ЛАБОРАТОРНА РОБОТА № 3

ДОСЛІДЖЕННЯ МЕТОДІВ РЕГРЕСІЇ ТА НЕКОНТРОЬОВАНОГО НАВЧАННЯ

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

Хід роботи:

Завдання 1

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)
    # Завантаження моделі
    with open(output_model_file, 'rb') as f:
        loaded_regressor = pickle.load(f)
    # Виконання передбачень з завантаженою моделлю
    y_test_pred_new = loaded_regressor.predict(X_test)
    # Розрахунок середньої абсолютної похибки
    from sklearn import metrics as sm
    print("\nNew mean absolute error =", round(sm.mean_absolute_error(y_test,
y_test_pred_new), 2))
```

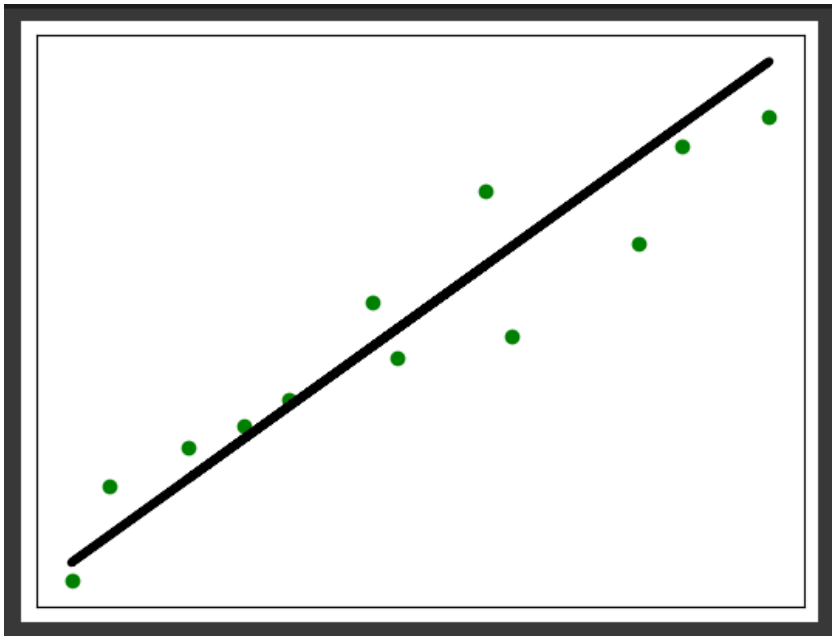


Рис.1. Графік функції. Результат прогнозування.

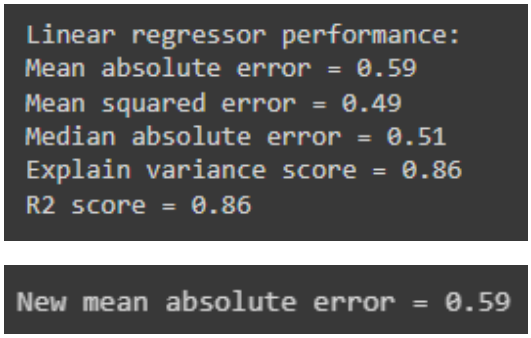


Рис.2. Результат оцінки якості.

Висновки: виконавши дане завдання ми провели прогнозування на тестових даних за допомогою лінійної регресії, після чого зберегли та завантажили отриману модель. Графік показує, що ситуація досить типова для моделі лінійної регресії. Більша частина даних має відносно невелику варіацію і лежить близько до лінії, що означає - модель лінійної регресії добре апроксимує дані частини.

Результати прогнозування показали наступне:

- **Mean Absolute Error**, **Mean Squared Error** та **Median Absolute Error** близькі до 0 означали б дуже точну модель. У нашому ж випадку усі значення відхилень варіюються від 0.49 до 0.59. Це непогані результати, але усе залежить від конкретної задачі та предметної області використання моделі.
- Модель досягла значення **Explained Variance Score** близько 0.86, що вказує на добру здатність пояснити зміни. Тобто модель добре пояснює дисперсію в даних.
- Значення R2 дорівнює близько 0.86. Це означає, що приблизно 86% варіації у цільовій змінній можна пояснити за допомогою даної моделі лінійної регресії.

Після збереження і завантаження моделі, ми знову отримали значення **Mean Absolute Error** близько 0.59, що підтверджує наступне - збережена модель виконується на тестових даних так само, як і оригінальна.

Завдання 2

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_4.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
    with open(output_model_file, 'rb') as f:
        loaded_regressor = pickle.load(f)

    # Виконання передбачень з завантаженою моделлю
    y_test_pred_new = loaded_regressor.predict(X_test.reshape(-1, 1))

    # Розрахунок середньої абсолютної похибки
    print("\nNew mean absolute error =", round(sm.mean_absolute_error(y_test,
y_test_pred_new), 2))
```

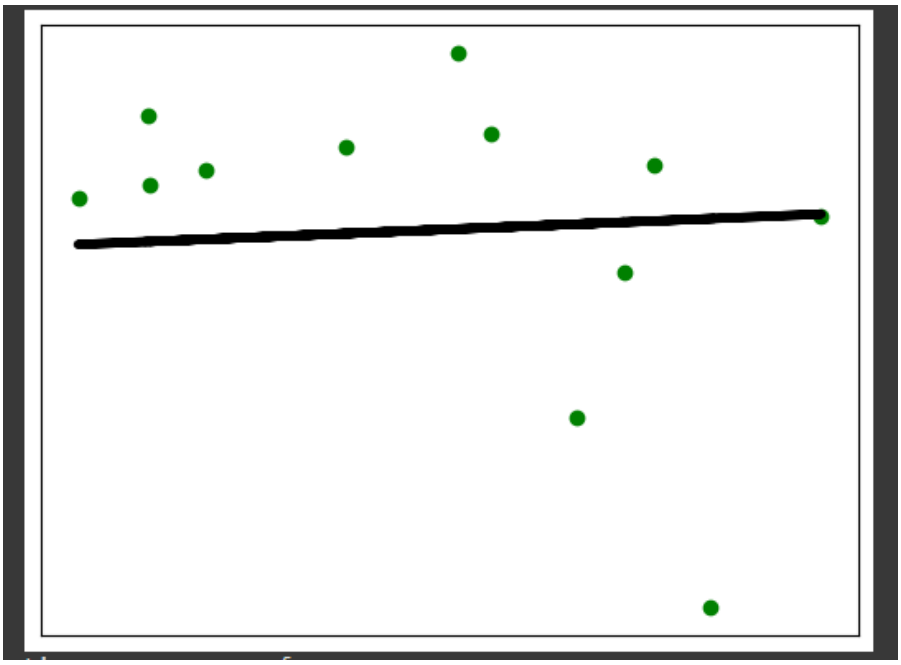


Рис.3. Графік функції. Результат прогнозування.

```
Linear regressor performance:
Mean absolute error = 2.72
Mean squared error = 13.16
Median absolute error = 1.9
Explain variance score = -0.07
R2 score = -0.07

New mean absolute error = 2.72
```

Рис.4. Результат оцінки якості.

Висновки: у порівнянні із завданням 1, результати роботи моделі лінійної регресії для даного набору даних виявились набагато гірше. Графік показує, що більшість даних має велику варіацію і лежить далеко від лінії, це означає - модель лінійної регресії пагано апроксимує дані.

- Показники **Mean Absolute Error** та **Median Absolute Error** збільшились у 4 рази. А **Mean Squared Error** дорівнює 13.16 що вказує на велику середньоквадратичну похибку.
- Показники **Explain Variance Score** і **R2 Score** дорівнюють -0.07. Це означає, що модель не пояснює варіацію у цільовій змінній, і її передбачення гірші, ніж просте середнє значення цільової змінної.

З огляду на це можна вважати, що лінійна регресійна модель, побудована на основі однієї змінної, не є задовільною для цих даних.

Завдання 3

Лістинг програми:

```
import numpy as np
from sklearn import linear_model
import sklearn.metrics as sm
from sklearn.preprocessing import PolynomialFeatures

# Завантаження даних з файлу
data = np.loadtxt('data_multivar_regr.txt', delimiter=',')

# Розбивка даних на навчальний та тестовий набори
X, y = data[:, :-1], data[:, -1]
num_training = int(0.8 * len(X))
X_train, y_train = X[:num_training], y[:num_training]
X_test, y_test = X[num_training:], y[num_training:]

# Створення та навчання моделі лінійної регресії
regressor = linear_model.LinearRegression()
regressor.fit(X_train, y_train)

# Прогноз результату для тестового набору даних
y_test_pred = regressor.predict(X_test)

# Виведення метрик якості лінійної регресії
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("Explained 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))
    # Створення поліноміального регресора ступеня 10 та навчання його на
тренувальних даних
    polynomial = PolynomialFeatures(degree=10)
    X_train_transformed = polynomial.fit_transform(X_train)

    # Вибір вибіркової точки даних і створення поліноміального представлення
    datapoint = [[7.75, 6.35, 5.56]]
    poly_datapoint = polynomial.transform(datapoint)

    # Створення об'єкта лінійного регресора та підгонка до полінома
    poly_linear_model = linear_model.LinearRegression()
    poly_linear_model.fit(X_train_transformed, y_train)

    # Прогноз з використанням лінійного регресора та поліноміального регресора
    linear_prediction = regressor.predict(datapoint)
    poly_prediction = poly_linear_model.predict(poly_datapoint)

    # Виведення результатів
    print("Linear regression prediction:\n", linear_prediction)
    print("Polynomial regression prediction:\n", poly_prediction)

    # Порівняння результатів
    print("\nComparison of predictions:")
    print("Linear regression prediction is", linear_prediction - 41.35, "away
from the actual value.")
    print("Polynomial regression prediction is", poly_prediction - 41.35,
"away from the actual value.")
```

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

Рис.5. Результат оцінки якості лінійної регресії.

```
Linear regression prediction:
 [36.05286276]
Polynomial regression prediction:
 [41.46319764]

Comparison of predictions:
Linear regression prediction is [-5.29713724] away from the actual value.
Polynomial regression prediction is [0.11319764] away from the actual value.
```

Рис.6. Різниця у прогнозах.

Висновки: за результатом порівняння отриманих прогнозів можемо сказати, що прогноз лінійної регресії відхиляється на -5.30 одиниць. Коли в свою чергу прогноз поліноміальної регресії відхиляється лише на 0.11 одиниць і, відповідно, набагато ближчий до фактичного значення. Це свідчить про кращу точність порівняно з лінійною регресією.

Завдання 4

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
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split
# Завантаження даних
diabetes = datasets.load_diabetes()
X = diabetes.data
y = diabetes.target

# Поділіть дані на навчальний та тестовий набори
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.5, ran-
dom_state=0)

# Створення моделі лінійної регресії та навчання
regr = linear_model.LinearRegression()
regr.fit(Xtrain, ytrain)

# Зробіть прогноз по тестовій вибірці
ypred = regr.predict(Xtest)

# Виведіть коефіцієнти регресії та показники
print("Коефіцієнти регресії:")
print(regr.coef_)
print("Перехоплення:")
```

```
print(regr.intercept_)
print("R-squared (R2) Score:", r2_score(ytest, ypred))
print("Mean Absolute Error (MAE):", mean_absolute_error(ytest, ypred))
print("Mean Squared Error (MSE):", mean_squared_error(ytest, ypred))

# Побудова графіку
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()
```

```
Коефіцієнти регресії:
[ -20.4047621  -265.88518066  564.65086437  325.56226865 -692.16120333
  395.55720874   23.49659361  116.36402337  843.94613929   12.71856131]
Перехоплення:
154.35892852801342
R-squared (R2) Score: 0.4377497118254099
Mean Absolute Error (MAE): 44.800645233553276
Mean Squared Error (MSE): 3075.3306886803252
```

Рис.7. Результат оцінки якості лінійної регресії.

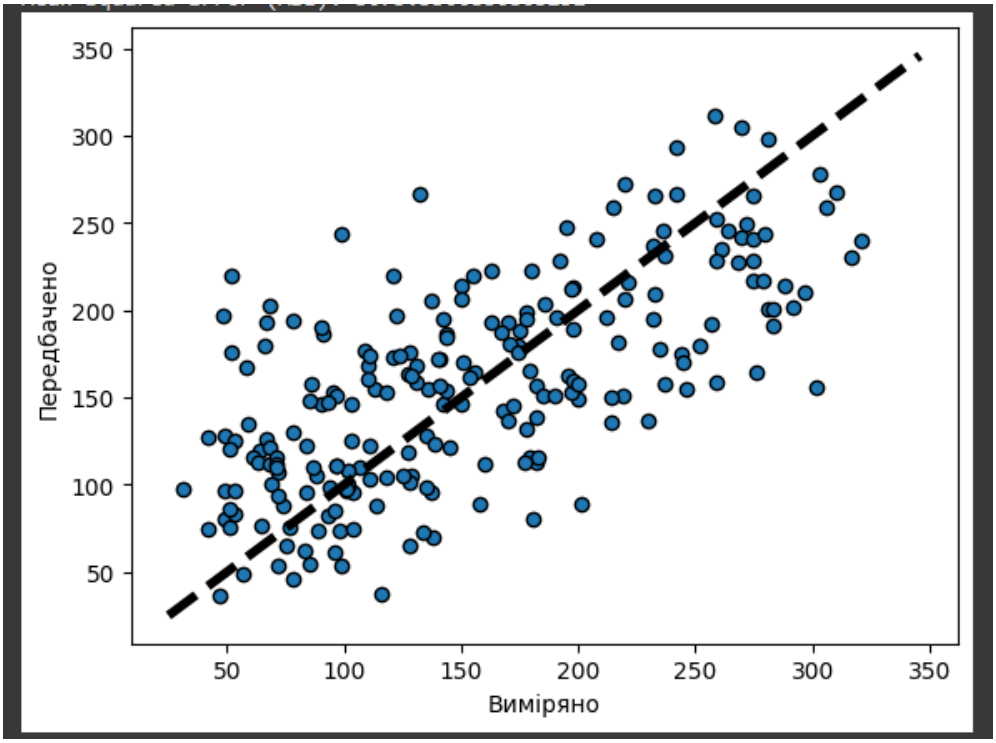


Рис.8. Графік.

Висновки: Модель включає 10 **коефіцієнтів регресії**, які вказують на вплив кожного вхідного параметра на цільову змінну. Наприклад, деякі параметри можуть мати позитивний вплив, тоді як інші - негативний. **Перехоплення** дорівнює

приблизно 154.36. Це означає: якщо всі вхідні параметри (коефіцієнти) дорівнюють нулю - очікуване значення цільової змінної дорівнює 154.36. Значення **R2 Score** дорівнює близько 0.44. Тобто модель пояснює лише 44% варіації у цільовій змінній, що не дуже добре підходить для передбачення даних. **Mean Absolute Error** дорівнює близько 44.80. Дане значення вказує на те, що модель має велику середню похибку в прогнозах.

Отриманий графік також показує розкид даних. Тож як висновок можна скзати, що лінійна регресійна модель, побудована для даних діабету, має обмежену здатність пояснювати залежності в даних.

Завдання 5

Лістинг програми:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

# Генеруємо випадкові дані
m = 100
X = 6 * np.random.rand(m, 1) - 5
y = 0.7 * X**2 + X + 3 + np.random.randn(m, 1)

# Побудова моделі лінійної регресії
lin_reg = LinearRegression()
lin_reg.fit(X, y)

# Побудова моделі поліноміальної регресії
poly_features = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly_features.fit_transform(X)
poly_reg = LinearRegression()
poly_reg.fit(X_poly, y)

# Виведення коефіцієнтів лінійної регресії
print("Лінійна регресія:")
print("Перехоплення:", lin_reg.intercept_)
print("Коефіцієнт регресії:", lin_reg.coef_)

# Виведення коефіцієнтів поліноміальної регресії
print("Поліноміальна регресія:")
print("Перехоплення:", poly_reg.intercept_)
print("Коефіцієнти регресії:", poly_reg.coef_)
```

```
      # Виведення даних на графіку
      plt.figure(figsize=(12, 6))

      plt.subplot(1, 2, 1)
      plt.scatter(X, y, c='b', label='Дані')
      plt.plot(X, lin_reg.predict(X), 'r-', linewidth=2, label='Лінійна
регресія')
      plt.xlabel('X')
      plt.ylabel('Y')
      plt.legend(loc='upper left')
      plt.title('Лінійна регресія')

      plt.subplot(1, 2, 2)
      X_new = np.linspace(-5, 1, 100).reshape(100, 1)
      X_new_poly = poly_features.transform(X_new)
      y_new = poly_reg.predict(X_new_poly)
      plt.scatter(X, y, c='b', label='Дані')
      plt.plot(X_new, y_new, 'r-', linewidth=2, label='Поліноміальна регресія')
      plt.xlabel('X')
      plt.ylabel('Y')
      plt.legend(loc='upper left')
      plt.title('Поліноміальна регресія')

      plt.show()
```

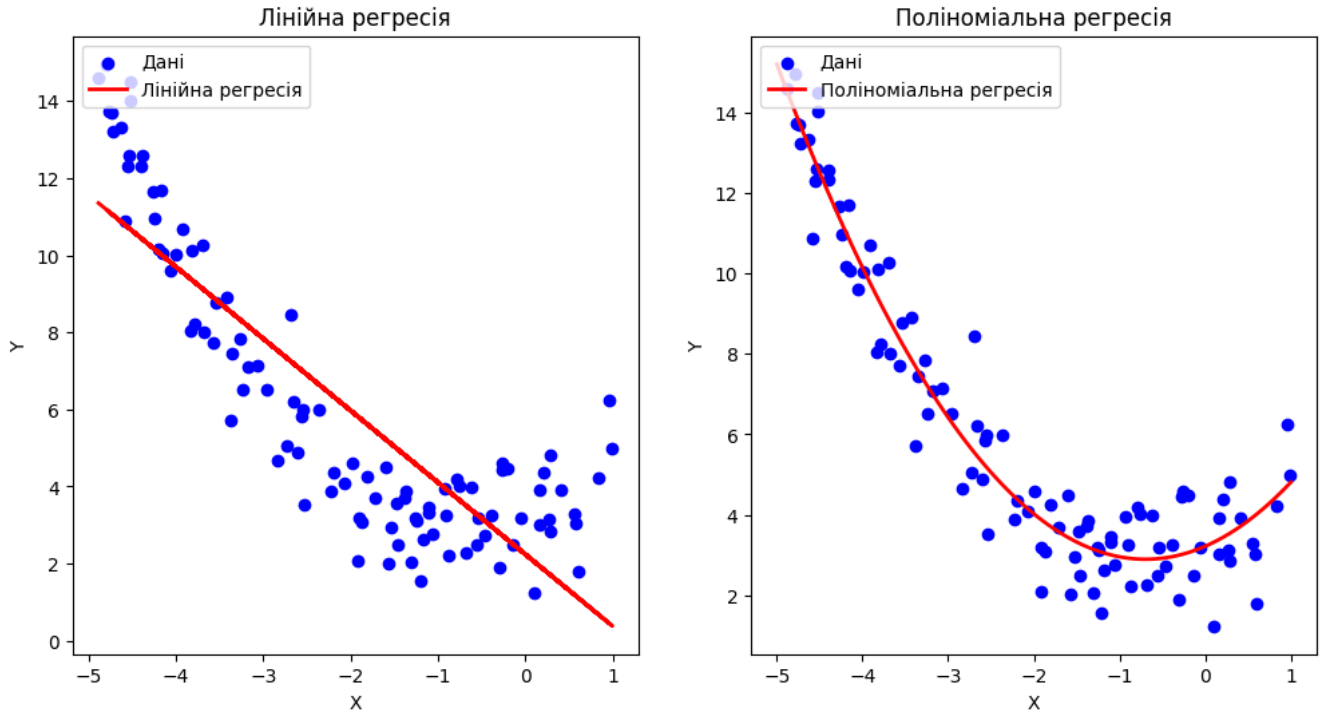


Рис.9. Графіки.

```
Лінійна регресія:
Перехоплення: [2.23660845]
Коефіцієнт регресії: [[-1.86568912]]
Поліноміальна регресія:
Перехоплення: [3.21822415]
Коефіцієнти регресії: [[0.92687194 0.66478908]]
```

Рис.10. Перехоплення та коефіцієнти регресії.

Модель 4 варіанта у вигляді математичного рівняння:

$y=0.7x^2+1x+3+гауссовий\ шум.$

Отримана модель регресії з передбаченими коефіцієнтами:

$y=0.93x^2+0.66x+3.22$

Висновки: для лінійної моделі коєфіцієнт перед *x* дуже близький до -1, що означає - ця модель практично відсутня. Поліноміальна модель, з іншого боку, близька до опису вихідних даних і має коефіцієнти перед x^2 (за варіантом 0.7, отримано 0.93) та *x* (за варіантом 1, отримано 0.66) , які вказують на квадратичний зв'язок між змінною *x* і змінною *y*. Отже поліноміальна регресія більше підходить для апроксимації цього набору даних.

Завдання 6

Лістинг програми:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

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")
        plt.xlabel("Training set size")
        plt.ylabel("RMSE")
        plt.legend()

    # Генеруємо випадкові дані, як у варіанті
    m = 100
    X = 6 * np.random.rand(m, 1) - 5
    y = 0.7 * X ** 2 + X + 3 + np.random.randn(m, 1)

    # Створення та побудова кривих навчання для лінійної регресії
    lin_reg = LinearRegression()
    plot_learning_curves(lin_reg, X, y)
    plt.show()

    from sklearn.pipeline import Pipeline
    from sklearn.preprocessing import PolynomialFeatures

    # Створення та побудова кривих навчання для поліноміальної регресії 10-го
ступеня
    polynomial_regression = Pipeline([
        ("poly features", PolynomialFeatures(degree=10, include_bias=False)),
        ("lin_reg", LinearRegression())
    ])

    plot_learning_curves(polynomial_regression, X, y)
    plt.show()
```

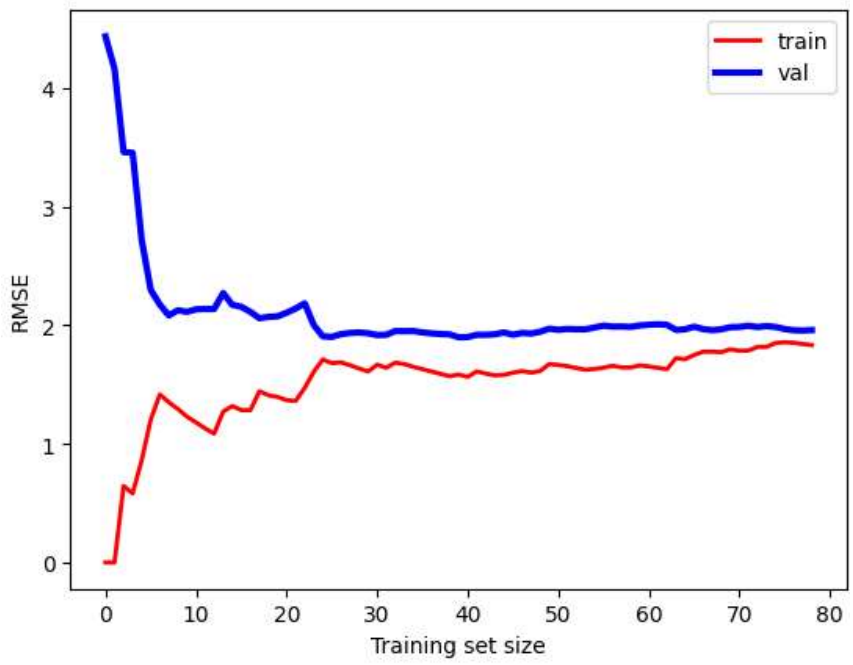


Рис.11.Криві навчання для лінійної моделі.

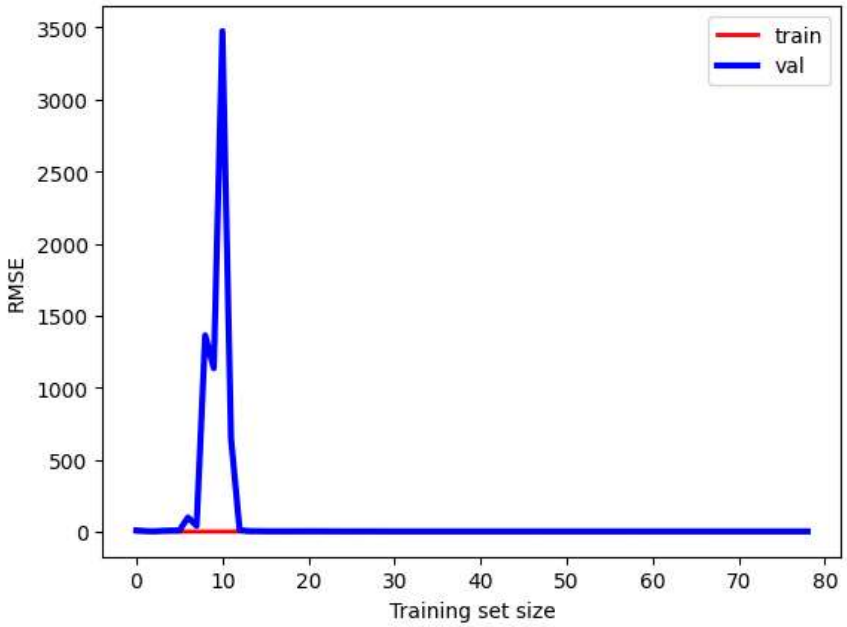


Рис.12. Криві навчання для поліноміальної моделі.

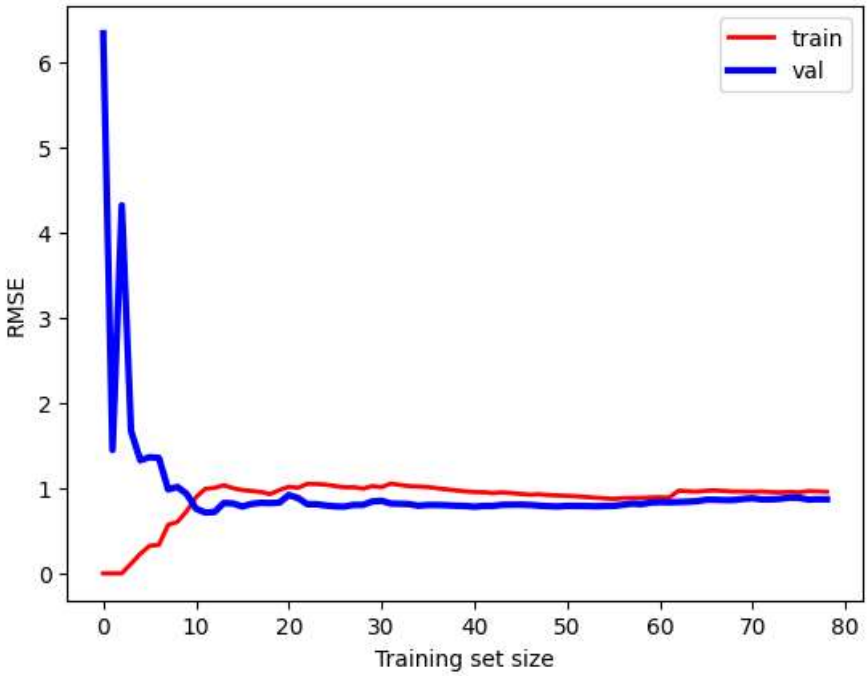


Рис.13. Криві навчання поліноміальної моделі 2-го ступеня.

Завдання 7

Лістинг програми:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import metrics
# Завантаження вхідних даних із файлу
X = np.loadtxt('data_clustering.txt', delimiter=',')
```

```
      # Кількість кластерів
      num_clusters = 5

      # Візуалізація вхідних даних
      plt.figure()
      plt.scatter(X[:, 0], X[:, 1], marker='o', facecolors='none', edgecol-
ors='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('Розподіл даних')
      plt.xlim(x_min, x_max)
      plt.ylim(y_min, y_max)
      plt.xticks(())
      plt.yticks(())

      # Створення та навчання моделі k-середніх
      kmeans = KMeans(n_clusters=num_clusters)
      kmeans.fit(X)

      # Побудова центроїд кластерів
      centroids = kmeans.cluster_centers_
      plt.scatter(centroids[:, 0], centroids[:, 1], s=200, marker='x', col-
or='red', label='Центроїди')

      # Вивід результату на графіку
      plt.legend()
      plt.show()

      # Виведення інформації про кластеризацію
      labels = kmeans.labels_
      silhouette_score = metrics.silhouette_score(X, labels, met-
ric='euclidean')
      print(f"Силуетний коефіцієнт: {silhouette_score}")

      import numpy as np
      import matplotlib.pyplot as plt
      from sklearn.cluster import KMeans

      # Завантаження вхідних даних із файлу
      X = np.loadtxt('data_clustering.txt', delimiter=',')

      # Кількість кластерів
      num_clusters = 5

      # Створення об'єкту KMeans
      kmeans = KMeans(init='k-means++', n_clusters=num_clusters, n_init=10)

      # Навчання моделі кластеризації KMeans
      kmeans.fit(X)
```

```
    # Визначення кроку сітки
    step_size = 0.01

    # Визначення меж сітки
    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))

    # Передбачення вихідних міток для всіх точок сітки
    output = kmeans.predict(np.c_[x_vals.ravel(), y_vals.ravel()])

    # Відображення областей та виділення їх кольором
    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, origin="lower")

    # Відображення вхідних точок
    plt.scatter(X[:, 0], X[:, 1], marker='o', facecolors="none", edgecol-
ors="black", s=80)

    # Відображення центрів кластерів
    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')

    plt.title('Кластеризація даних')
    plt.xlim(x_min, x_max)
    plt.ylim(y_min, y_max)
    plt.xticks(())
    plt.yticks(())
    plt.show()
```

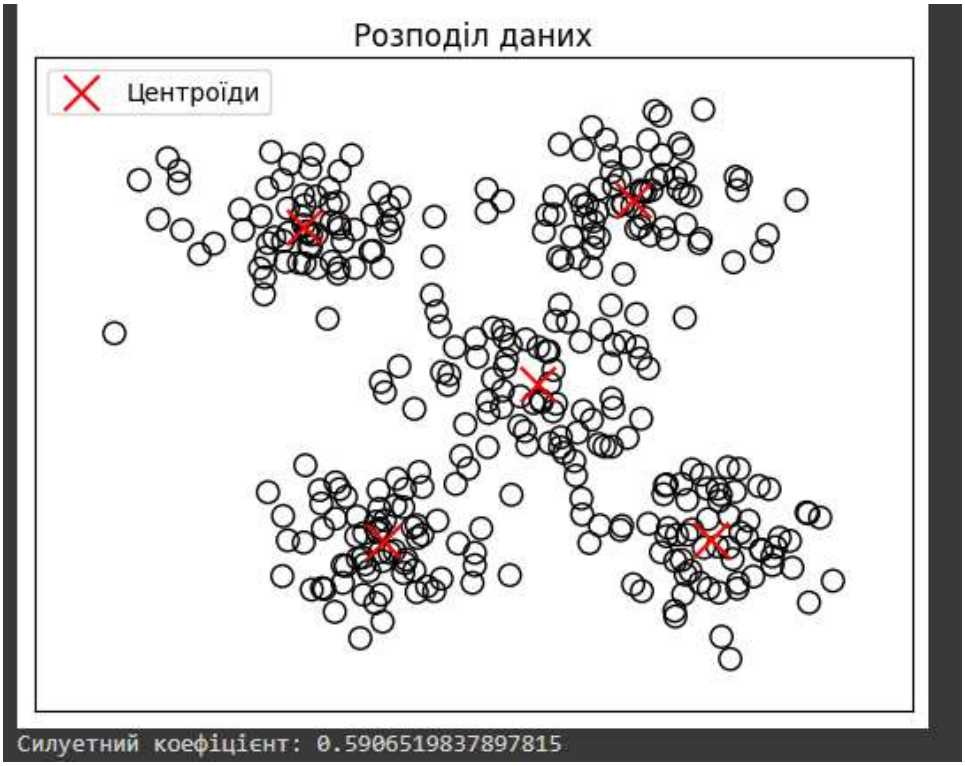


Рис.14. Графік результату і силуетний коефіцієнт.

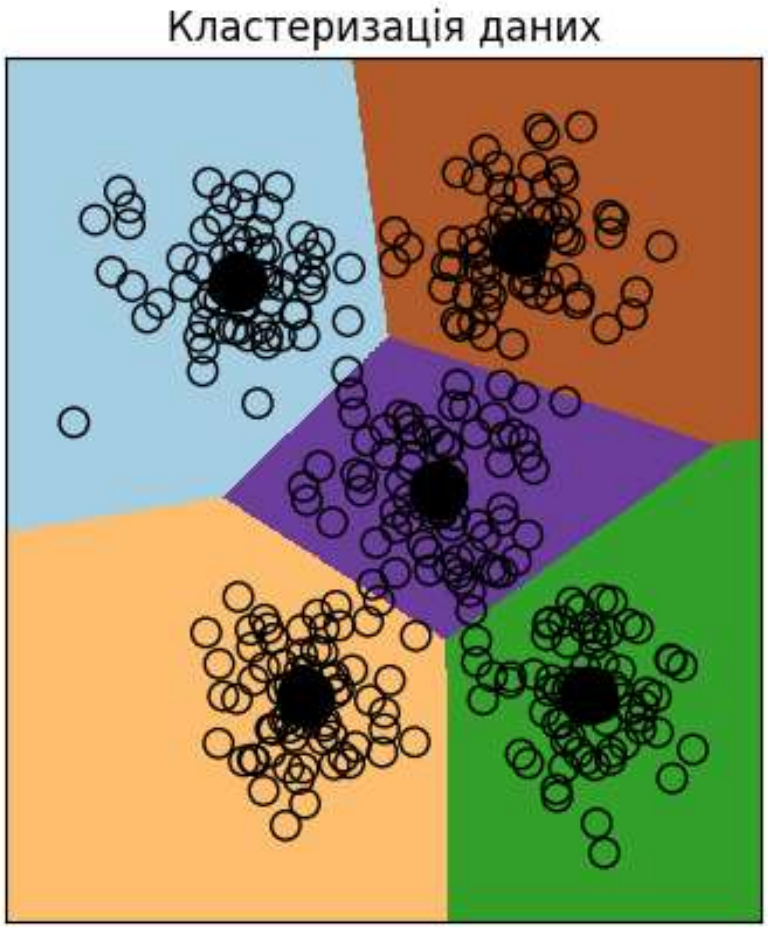


Рис.15. Графік з результатом кластеризації даних.

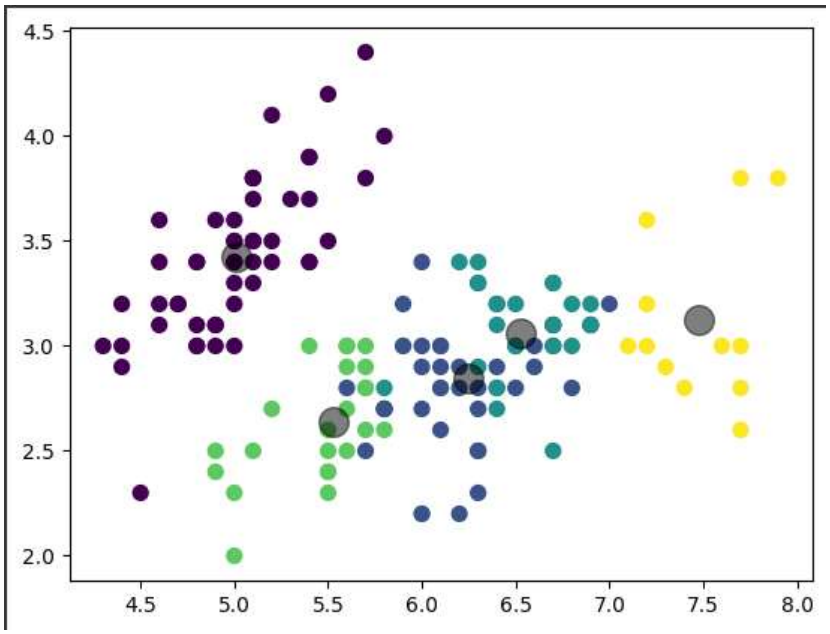
Висновки: Силуетний коефіцієнт в значенні близько 0.59 свідчить про досить високу якість кластеризації за методом k-середніх. Об'єкти в кожному кластері розташовані досить близько один до одного і більшість із них віддалені від об'єктів інших кластерів (хоча і не усі). На графіку видно 5 кластерів, алгоритм k-середніх знайшов відмінні кластери і вдало визначив їхні центроїди.

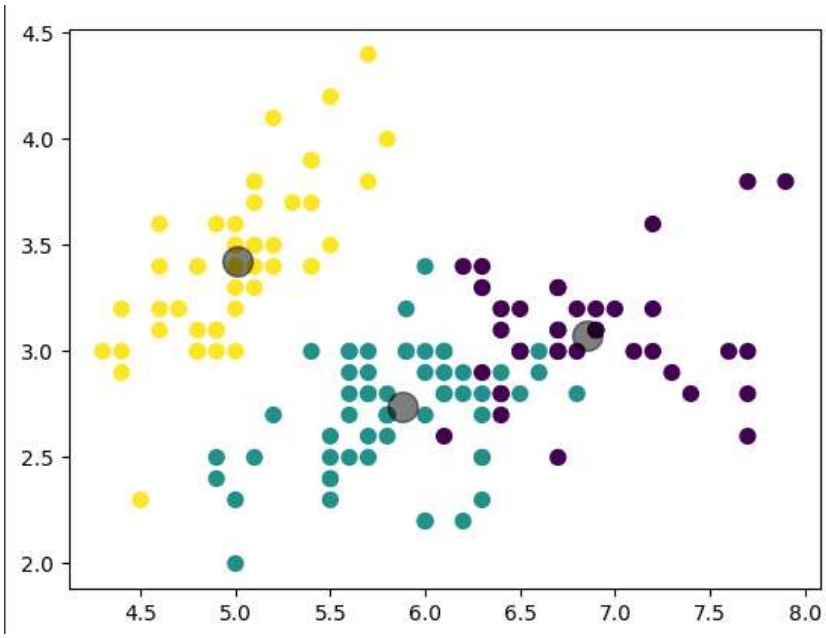
Завдання 8

Лістинг програми:

```
    # Імпортуємо необхідні бібліотеки
    from sklearn.cluster import KMeans
    import numpy as np
    from sklearn.metrics import pairwise_distances_argmin
    import matplotlib.pyplot as plt
    from sklearn.datasets import load_iris

    # Завантажуємо датасет iris
    iris = load_iris()
    X = iris.data  # Ознаки
    y = iris.target  # Мітки класів

    # Ініціалізуємо модель k-середніх з 5 кластерами
    kmeans = KMeans(n_clusters=5)
    kmeans.fit(X)  # Навчання моделі

    # Передбачаємо кластери для вхідних даних
    y_kmeans = kmeans.predict(X)

    # Візуалізація результатів
    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()

    # Функція для знаходження кластерів, використовуючи pair-
wise_distances_argmin
    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

    # відображення кластерів
    centers, labels = find_clusters(X, 3, rseed=0)
    plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')
    plt.show()
```






```
from sklearn.metrics import silhouette_score
score = silhouette_score(X, y_kmeans)
print("Силуетний коефіцієнт:", score)
```

```
Силуетний коефіцієнт: 0.4930804067193521
```

Рис.16. Результат кластеризації.

Висновки: На першому графіку показана кластеризація датасету iris за допомогою методу **k-середніх** на 5 кластерів. Кожен кластер має свою відмінну мітку кольору, і центри кластерів позначені чорним кольором. На другому графіку використовується ваша функція **find_clusters** для поділу даних на 3 кластери.

Можна побачити, що розміщення даних не змінилось, єдина відмінність – поділ цих даних, 4 менших кластери об'єднались у два більших. Силуетний коефіцієнт дорівнює приблизно 0.493, що свідчить про те, що кластеризація даних методом k-середніх має помірний рівень якості.

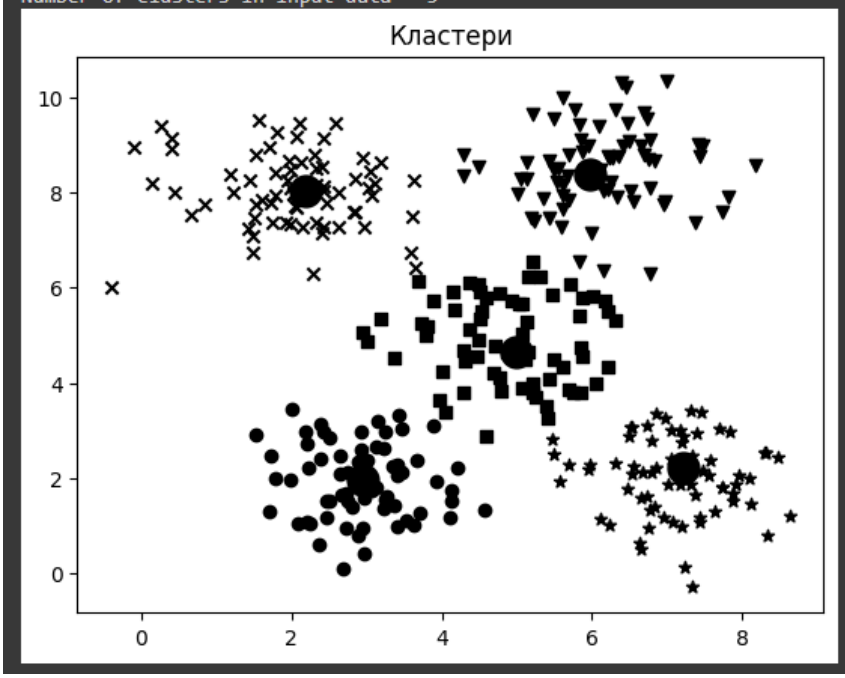
Завдання 9

Лістинг програми:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import MeanShift, estimate_bandwidth
from itertools import cycle

# Завантаження вхідних 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):
    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', marker-
facecolor='black', markeredgecolor='black',
                 markersize=15)

    plt.title('Кластери')
    plt.show()
```

```
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
```




```
from sklearn.metrics import silhouette_score

silhouette_avg = silhouette_score(X, labels)
print("Silhouette Score:", silhouette_avg)
```

```
Silhouette Score: 0.586627762532276
```

Рис.17. Результат кластеризації.

Висновки: порівнюючи результат кластеризації MeanShift з k-середніх із 7 завдання можемо побачити, що візуально результати досить ідентичні. Маємо 5 кластерів, із визначеними центроїдами. Значення силуетного коефіцієнта дорів-

нює 0.587, що свідчить про те, що кластеризація даних за допомогою алгоритму зсуву середньою (Mean Shift) виявилася досить ефективною. Різниця між ефективністю 2-ох алгоритмів не дуже велика 0.01.