**Лабораторна робота 3**

**ДОСЛІДЖЕННЯ МЕТОДІВ РЕГРЕСІЇ ТА НЕКОНТРОЬОВАНОГО НАВЧАННЯ**

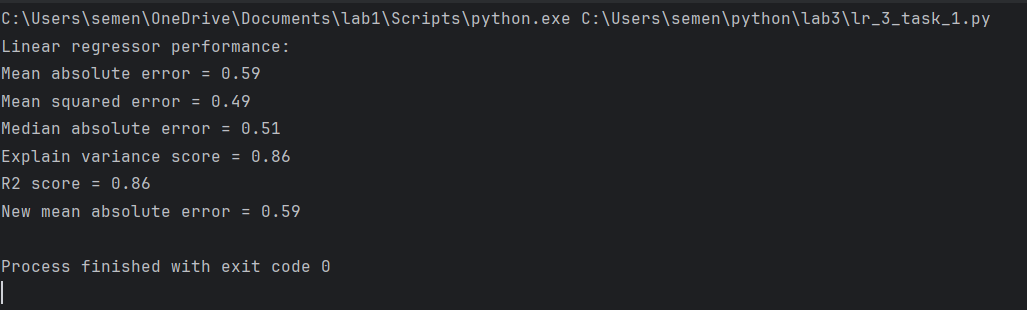
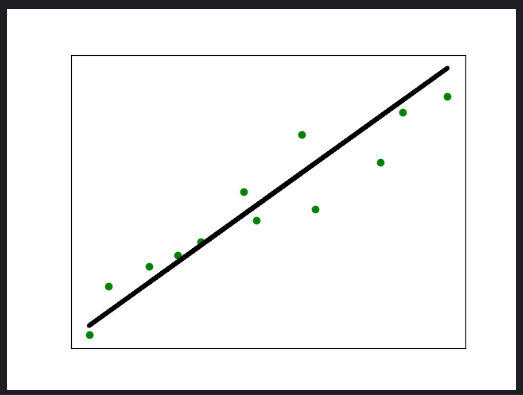
Завдання 2.1. Створення регресора однієї змінної

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\_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:  
 regressor\_model = pickle.load(f)  
  
y\_test\_pred\_new = regressor\_model.predict(X\_test)  
print("New mean absolute error =", round(sm.mean\_absolute\_error(y\_test, y\_test\_pred\_new), 2))

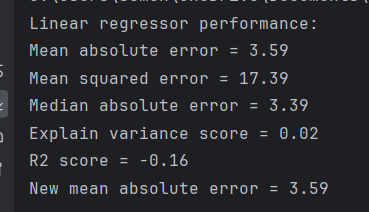
Результат:

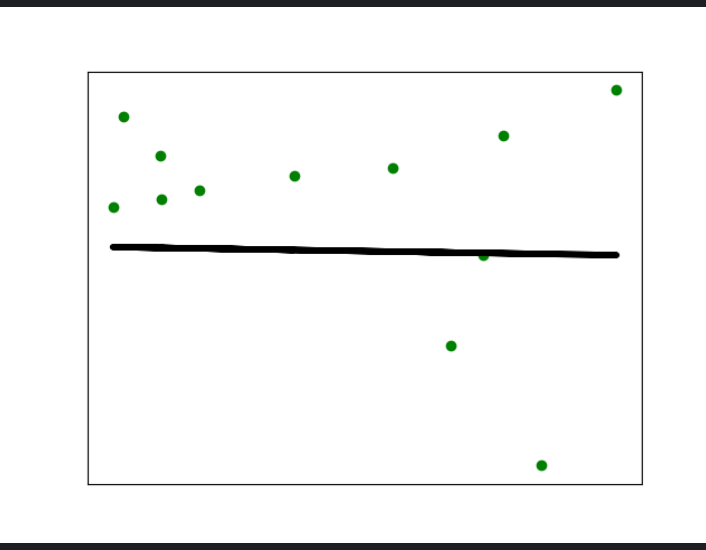
 

Завдання 2.2. Передбачення за допомогою регресії однієї змінної

# Варіант 3 (за списком 13)  
  
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\_3.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\_task2.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:  
 regressor\_model = pickle.load(f)  
  
y\_test\_pred\_new = regressor\_model.predict(X\_test)  
print("New mean absolute error =", round(sm.mean\_absolute\_error(y\_test, y\_test\_pred\_new), 2))

Результат:



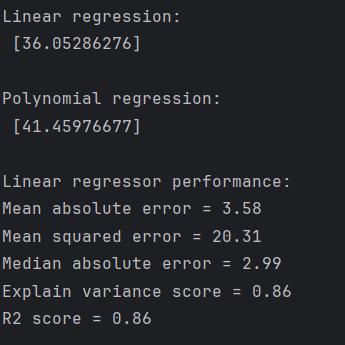


Завдання 2.3. Створення багатовимірного регресора

Лістинг:

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 = 'lab3/data/input/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:]  
  
# 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)  
  
# Створення об'єкта полілінійного регресора  
poly\_linear\_model = linear\_model.LinearRegression()  
poly\_linear\_model.fit(X\_train\_transformed, y\_train)  
  
# Створення об'єкта лінійного регресора  
linear\_regressor = linear\_model.LinearRegression()  
linear\_regressor.fit(X\_train, y\_train)  
  
print("\nLinear regression:\n",  
linear\_regressor.predict(datapoint))  
print("\nPolynomial regression:\n",  
poly\_linear\_model.predict(poly\_datapoint))  
  
# Прогнозування результату  
y\_test\_pred = linear\_regressor.predict(X\_test)  
  
print("\nLinear 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))

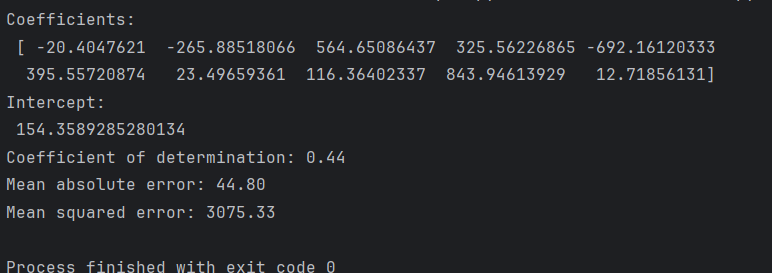
Результат:

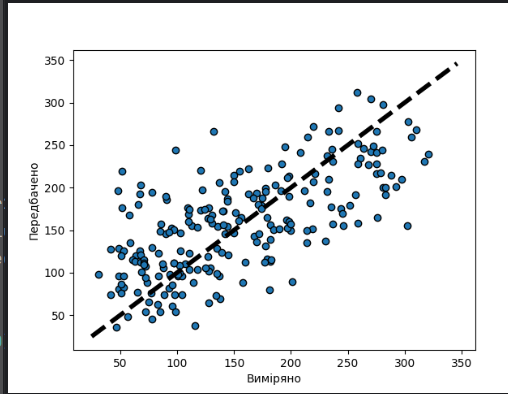


**Завдання 2.4.** Розробіть лінійний регресор, використовуючи набір даних по діабету, який існує в sklearn.datasets.

import matplotlib.pyplot as plt  
import numpy as np  
from sklearn import datasets, linear\_model  
from sklearn.metrics import mean\_squared\_error, r2\_score  
from sklearn.metrics import mean\_absolute\_error  
from sklearn.model\_selection import train\_test\_split  
  
diabetes = datasets.load\_diabetes()  
X = diabetes.data  
y = diabetes.target  
  
Xtrain, Xtest, ytrain, ytest = train\_test\_split(X, y, test\_size= 0.5, random\_state = 0)  
  
regr = linear\_model.LinearRegression()  
regr.fit(Xtrain, ytrain)  
ypred = regr.predict(Xtest)  
  
print('Coefficients: \n', regr.coef\_)  
print('Intercept: \n', regr.intercept\_)  
print('Coefficient of determination: %.2f' % r2\_score(ytest, ypred))  
print('Mean absolute error: %.2f' % mean\_absolute\_error(ytest, ypred))  
print('Mean squared error: %.2f' % mean\_squared\_error(ytest, ypred))  
  
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()

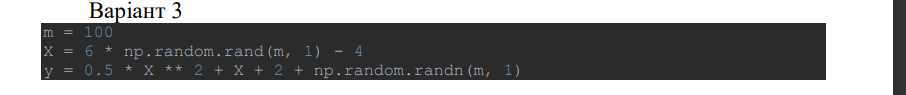
Результат:





Завдання 2.5. Самостійна побудова регресії

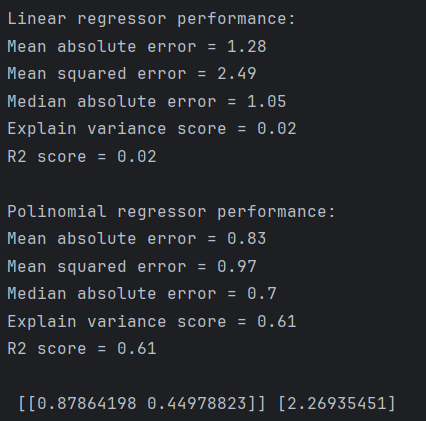


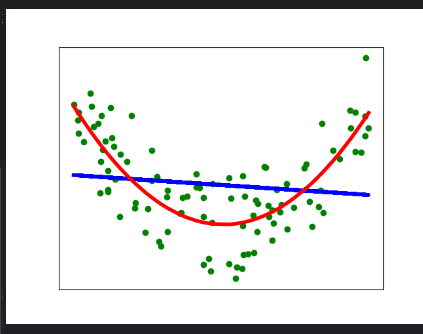


Лістинг:

# Варіант 3 (за списком 13)  
  
import numpy as np  
from sklearn import linear\_model  
import sklearn.metrics as sm  
import matplotlib.pyplot as plt  
from sklearn.preprocessing import PolynomialFeatures  
  
np.random.seed(0)  
m = 100  
X = 6 \* np.random.rand(m, 1) - 4  
y = 0.5 \* X \*\* 2 + X + 2 + np.random.randn(m, 1)  
  
# Побудова графіка  
plt.scatter(X, y, color='green', label='data')  
  
# Лінійна регресія  
linear\_regression = linear\_model.LinearRegression()  
linear\_regression.fit(X, y)  
y\_linear\_pred = linear\_regression.predict(X)  
  
# Поліноміальна регресія  
polynomial\_features = PolynomialFeatures(degree=2, include\_bias=False)  
X\_poly = polynomial\_features.fit\_transform(X)  
  
linear\_regression\_poly = linear\_model.LinearRegression()  
linear\_regression\_poly.fit(X\_poly, y)  
  
y\_poly\_pred = linear\_regression\_poly.predict(X\_poly)  
  
# Побудова графіка  
plt.plot(X, y\_linear\_pred, color='blue', label='linear', linewidth=4)  
sort\_indices = np.argsort(X[:, 0])  
X\_sorted = X[sort\_indices]  
y\_poly\_pred\_sorted = y\_poly\_pred[sort\_indices]  
plt.plot(X\_sorted, y\_poly\_pred\_sorted, color='red', label='polynomial', linewidth=4)  
plt.xticks(())  
plt.yticks(())  
plt.show()  
  
print("Linear regressor performance:")  
print("Mean absolute error =",  
round(sm.mean\_absolute\_error(y, y\_linear\_pred), 2))  
print("Mean squared error =",  
round(sm.mean\_squared\_error(y, y\_linear\_pred), 2))  
print("Median absolute error =",  
round(sm.median\_absolute\_error(y, y\_linear\_pred), 2))  
print("Explain variance score =", round(sm.explained\_variance\_score(y, y\_linear\_pred), 2))  
print("R2 score =", round(sm.r2\_score(y, y\_linear\_pred), 2))  
  
print("\nPolinomial regressor performance:")  
print("Mean absolute error =",  
round(sm.mean\_absolute\_error(y, y\_poly\_pred), 2))  
print("Mean squared error =",  
round(sm.mean\_squared\_error(y, y\_poly\_pred), 2))  
print("Median absolute error =",  
round(sm.median\_absolute\_error(y, y\_poly\_pred), 2))  
print("Explain variance score =", round(sm.explained\_variance\_score(y, y\_poly\_pred), 2))  
print("R2 score =", round(sm.r2\_score(y, y\_poly\_pred), 2))  
  
print('\n', linear\_regression\_poly.coef\_, linear\_regression\_poly.intercept\_)

Результат:





y = 0.5 \* x2 + x + 2

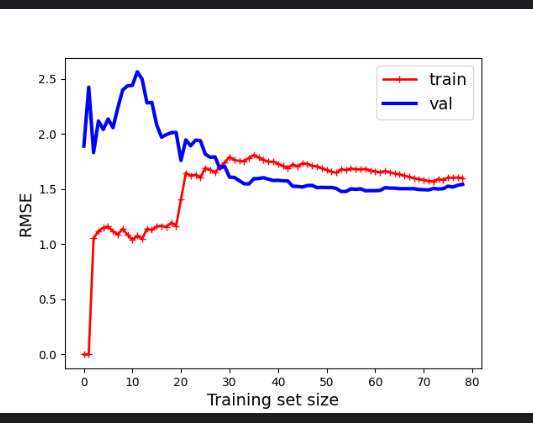
y = 0.45 \* x2 + 0.87 \* x + 2.27

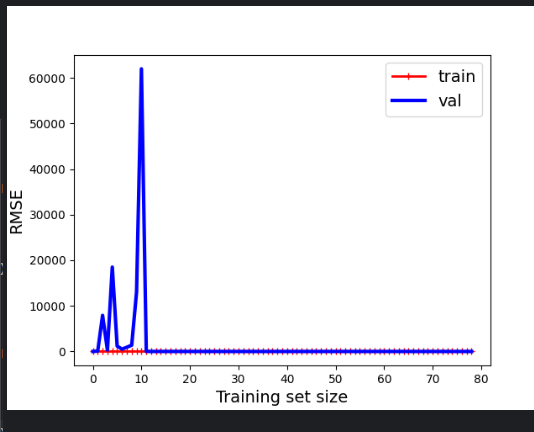
По графіку та показникам ефективності можу бачити, що модель поліноміальної регресії набагато краще справляється з нелінійними даними, бо вона може працювати з кривою формою даних, отриманих в цьому випадку.

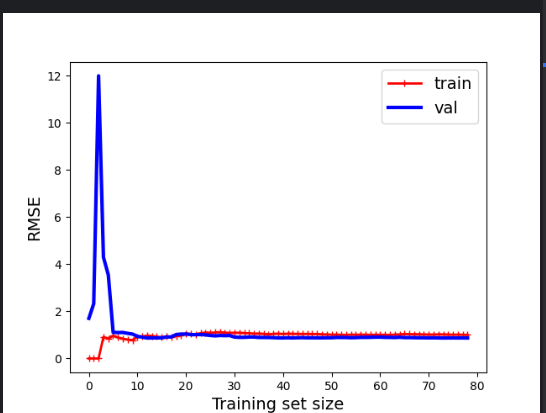
Завдання 2.6. Побудова кривих навчання

from matplotlib import pyplot as plt  
import numpy as np  
from sklearn.linear\_model import LinearRegression  
from sklearn.metrics import mean\_squared\_error  
from sklearn.model\_selection import train\_test\_split  
from sklearn.pipeline import Pipeline  
from sklearn.preprocessing import PolynomialFeatures  
  
np.random.seed(0)  
m = 100  
X = 6 \* np.random.rand(m, 1) - 4  
y = 0.5 \* X \*\* 2 + X + 2 + np.random.randn(m, 1)  
  
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.legend(loc = 'upper right', fontsize = 14)  
 plt.xlabel('Training set size', fontsize = 14)  
 plt.ylabel('RMSE', fontsize = 14)  
 plt.show()  
   
lin\_reg = LinearRegression()  
plot\_learning\_curves(lin\_reg, X, y)  
  
polynomial\_regression = Pipeline([  
 ('poly\_features', PolynomialFeatures(degree = 10, include\_bias = False)),  
 ('lin\_reg', LinearRegression()),  
])  
plot\_learning\_curves(polynomial\_regression, X, y)  
  
polynomial\_regression = Pipeline([  
 ('poly\_features', PolynomialFeatures(degree = 2, include\_bias = False)),  
 ('lin\_reg', LinearRegression()),  
])  
plot\_learning\_curves(polynomial\_regression, X, y)

Результат:



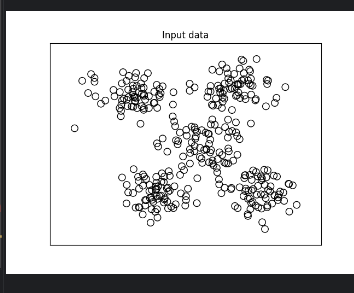


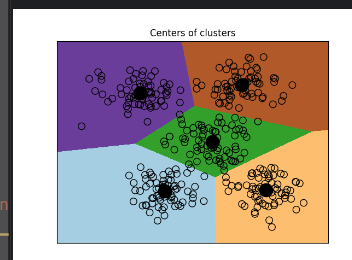


Завдання 2.7. Кластеризація даних за допомогою методу k-середніх

import numpy as np  
from sklearn. cluster import KMeans  
from matplotlib import pyplot as plt  
  
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('Input data')  
plt.xlim(x\_min, x\_max)  
plt.ylim(y\_min, y\_max)  
plt.xticks(())  
plt.yticks(())  
plt.show()  
  
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\_values, y\_values = np.meshgrid(np.arange(x\_min, x\_max, step\_size), np.arange(y\_min, y\_max, step\_size))  
predicted\_labels = kmeans.predict(np.c\_[x\_values.ravel(), y\_values.ravel()])  
predicted\_labels = predicted\_labels.reshape(x\_values.shape)  
plt.figure()  
plt.clf()  
plt.imshow(predicted\_labels, interpolation='nearest', extent=(x\_values.min(), x\_values.max(), y\_values.min(), y\_values.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')  
plt.title('Centers of clusters')  
plt.xlim(x\_min, x\_max)  
plt.ylim(y\_min, y\_max)  
plt.xticks(())  
plt.yticks(())  
plt.show()

Результат:



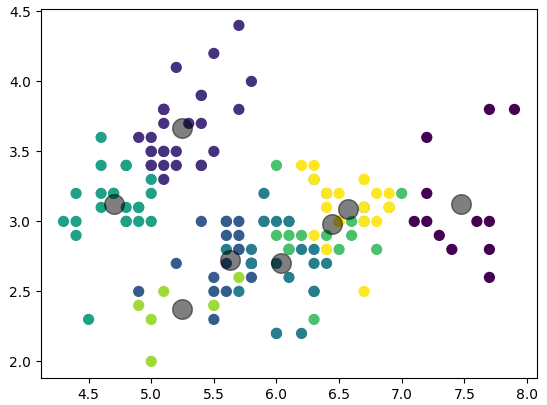


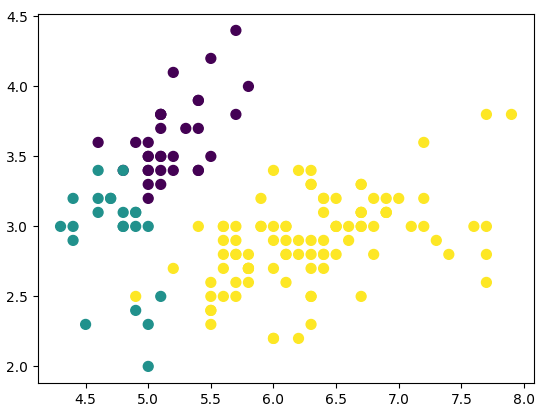
Модель к-середніх класифікує дані за схожими ознаками та намагається розділити їх на кластери. На графіку видно 5 кластерів та їх центри.

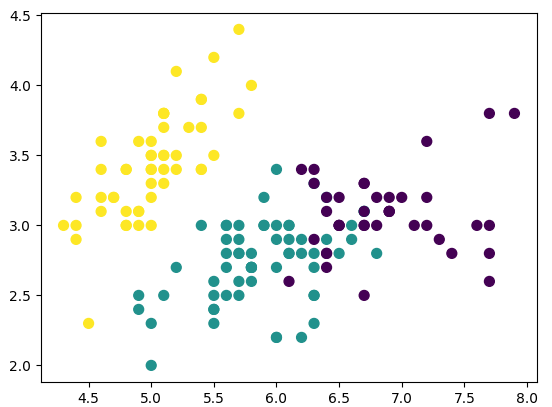
Завдання 2.8. Кластеризація K-середніх для набору даних Iris

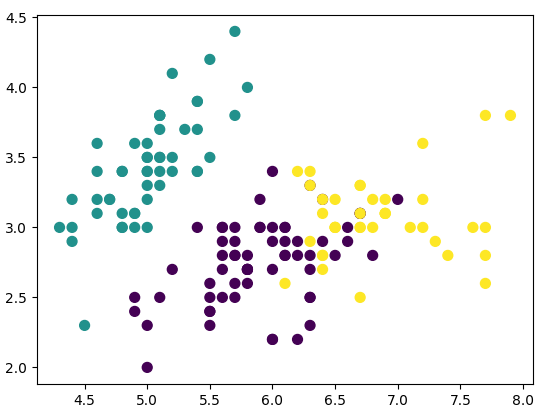
Лістинг:

from matplotlib import pyplot as plt  
from sklearn import datasets  
from sklearn.cluster import KMeans  
from sklearn.svm import SVC  
from sklearn.metrics import pairwise\_distances\_argmin  
import numpy as np  
  
iris = datasets.load\_iris()  
X = iris['data']  
y = iris['target']  
  
# K-means clustering  
kmeans = KMeans(algorithm='auto').fit(X)  
kmeans.fit(X)  
y\_kmeans = kmeans.predict(X)  
  
# Plot the decision boundary  
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()  
  
  
def find\_clusters(X, n\_clusters, rseed=2):  
 # Randomly choose clusters  
 rng = np.random.RandomState(rseed)  
 i = rng.permutation(X.shape[0])[:n\_clusters]  
 centers = X[i]  
  
 while True:  
 # Assign labels based on closest center  
 labels = pairwise\_distances\_argmin(X, centers)  
  
 # Find new centers from means of points  
 new\_centers = np.array([X[labels == i].mean(0) for i in range(n\_clusters)])  
  
 # Check for convergence  
 if np.all(centers == new\_centers):  
 break  
 centers = new\_centers  
 return centers, labels  
  
  
centers, labels = find\_clusters(X, 3)  
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')  
plt.show()  
  
centers, labels = find\_clusters(X, 3, rseed=0)  
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')  
plt.show()  
  
labels = KMeans(3, random\_state=0).fit\_predict(X)  
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')  
plt.show()

Результат: 





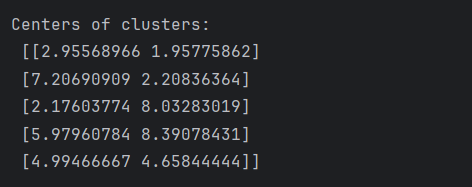


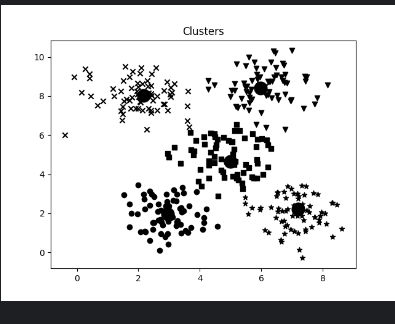
**Завдання 2.9.** Відповідно до рекомендацій, напишіть програму та оцініть максимальну кількість кластерів у заданому наборі даних за допомогою алгоритму зсуву середньою.

Лістинг:

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.cluster import MeanShift, estimate\_bandwidth  
from itertools import cycle  
  
# Load data from input file  
X = np.loadtxt('data\_clustering.txt', delimiter=',')  
  
# Estimate the bandwidth of X  
bandwidth\_X = estimate\_bandwidth(X, quantile=0.1, n\_samples=len(X))  
  
# Cluster data with MeanShift  
meanshift\_model = MeanShift(bandwidth=bandwidth\_X, bin\_seeding=True)  
meanshift\_model.fit(X)  
  
# Extract the centers of clusters  
cluster\_centers = meanshift\_model.cluster\_centers\_  
print('\nCenters of clusters:\n', cluster\_centers)  
  
# Estimate the number of clusters  
labels = meanshift\_model.labels\_  
num\_clusters = len(np.unique(labels))  
print("\nNumber of clusters in input data =", num\_clusters)  
  
# Plot the points and cluster centers  
plt.figure()  
markers = 'o\*xvs'  
for i, marker in zip(range(num\_clusters), markers):  
 # Plot points that belong to the current cluster  
 plt.scatter(X[labels==i, 0], X[labels==i, 1], marker=marker, color='black')  
  
 # Plot the cluster center  
 cluster\_center = cluster\_centers[i]  
 plt.plot(cluster\_center[0], cluster\_center[1], marker='o',  
 markerfacecolor='black', markeredgecolor='black',  
 markersize=15)  
  
plt.title('Clusters')  
plt.show()

Результат:





Модель визначила, що вхідні дані мають 5 кластерів.

**Завдання 2.10.** Використовуючи модель поширення подібності, знайти підгрупи серед учасників фондового ринку.

Лістинг:

import json  
import numpy as np  
import yfinance as yf  
from sklearn import cluster, covariance  
  
# Вхідний файл із символічними позначеннями компаній  
input\_file = "company\_symbol\_mapping.json"  
  
# Завантаження прив'язок символів компаній до їх повних назв  
with open(input\_file, "r") as f:  
 company\_symbols\_map = json.loads(f.read())  
  
symbols, names = np.array(list(company\_symbols\_map.items())).T  
  
# Визначення архівних даних котирувань  
start\_date = "2003-07-03"  
end\_date = "2007-05-04"  
  
# Завантаження архівних даних котирувань  
quotes = []  
valid\_symbols = []  
for symbol in symbols:  
 try:  
 data = yf.download(symbol, start=start\_date, end=end\_date)  
 if not data.empty:  
 quotes.append(data)  
 valid\_symbols.append(symbol)  
 except Exception as e:  
 print(f"Failed to download data for {symbol}: {e}")  
  
# Перевірка чи є валідні дані  
if not quotes:  
 print(  
 "No valid data available for any symbol. Check your symbol mapping and data availability."  
 )  
else:  
 # Оновлення символів на дійсні  
 symbols = valid\_symbols  
  
 # Вилучення котирувань, що відповідають відкриттю та закриттю біржі  
 opening\_quotes = np.array([quote["Open"].values for quote in quotes]).T  
 closing\_quotes = np.array([quote["Close"].values for quote in quotes]).T  
  
 # Обчислення різниці між двома видами котирувань  
 quotes\_diff = closing\_quotes - opening\_quotes  
  
 # Нормалізація даних  
 X = quotes\_diff.copy()  
 X /= X.std(axis=0)  
  
 # Створення моделі графа  
 edge\_model = covariance.GraphicalLassoCV()  
  
 # Навчання моделі  
 with np.errstate(invalid="ignore"):  
 edge\_model.fit(X)  
  
 # Створення моделі кластеризації на основі поширення подібності  
 \_, labels = cluster.affinity\_propagation(edge\_model.covariance\_)  
 num\_labels = labels.max()  
  
 # Виведення результатів  
 print("\nClustering of stocks based on difference in opening and closing quotes:\n")  
 for i in range(num\_labels + 1):  
 cluster\_indices = np.where(labels == i)[0]  
 cluster\_names = names[cluster\_indices]  
 if len(cluster\_names) > 0:  
 print("Cluster", i + 1, "==>", ", ".join(cluster\_names))

Результат:

