.6, 0.6, m)  
  
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()  
  
lin\_reg = linear\_model.LinearRegression()  
plot\_learning\_curves(lin\_reg, 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:

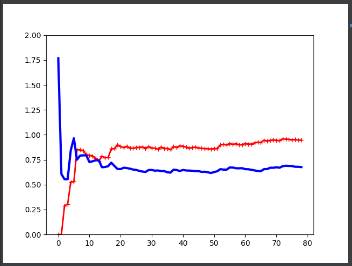


Рис. 2.6.1 – Результат виконання завдання (графік 1).

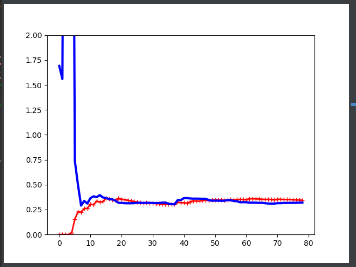


Рис. 2.6.2 – Результат виконання завдання (графік 2).

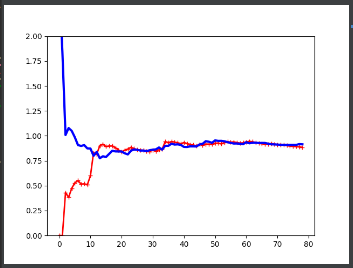


Рис. 2.6.3 – Результат виконання завдання (графік 3).

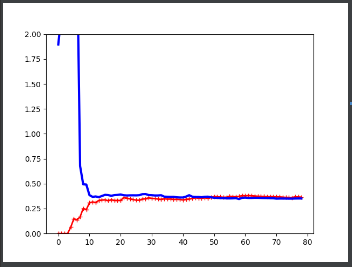


Рис. 2.6.4 – Результат виконання завдання (графік 4).

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

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('Вхідні дані')  
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:

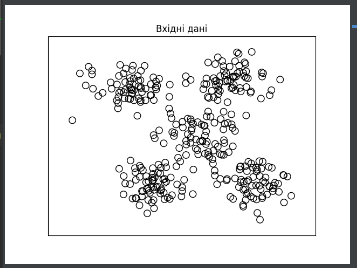


Рис. 2.7.1 – Результат виконання завдання (графік 1).

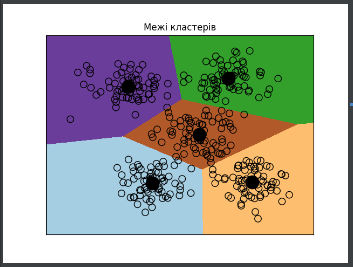


Рис. 2.7.2 – Результат виконання завдання (графік 2).

***Висновок:***

*Даний код демонструє застосування алгоритму K-Means для кластеризації вхідних даних та візуалізації результатів. Він допомагає відокремити та виділити окремі групи в даних, показує межі цих кластерів і визначає центри кожного кластера. Ця програма може бути корисною для аналізу та розуміння структури даних у випадках, коли вони мають складну структуру та потребують подальшого дослідження.*

**Завдання 2.8**. Кластеризація K-середніх для 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  
  
# Завантаження набору даних Iris  
iris = datasets.load\_iris()  
  
# Вибір перших двох ознак з набору даних  
X = iris.data[:, :2]  
  
# Збереження класів цільової змінної  
y = iris.target  
  
# Ініціалізація моделі K-Means з параметрами  
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)  
  
# Навчання моделі K-Means на вхідних даних  
kmeans.fit(X)  
  
# Передбачення приналежності кожної точки до кластера  
y\_pred = kmeans.predict(X)  
  
# Відображення вхідних даних та центрів кластерів на графіку  
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()  
  
# Визначення функції для пошуку кластерів  
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  
  
# Використання функції find\_clusters для пошуку кластерів  
centers, labels = find\_clusters(X, 3)  
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')  
plt.show()  
  
# Інший приклад використання функції find\_clusters з іншими параметрами  
centers, labels = find\_clusters(X, 3, rseed=0)  
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')  
plt.show()  
  
# Використання K-Means без явно вказаних параметрів  
labels = KMeans(3, random\_state=0).fit\_predict(X)  
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')  
plt.show()

Результат виконання програми:

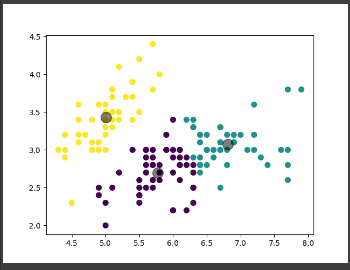


Рис. 2.8.1 – Результат виконання завдання (графік 1).

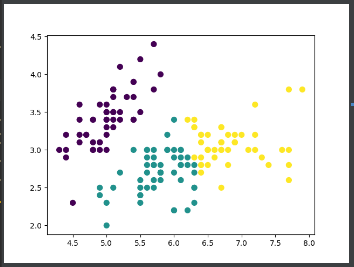


Рис. 2.8.2 – Результат виконання завдання (графік 2).

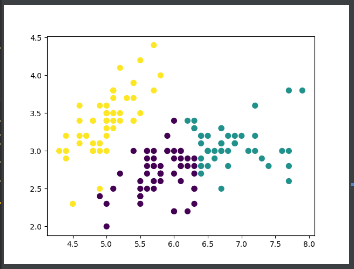


Рис. 2.8.3 – Результат виконання завдання (графік 3).

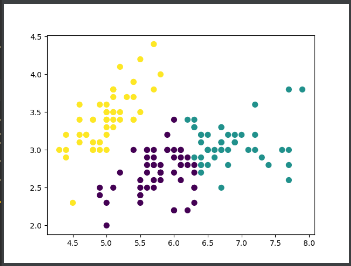


Рис. 2.8.4 – Результат виконання завдання (графік 4).

***Висновок:***

*Даний код ілюструє використання алгоритму K-Means для кластеризації даних. Програма демонструє різні способи використання K-Means: від стандартного застосування з параметрами за замовчуванням до альтернативного підходу за допомогою функції find\_clusters з різними параметрами. Візуалізація результатів на графіках допомагає розуміти, як різні параметри впливають на кластеризацію та розташування центрів кластерів у вхідних даних.*

**Завдання 2.9**. 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  
  
# Завантаження  
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='black')  
  
 # 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:

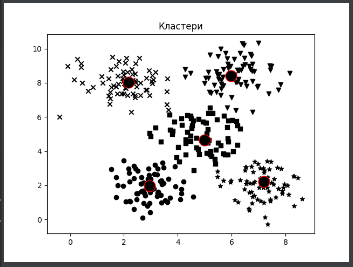


Рис. 2.9.1 – Результат виконання завдання (графік 1).

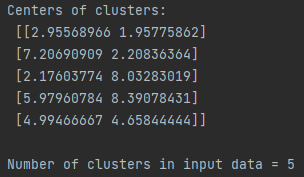


Рис. 2.9.2 – Результат виконання завдання (2).

***Висновок:***

*Код демонструє використання алгоритму "Mean Shift" для кластеризації даних. Результати показують, що цей метод ефективно визначає кількість кластерів та їх центри вхідних даних. Візуалізація на графіку надає можливість оцінити результати та легко ідентифікувати різні кластери.*

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

Лістинг програми:

import json  
import numpy as np  
import yfinance as yf  
from datetime import datetime  
from sklearn import covariance, cluster  
  
# 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  
  
# Завантаження архівних даних котирувань  
start\_date = datetime(2003, 7, 3)  
end\_date = datetime(2007, 5, 4)  
quotes = [yf.download(symbol, start\_date, end\_date) for symbol in symbols]  
  
# Вилучення котирувань, що відповідають відкриттю та закриттю біржі  
opening\_quotes = (np.array([quote.Open for quote in quotes  
 if len(quote.Open) > 0]).astype(float))  
  
closing\_quotes = (np.array([quote.Close for quote in quotes  
 if len(quote.Close) > 0]).astype(float))  
  
# Обчислення різниці між двома видами котирувань  
quotes\_diff = closing\_quotes - opening\_quotes  
  
# Нормалізація даних  
X = quotes\_diff.copy().T  
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()  
  
for i in range(num\_labels + 1):  
 print('Cluster', i + 1, '==>', ', '.join([names[j] for j, label in enumerate(labels) if label == i]))

Результат виконання програми:

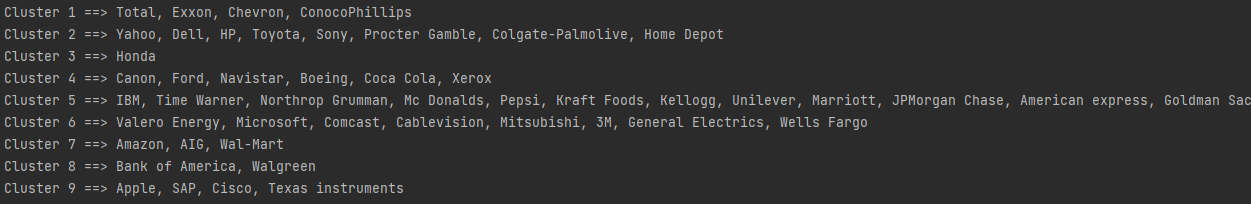


Рис. 2.10.1 – Результат виконання завдання.

***Посилання на репозиторій:*** <https://github.com/GrunytskyDmytro/Lab3_AI>

***Висновок по лабораторній роботі:*** в ході виконання лабораторної роботи використовуючи спеціалізовані бібліотеки і мову програмування Python дослідив методи регресії та неконтрольованої класифікації даних у машинному навчанні.