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

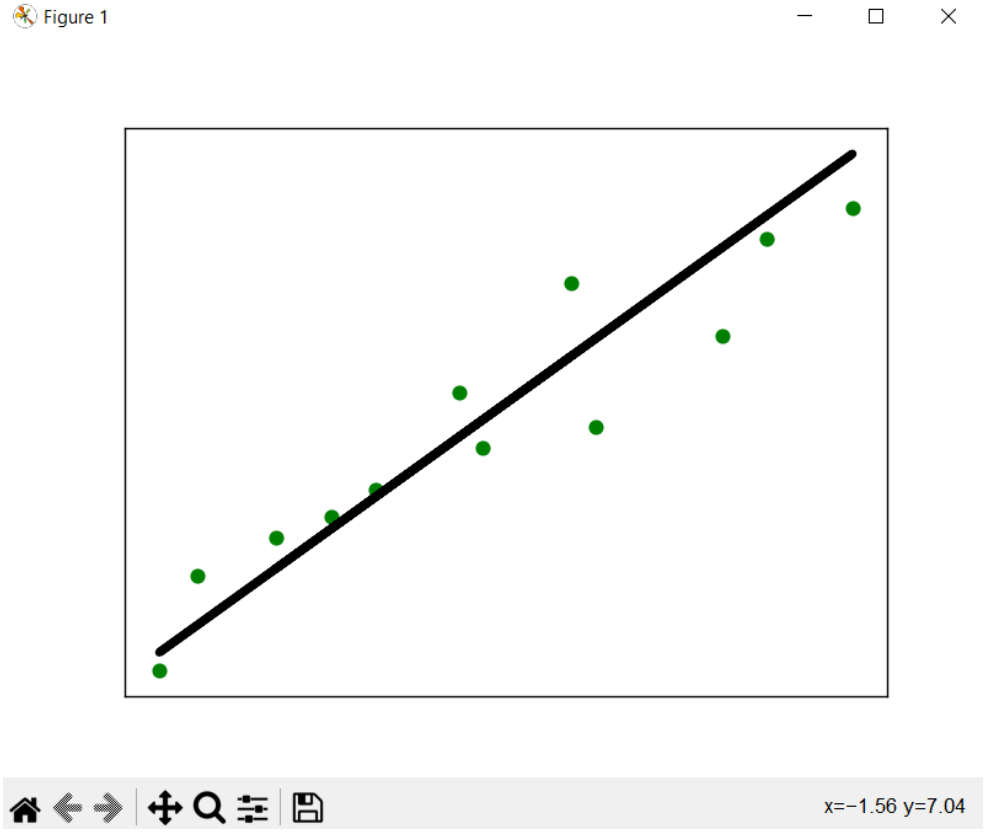
Лістинг програми:

```
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:

```
Linear regressor performance:
Mean absolute error = 0.59
Mean squared error = 0.49
Median absolute error = 0.51
Explain variance score = 0.86
R2 score = 0.86

New mean absolute error = 0.59
```



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.

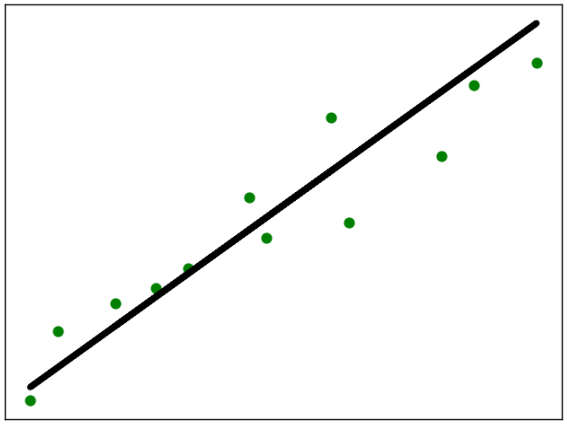
Завдання 2. Передбачення за допомогою регресії однієї змінної

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:



```
Linear regressor performance:
Mean absolute error = 0.59
Mean squared error = 0.49
Median absolute error = 0.51
Explain variance score = 0.86
R2 score = 0.86

New mean absolute error = 0.59
```

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:

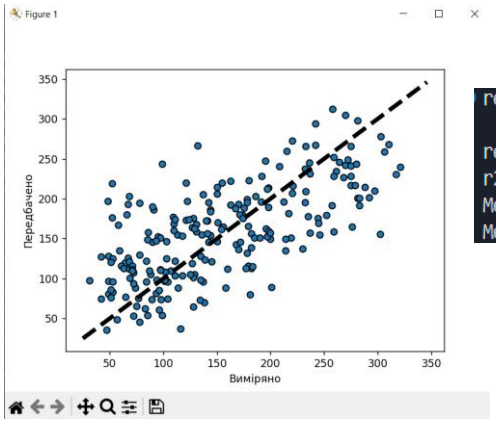
```
Linear regressor performance:
Mean absolute error = 3.58
Mean squared error = 20.31
Median absolute error = 2.99
Explain variance score = 0.86
R2 score = 0.86

Linear regression:
 [36.05286276]

Polynomial regression:
 [41.45561819]
```

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 змінних
Лістинг програми:

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from sklearn.model_selection import train_test_split

diabetes = datasets.load_diabetes()
X = diabetes.data
y = diabetes.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, ran-
dom_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:




```
regr.coef_ = [ -20.4047621  -265.88518066  564.65086437  325.56226865 -692.16120333
  395.55720874   23.49659361  116.36402337  843.94613929   12.71856131]
regr.intercept_ = 154.3589285280134
r2_score = 0.44
Mean_absolute_error = 44.8
Mean_squared_error = 3075.33
```

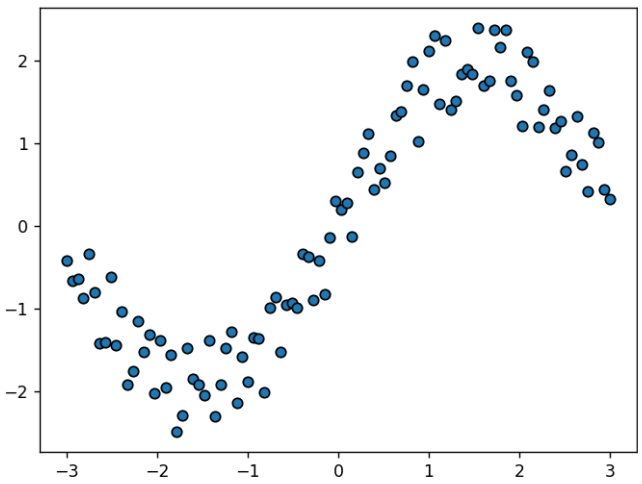
Висновок: Оскільки середня абсолютна і середня квадратична похибки досить високі, можна зробити висновок, що модель не дуже точна, але вона все ж таки може бути корисною для отримання приблизних прогнозів передбачених значень.

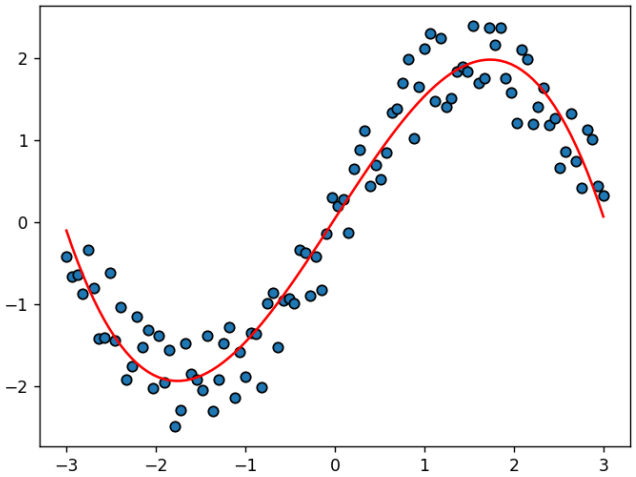
Завдання 5. Самостійна побудова регресії

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:





```
-3.0 -0.4246172518655525
0.042948655148540224 [ 1.68424988 -0.00673442 -0.18397459]
```

Модель у вигляді математичного рівняння: y = 2*sin(x) + випадковий шум
Модель регресії з передбаченими коефіцієнтами: $y = 1.68x^2 – 0.00x – 0.18$.
З графіку видно, що модель поліноміальної регресії досить точно показує вхідні дані.

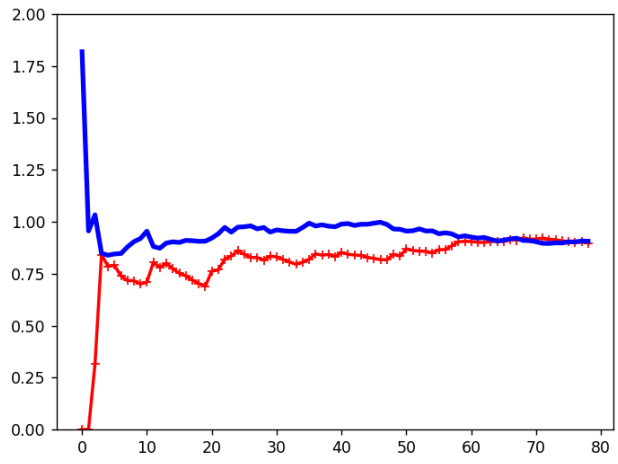
Завдання 6. Побудова кривих навчання

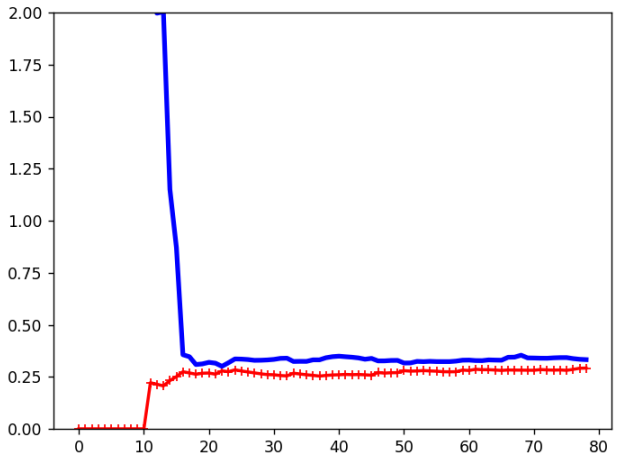
Лістинг програми:

```
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)
```

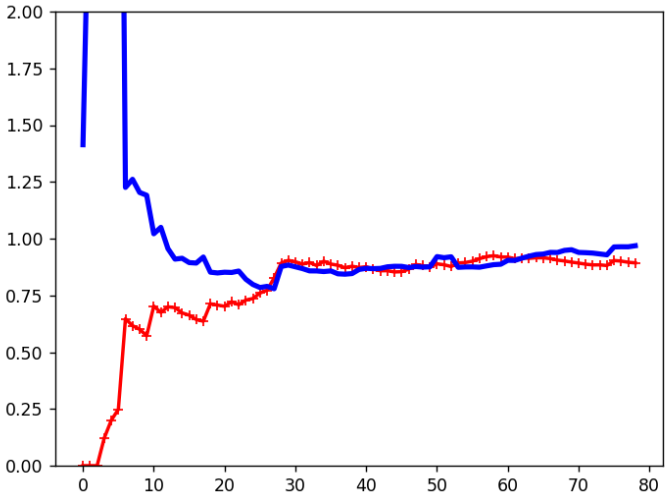
```
        train_errors.append(mean_squared_error(y_train_predict, y_train[:m]))
        val_errors.append(mean_squared_error(y_val_predict, y_val))
    fig, ax = plt.subplots()
    plt.ylim(0, 2)
    ax.plot(np.sqrt(train_errors), "r-+", linewidth=2, label='train')
    ax.plot(np.sqrt(val_errors), "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



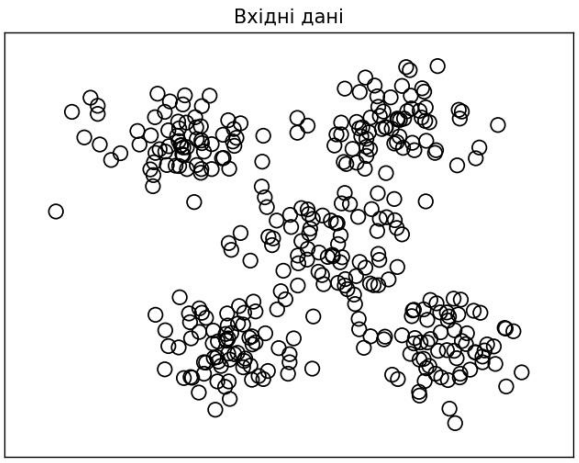
Завдання 7. Кластеризація даних за допомогою методу k-середніх

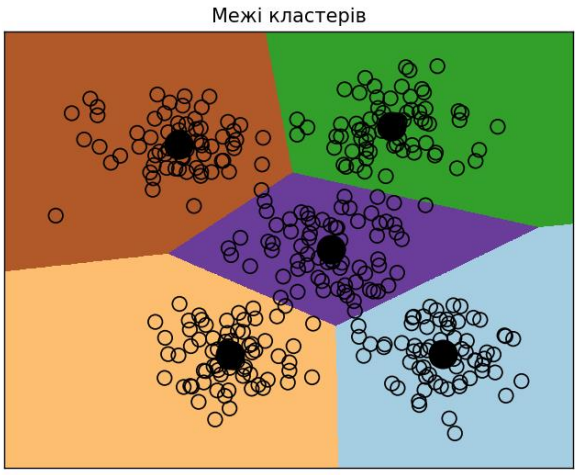
Лістинг програми:

```
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, lin-
ewidths=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 кластеризуваннння
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 - X, 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 кла-
стеру
        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:

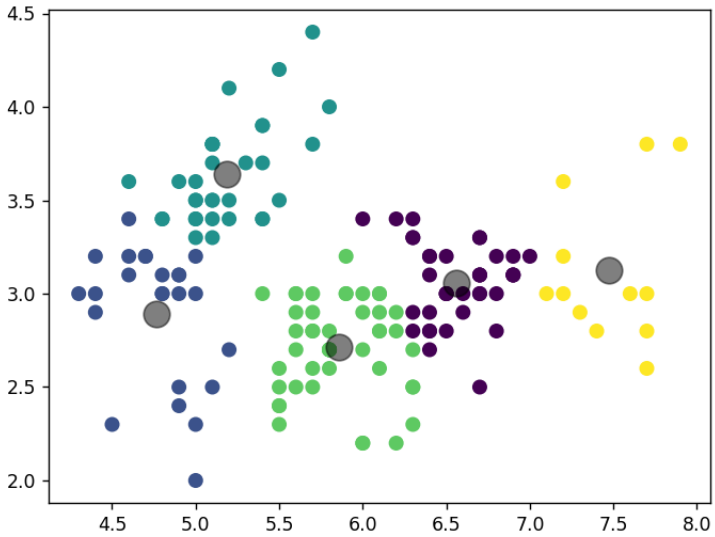


Рис. 8.2. дані розподілені на 5 кластерів за допомогою алгоритму K-Means

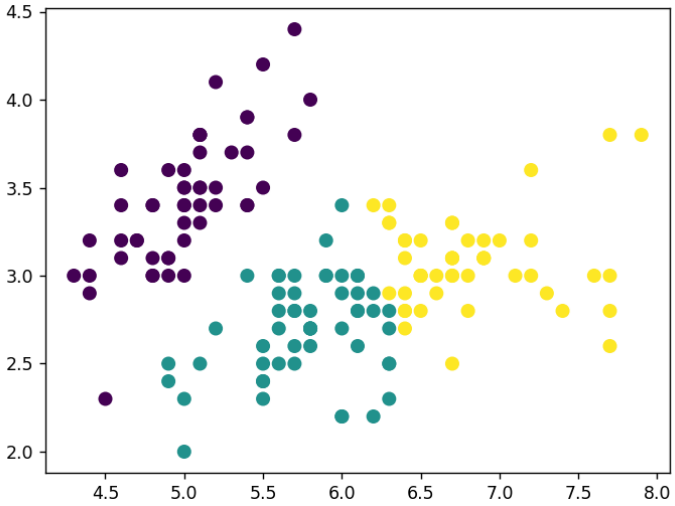


Рис. 8.1. дані розподілені на 3 кластери методом find_clusters()
з параметром rseed = 2

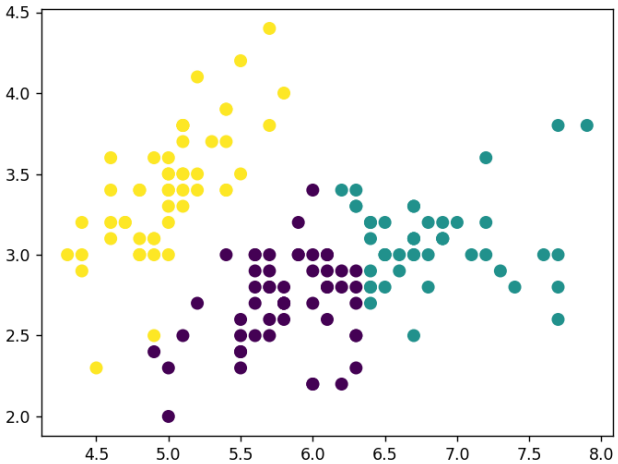


Рис. 8.3. дані розподілені на 3 кластери методом find_clusters()
з параметром rseed = 0

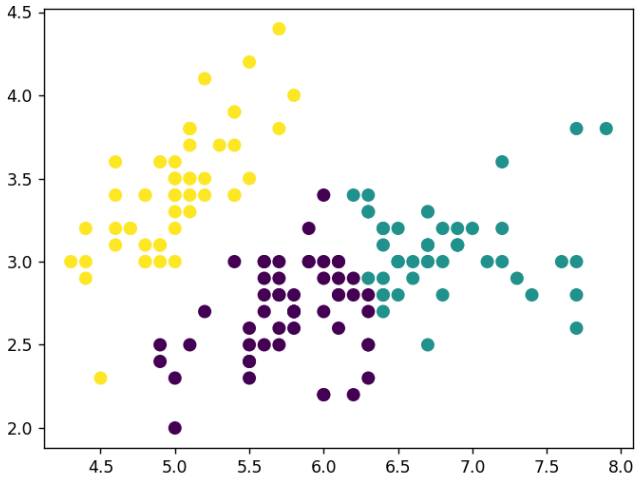


Рис. 8.4. дані розподілені на 3 кластери за допомогою алгоритму K-Means

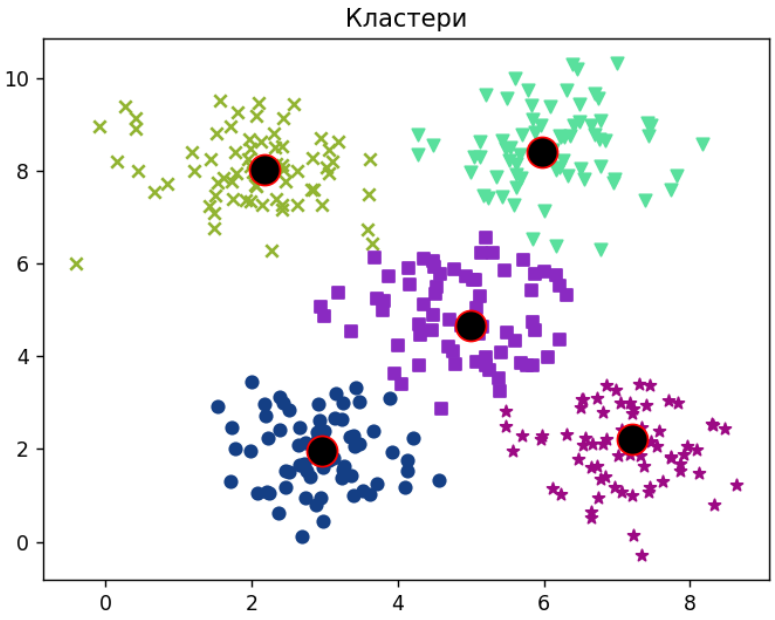
Висновок: Результат кластеризації даних за допомогою алгоритму K-Means з 5 кластерами є більш чітким і зручним для інтерпретації, ніж результат кластеризації даних методом find_clusters() з 3 кластерами. Результат кластеризації даних методом find_clusters() залежить від значення для генератора випадкових 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:




```
Centers of clusters:
 [[2.95568966 1.95775862]
 [7.20690909 2.20836364]
 [2.17603774 8.03283019]
 [5.97960784 8.39078431]
 [4.99466667 4.65844444]]

Number of clusters in input data = 5
```

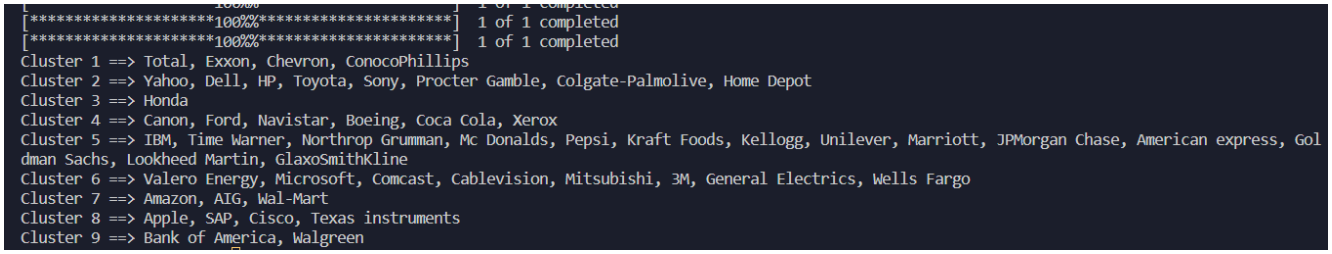
Висновок: Метод зсуву середнього досить точно зміг визначити центри кластерів і визначив 5 кластерів.

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(la-
bels) if label == i]))
```

Результат виконання:



Github: ***https://github.com/mtvi/ipz202_Shcherback_lab3***

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