**ЛАБОРАТОРНА РОБОТА № 1**

**ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ ДАНИХ**

**Мета роботи:**

використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити попередню обробку та класифікацію даних.

**Хід роботи:**

**Завдання 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:]

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)

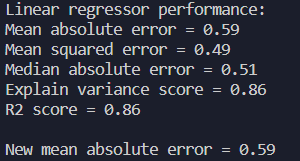
*with* open(output\_model\_file, 'rb') *as* f:

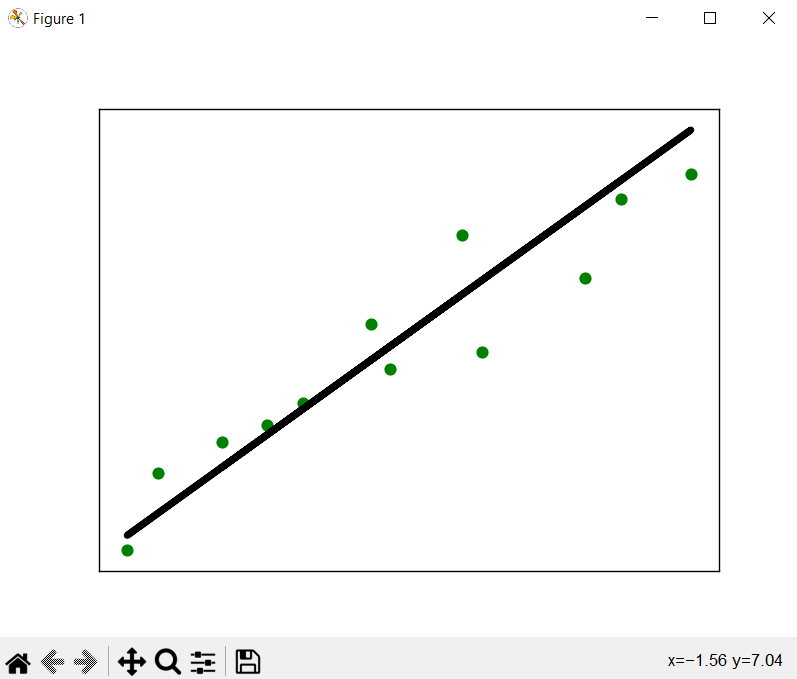
    regressor\_model = pickle.load(f)

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:

****

****

Висновок: модель досягла високого рівня точності у передбаченні цільових значень на основі навчальних даних. Показник поясненої варіації (Explained Variance Score) та Коефіцієнт детермінації (R2 Score) мають хорошу оцінку (0.86), що вказує на те, що модель добре пояснює дисперсію вихідних даних та має високу якість підгонки моделі до даних.

**Завдання 2. Передбачення за допомогою регресії однієї змінної**

Код 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:]

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:

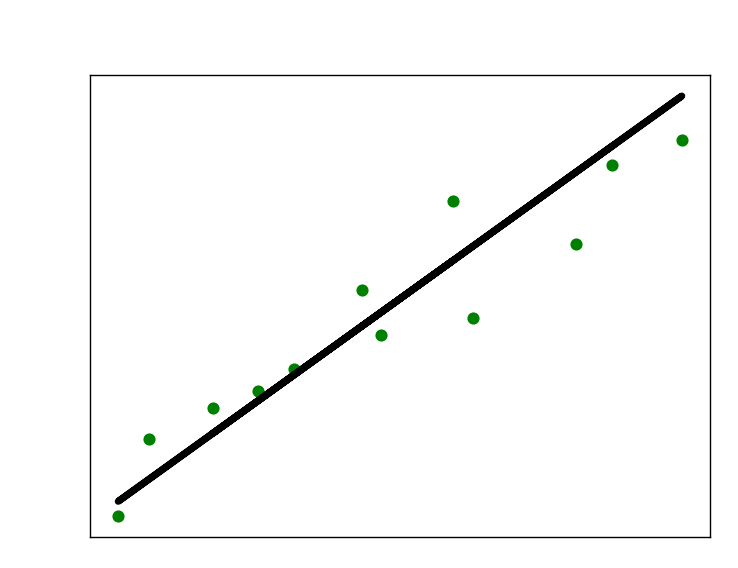
    pickle.dump(regressor, f)

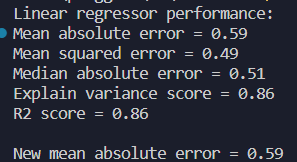
*with* open(output\_model\_file, 'rb') *as* f:

    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:



Висновок: результат роботи програми є ідентичним до минулого завдання, що свідчить про те що вхідні дані у файлі data\_regr\_1.txt такі самі як у файлі data\_singlevar\_regr.txt.

**Завдання 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

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)

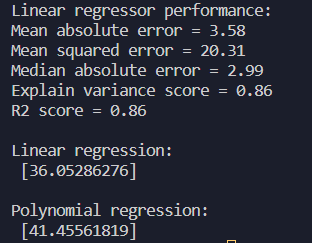
print("\nLinear regression:\n",

      linear\_regressor.predict(datapoint))

print("\nPolynomial regression:\n",

      poly\_linear\_model.predict(poly\_datapoint))

Результат виконання:



Висновок: Поліноміальна регресія дає більш точні прогнози, ніж лінійна регресія, для даних з кількома змінними та нелінійним зв'язком між змінними оскільки його значення ближче до значення 41.35.

Завдання 4. Регресія багатьох зміннихЛістинг 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)

regr = linear\_model.LinearRegression()

regr.fit(X\_train, y\_train)

ypred = regr.predict(X\_test)

print("regr.coef\_ =", regr.coef\_)

print("regr.intercept\_ =", regr.intercept\_)

print("r2\_score =", round(r2\_score(y\_test, ypred), 2))

print("Mean\_absolute\_error =", round(mean\_absolute\_error(y\_test, ypred), 2))

print("Mean\_squared\_error =", round(mean\_squared\_error(y\_test, ypred), 2))

fig, ax = plt.subplots()

ax.scatter(y\_test, ypred, edgecolors=(0, 0, 0))

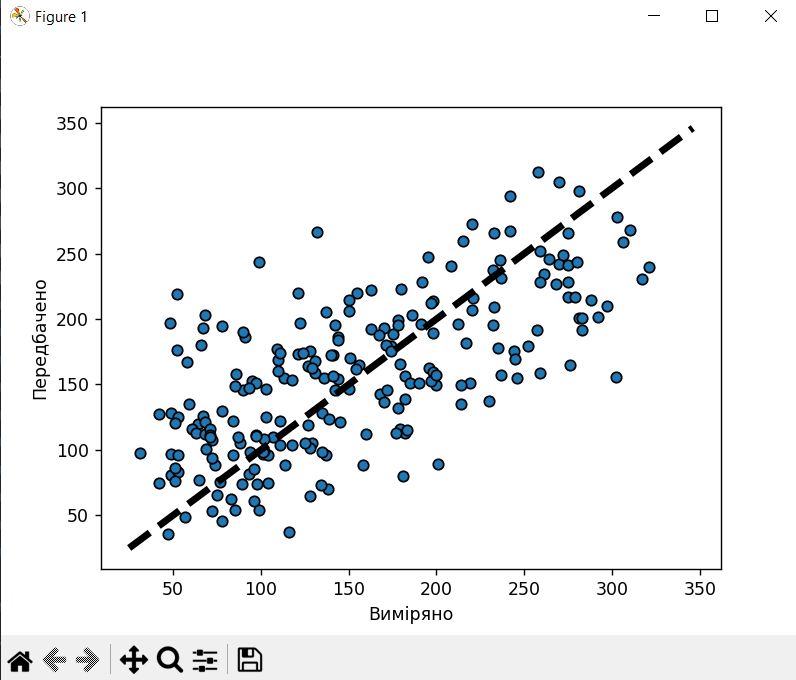
ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=4)

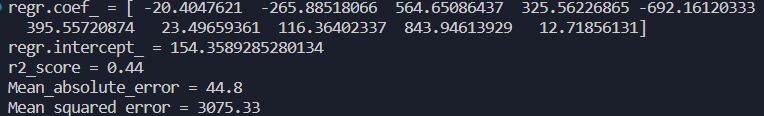
ax.set\_xlabel('Виміряно')

ax.set\_ylabel('Передбачено')

plt.show()

Результат виконання програми:



******

Висновок: Оскільки середня абсолютна і середня квадратична похибки досить високі, можна зробити висновок, що модель не дуже точна, але вона все ж таки може бути корисною для отримання приблизних прогнозів передбачених значень.

Завдання 5. Самостійна побудова регресії

Лістинг програми:

*import* matplotlib.pyplot *as* plt

*import* numpy *as* np

*from* sklearn *import* linear\_model

*from* sklearn.preprocessing *import* PolynomialFeatures

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()

poly\_features = PolynomialFeatures(degree=3, include\_bias=False)

X\_poly = poly\_features.fit\_transform(np.array(X).reshape(-1, 1))

print(X[0], y[0])

*# print(X\_poly)*

lin\_reg = linear\_model.LinearRegression()

lin\_reg.fit(X\_poly, y)

print(lin\_reg.intercept\_, lin\_reg.coef\_)

y\_pred = lin\_reg.predict(X\_poly)

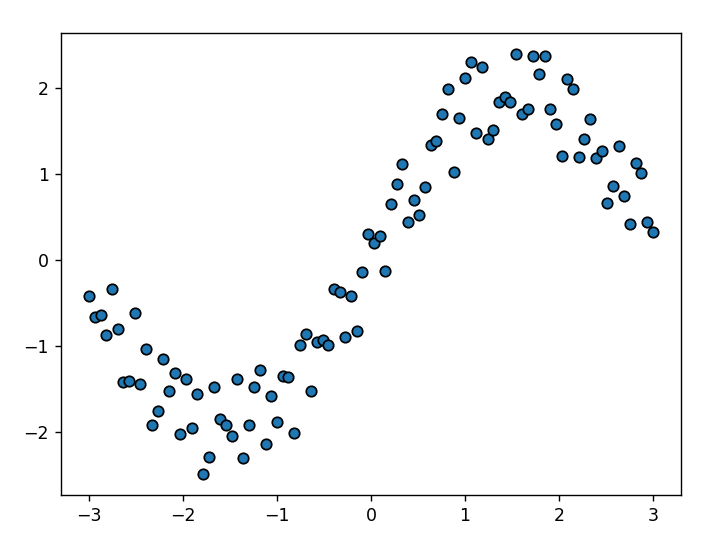
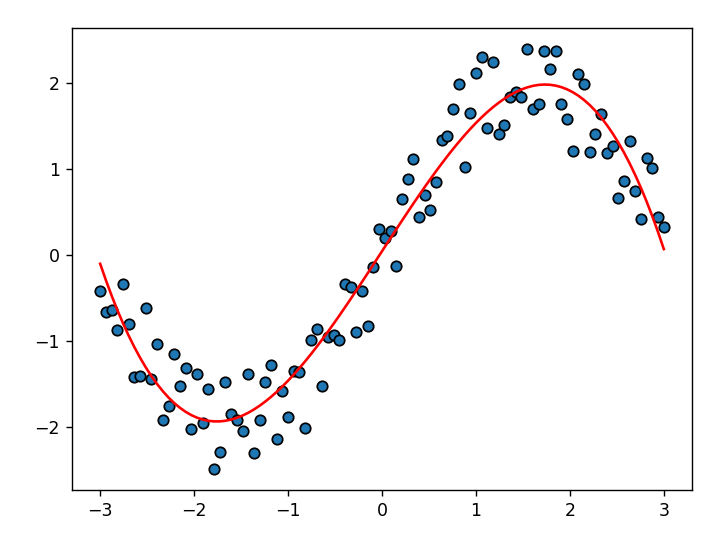
fig, ax = plt.subplots()

ax.scatter(X, y, edgecolors=(0, 0, 0))

plt.plot(X, y\_pred, color='red')

plt.show()

Результат виконання роботи:

******

******

Модель у вигляді математичного рівняння: y = 2\*sin(x) + випадковий шум

Модель регресії з передбаченими коефіцієнтами: y = 1.68x2 – 0.00x – 0.18.

З графіку видно, що модель поліноміальної регресії досить точно показує вхідні дані.

Завдання 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)

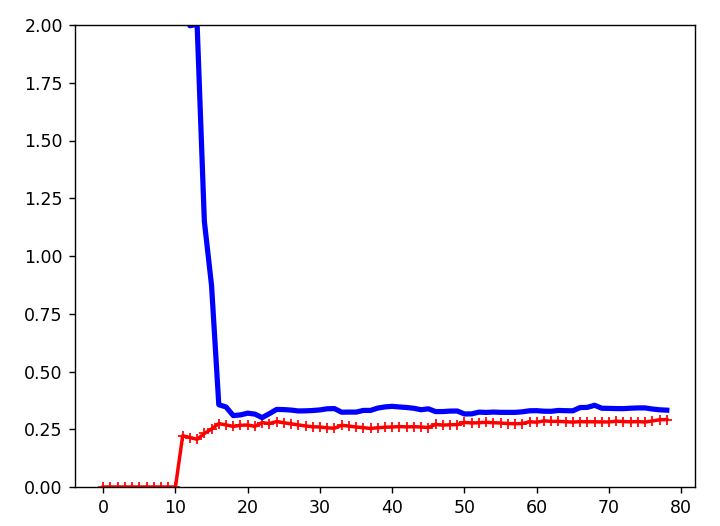
polynomial\_regression = Pipeline([

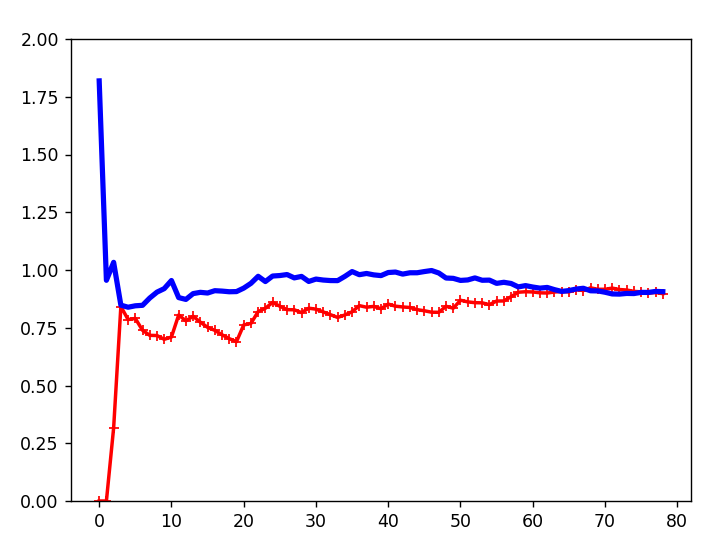
    ('poly\_features', PolynomialFeatures(degree=10, include\_bias=False)),

    ('lin\_reg', linear\_model.LinearRegression()),

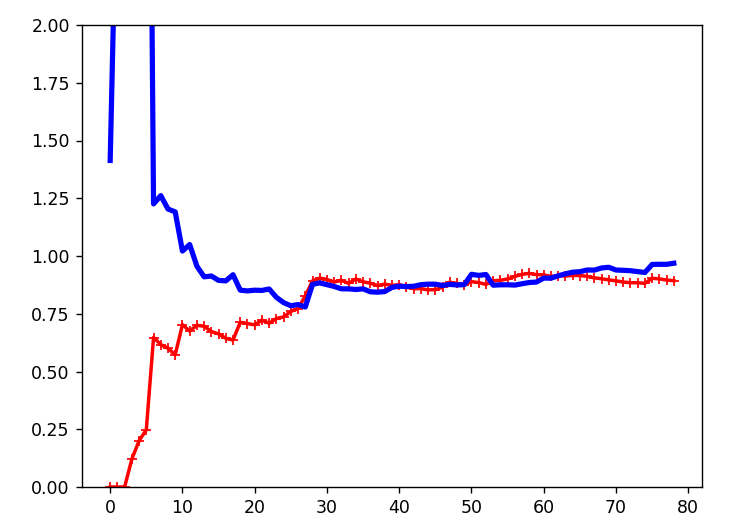
])

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

Результат виконання програми:



Криві навчання поліноміальної моделі 2-го ступеня



Завдання 7. Кластеризація даних за допомогою методу k-середніх

Лістинг програми:

*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('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)

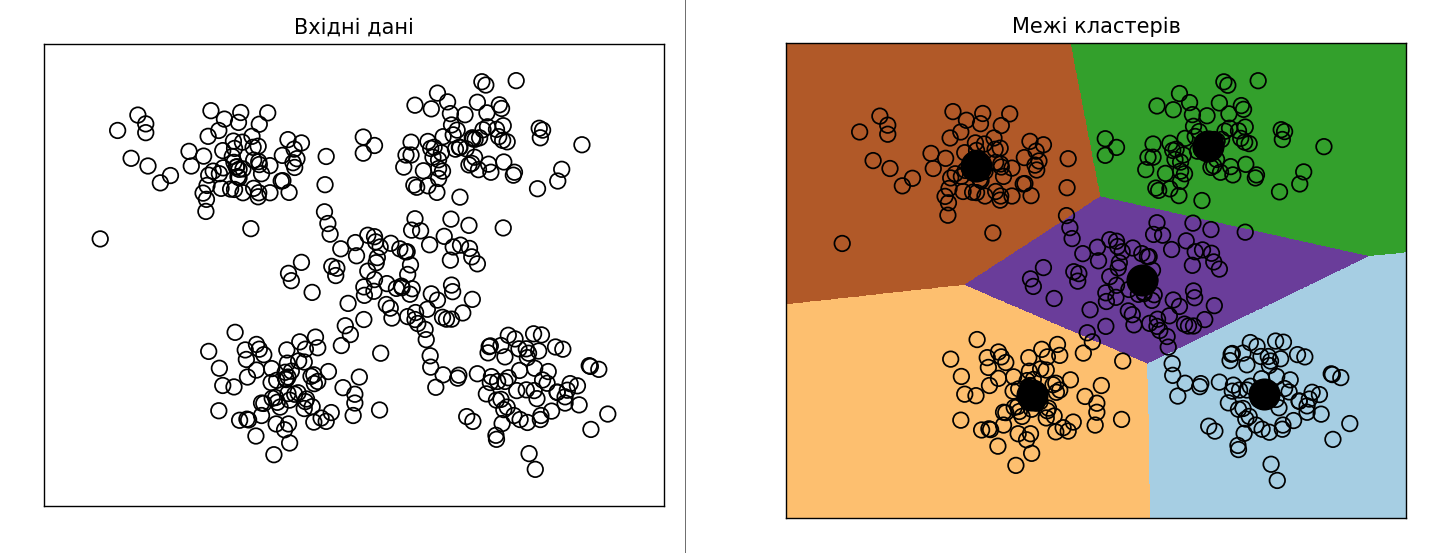
plt.ylim(y\_min, y\_max)

plt.xticks(())

plt.yticks(())

plt.show()

Результат виконання програми:



З зображень видно, що алгоритм K-Means добре впорався з кластеризацією даних, оскільки межі кластерів добре відображають розподіл даних.

Завдання 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

iris = datasets.load\_iris()

X = iris.data[:, :2]

Y = iris.target

*# створення об'єкту K-Means з вказаними параметрами для подальшої кластеризації даних*

kmeans = KMeans(n\_clusters=5, init='k-means++', n\_init=10, max\_iter=300, tol=0.0001,

                verbose=0, random\_state=None, copy\_x=True, algorithm='auto')

*#обчислення к-середнього кластеризуванння*

kmeans.fit(X)

*# Обчислюэкластерні центри та передбаэ індекс кластера для кожного зразка.*

y\_kmeans = kmeans.predict(X)

plt.figure()

*# візуалізація результатів кластерізації з пошуком 5 кластерів*

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()

*# метод приймає набір даних - Х, кількість кластерів, що шукаємо - n\_clusters та rseed для генерації випадкових чисел*

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

*# створення {n\_clusters} випадкових центрів кластерів з точок набору даних*

    rng = np.random.RandomState(rseed)

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

    centers = X[i]

*while* True:

*# оцінка приналежності точки до кожного центру*

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

*# обчислення нового центру кластера як середньогог значення всіх точок кластеру*

        new\_centers = np.array([X[labels == i].mean(0) *for* i *in* range(n\_clusters)])

*# якщо усі нові центри та старі ідентичні - цикл завершується*

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

*break*

        centers = new\_centers

*return* centers, labels

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

*#візуалізація результатів кластерізації методом find\_clusters() з трьома кластерами і значенням для генератора випадкових чисел 2*

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

plt.show()

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

*#візуалізація результатів кластерізації методом find\_clusters() з трьома кластерами і значенням для генератора випадкових чисел 0*

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

plt.show()

*#візуалізація результатів кластерізації з трьома кластерами*

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:

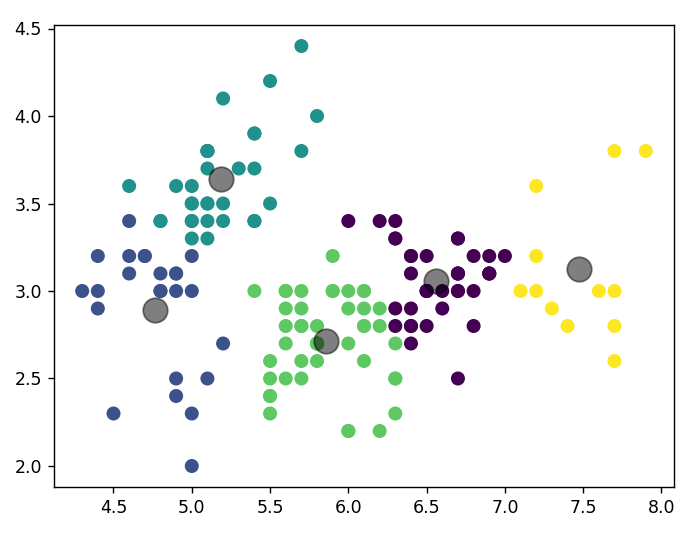


Рис. 8.2. дані розподілені на 5 кластерів за допомогою алгоритму K-Means

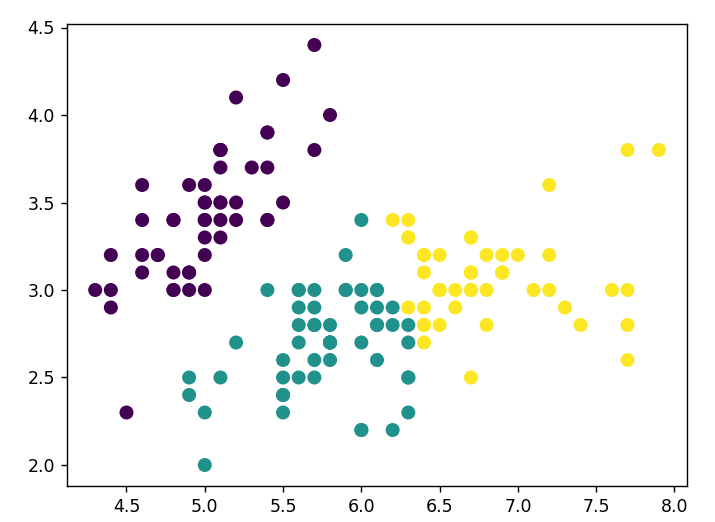


Рис. 8.1. дані розподілені на 3 кластери методом find\_clusters()

з параметром rseed = 2

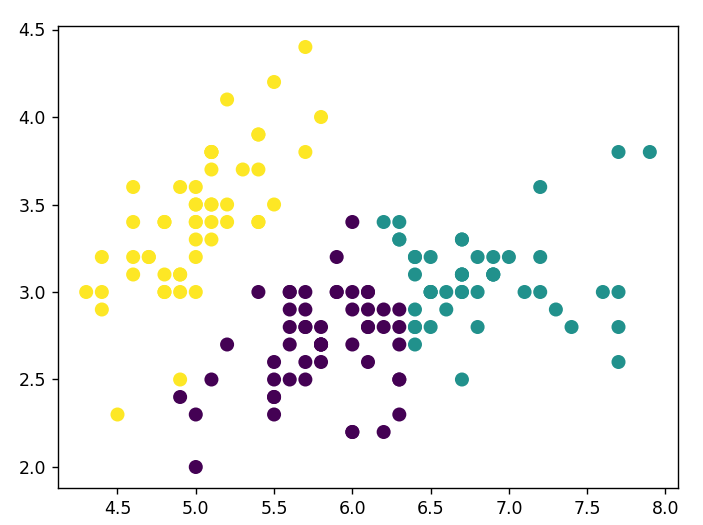


Рис. 8.3. дані розподілені на 3 кластери методом find\_clusters()

з параметром rseed = 0

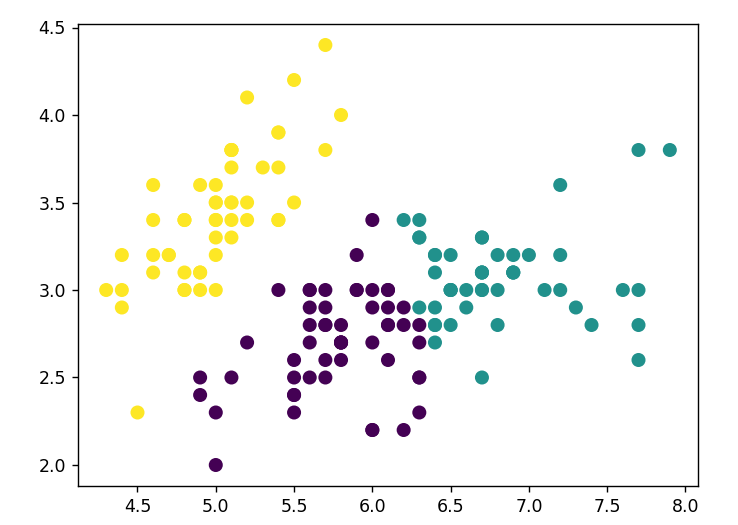


Рис. 8.4. дані розподілені на 3 кластери за допомогою алгоритму K-Means

Висновок: Результат кластеризації даних за допомогою алгоритму K-Means з 5 кластерами є більш чітким і зручним для інтерпретації, ніж результат кластеризації даних методом find\_clusters() з 3 кластерами. Результат кластеризації даних методом find\_clusters() залежить від значення для генератора випадкових чисел, хоча й не суттєво.

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',

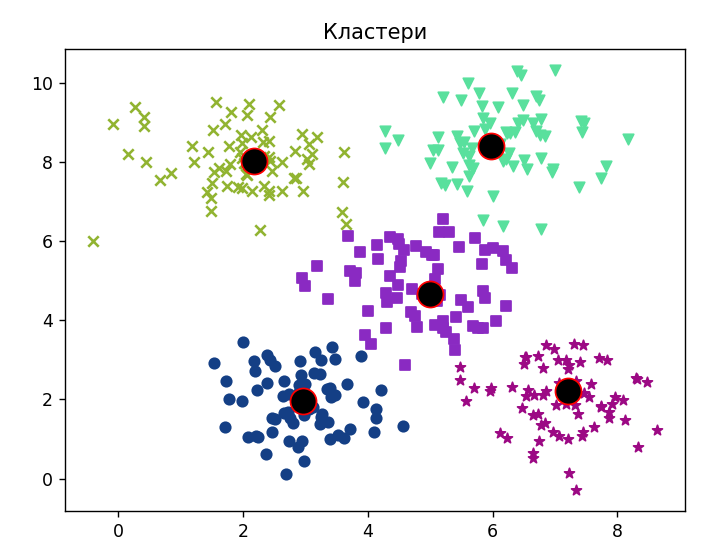
             markerfacecolor='black', markeredgecolor='red',

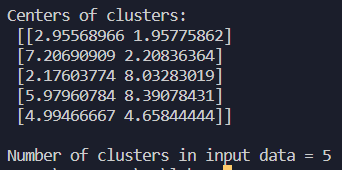
             markersize=15)

plt.title('Кластери')

plt.show()

Результат виконання:





Висновок: Метод зсуву середнього досить точно зміг визначити центри кластерів і визначив 5 кластерів.

Завдання 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 = "2003-07-03"

end\_date = "2007-05-05"

quotes = [yf.download(symbol, start=start\_date, end=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

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*

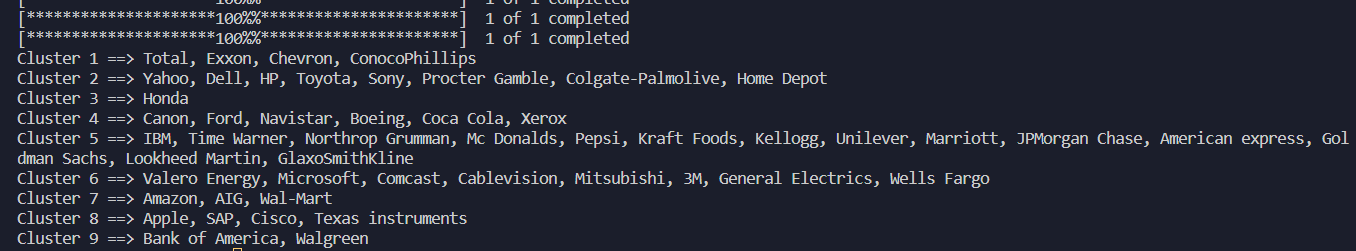
\_, labels = cluster.affinity\_propagation(edge\_model.covariance\_)

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:



***Github:*** [***https://github.com/mtvi/ipz202\_Shcherback\_lab3***](%20https:/github.com/mtvi/ipz202_Shcherback_lab3)

***Висновки:*** використовуючи спеціалізовані бібліотеки та мову програмування Python дослідили попередню обробку та класифікацію даних.