**ЛАБОРАТОРНА РОБОТА № 3**

**ДОСЛІДЖЕННЯ МЕТОДІВ РЕГРЕСІЇ ТА НЕКОНТРОЬОВАНОГО НАВЧАННЯ**

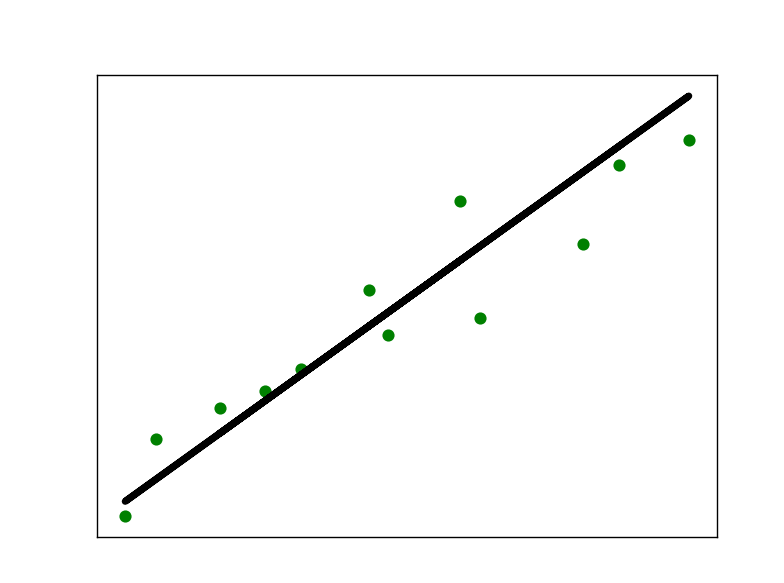
***Мета:*** *використовуючи спеціалізовані бібліотеки і мову програмування Python дослідити методи регресії та неконтрольованої класифікації даних у машинному навчанні*.

**GitHub**: <https://github.com/ingaliptn/AI>

**Хід роботи:**

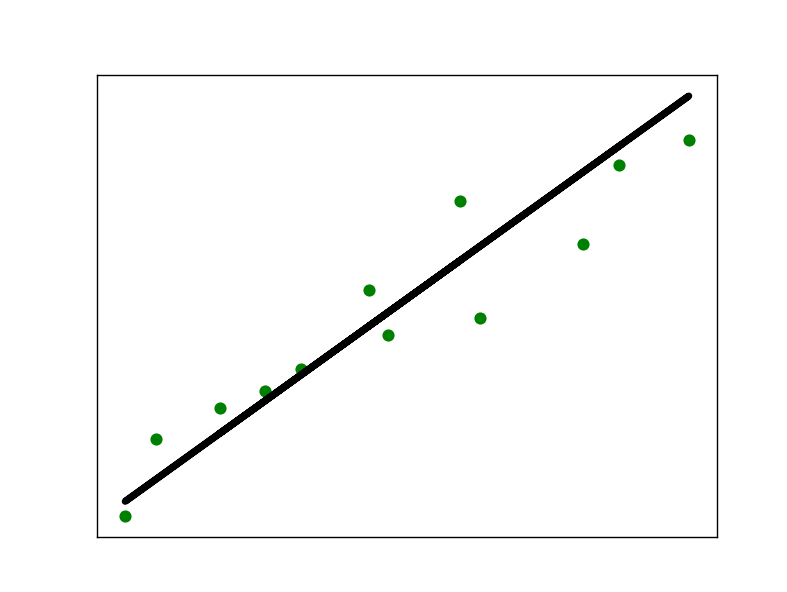
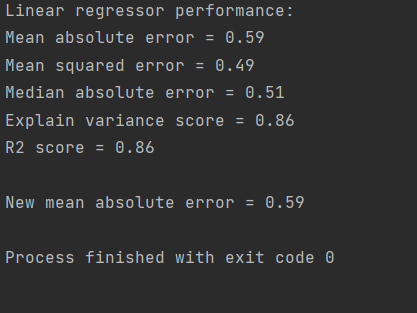
**Завдання 2.1.** Створення регресора однієї змінної

|  |
| --- |
| import pickle  import numpy as np  from sklearn import linear\_model  import sklearn.metrics as sm  import matplotlib.pyplot as plt  input\_file = 'data\_singlevar\_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:]  regressor = linear\_model.LinearRegression()  regressor.fit(X\_train, y\_train)  y\_test\_pred = regressor.predict(X\_test)  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))  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)  # Perform prediction on test data  print("\nNew mean absolute error =", round(sm.mean\_absolute\_error(y\_test, y\_test\_pred\_new), 2)) |



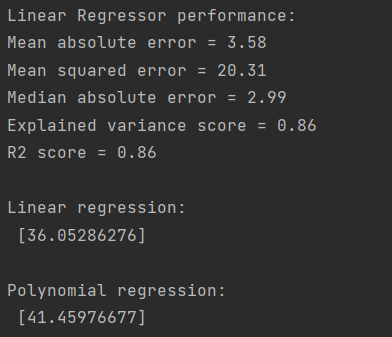
**Завдання 2.2. Передбачення за допомогою регресії однієї змінної**

|  |
| --- |
| import pickle  import numpy as np  from sklearn import linear\_model  import sklearn.metrics as sm  import matplotlib.pyplot as plt  input\_file = 'data\_regr\_5.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:]  regressor = linear\_model.LinearRegression()  regressor.fit(X\_train, y\_train)  y\_test\_pred = regressor.predict(X\_test)  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))  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)) |



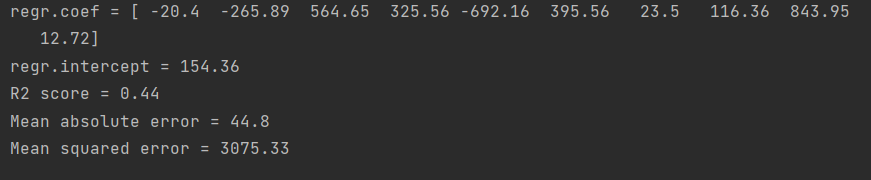
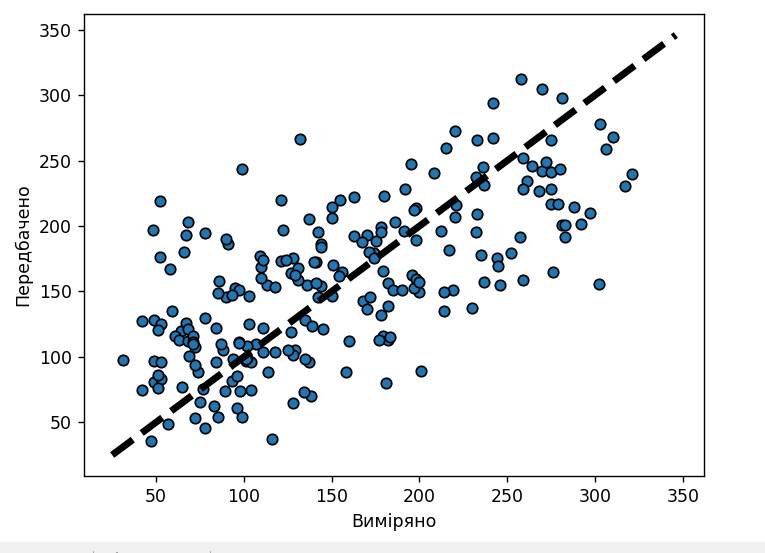
**Завдання 2.3. Створення багатовимірного регресора**

|  |
| --- |
| 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("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))  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)) |



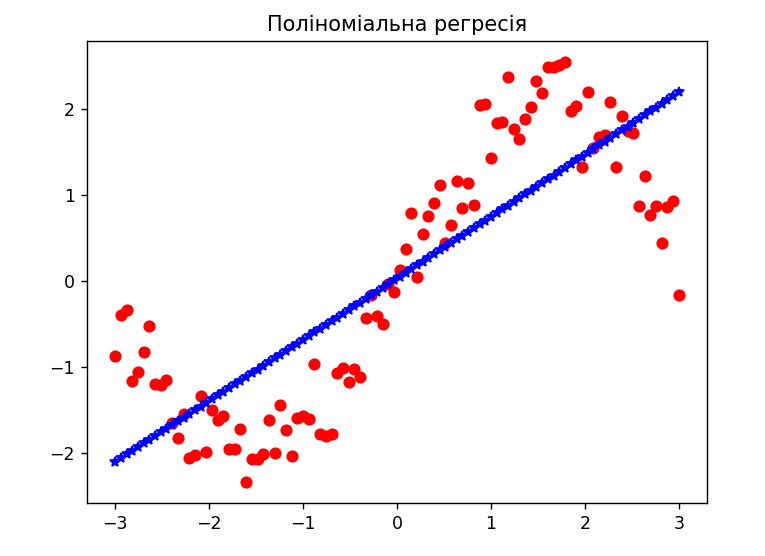
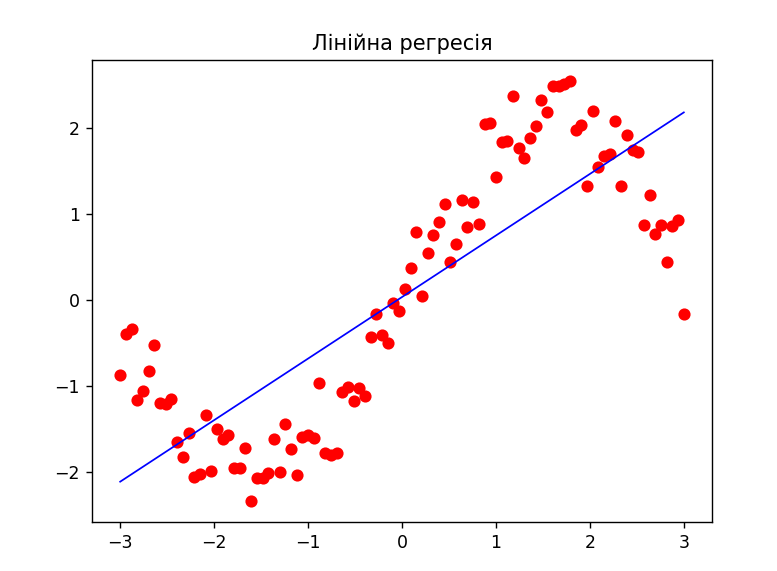
**Завдання 2.4. Регресія багатьох змінних**

|  |
| --- |
| 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)  # Обрахування метрик  print("regr.coef =", np.round(regr.coef\_, 2))  print("regr.intercept =", round(regr.intercept\_, 2))  print("R2 score =", round(r2\_score(ytest, ypred), 2))  print("Mean absolute error =", round(mean\_absolute\_error(ytest, ypred), 2))  print("Mean squared error =", round(mean\_squared\_error(ytest, ypred), 2))  fig, ax = plt.subplots()  ax.scatter(ytest, ypred, edgecolors=(0, 0, 0))  ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=4)  ax.set\_xlabel('Виміряно')  ax.set\_ylabel('Передбачено')  plt.show() |



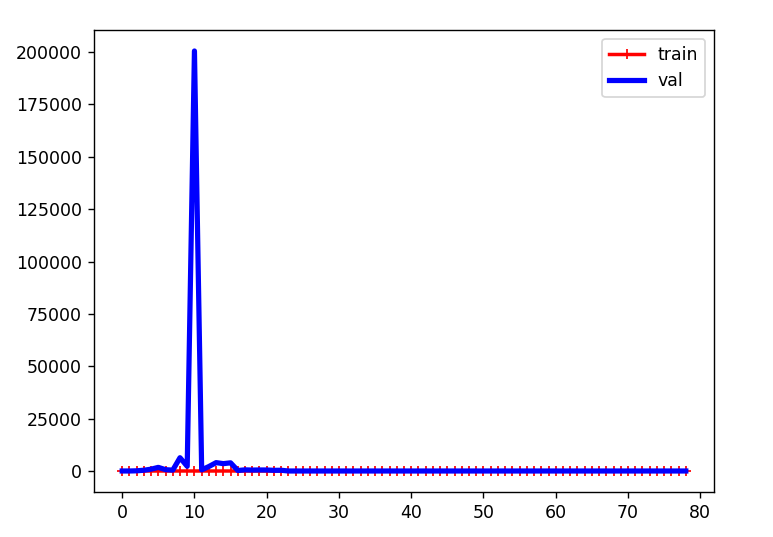
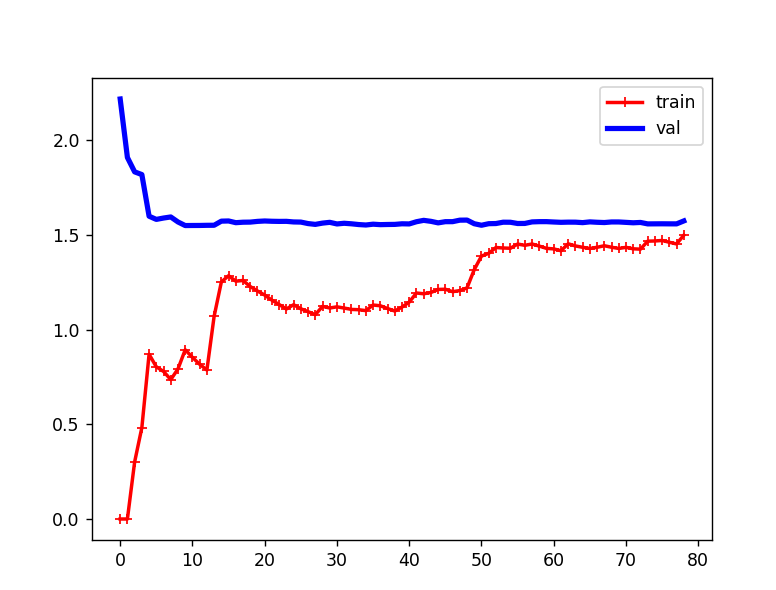
**Завдання 2.5. Самостійна побудова регресії**

|  |
| --- |
| import numpy as np  from matplotlib import pyplot as plt  from sklearn import linear\_model  import sklearn.metrics as sm  from sklearn.preprocessing import PolynomialFeatures  m = 100  X = 6 \* np.random.rand(m, 1) - 3  y = 0.4 \* X \*\* 2 + X + 4 + np.random.randn(m, 1)  X = X.reshape(-1, 1)  y = y.reshape(-1, 1)  linear\_regressor = linear\_model.LinearRegression()  linear\_regressor.fit(X, y)  polynomial = PolynomialFeatures(degree=2, include\_bias=False)  X\_poly = polynomial.fit\_transform(X)  polynomial.fit(X\_poly, y)  poly\_linear\_model = linear\_model.LinearRegression()  poly\_linear\_model.fit(X\_poly, y)  y\_pred = poly\_linear\_model.predict(X\_poly)  print("\nr2: ", sm.r2\_score(y, y\_pred))  plt.scatter(X, y, color='red')  plt.plot(X, linear\_regressor.predict(X), color='blue', linewidth=1)  plt.title("Лінійна регресія")  plt.show()  plt.scatter(X, y, color='red')  plt.plot(X, y\_pred, "\*", color='blue', linewidth=2)  plt.title("Поліноміальна регресія")  plt.show() |



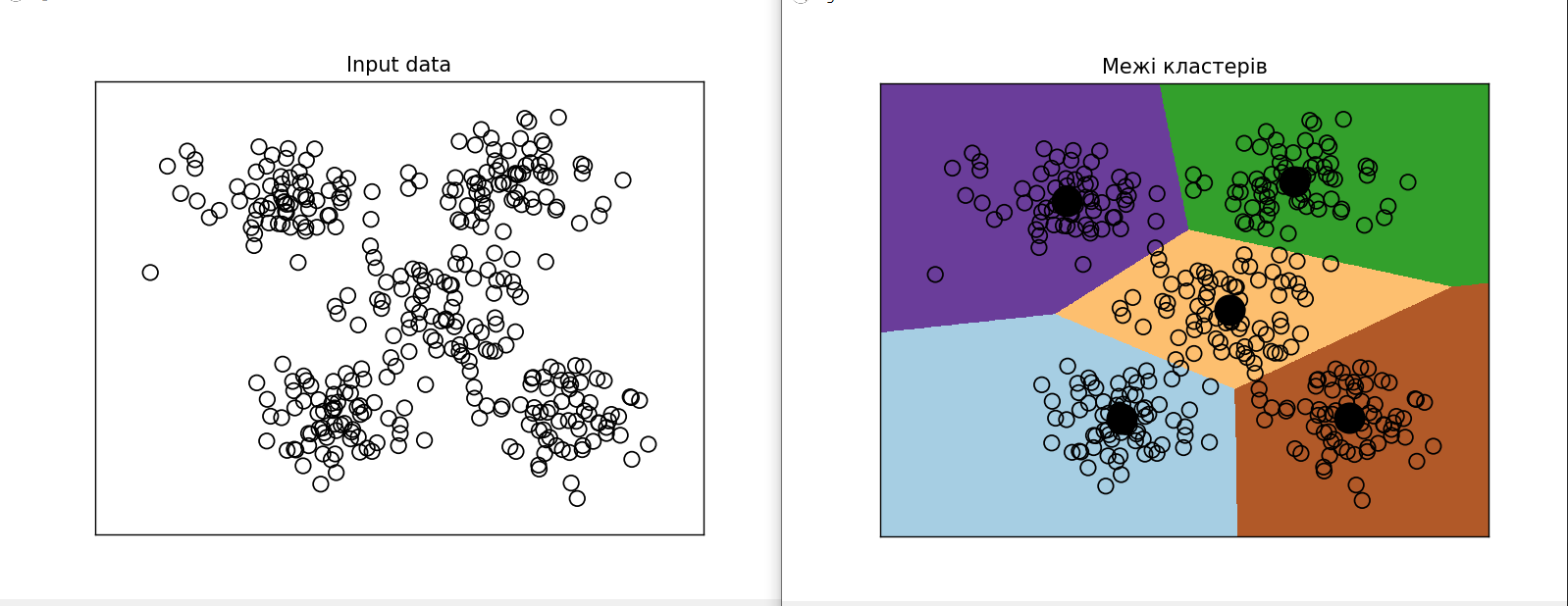
**Завдання 2.6. Побудова кривих навчання**

|  |
| --- |
| 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 = 6 \* np.random.rand(m, 1) - 5  y = 0.7 \* X \*\* 2 + X + 3 + 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()  plt.show()  lin\_reg = linear\_model.LinearRegression()  polynomial\_regression = Pipeline([  ("poly\_features",  PolynomialFeatures(degree=10, include\_bias=False)),  ("lin\_reg", linear\_model.LinearRegression())  ])  plot\_learning\_curves(polynomial\_regression, X, y) |



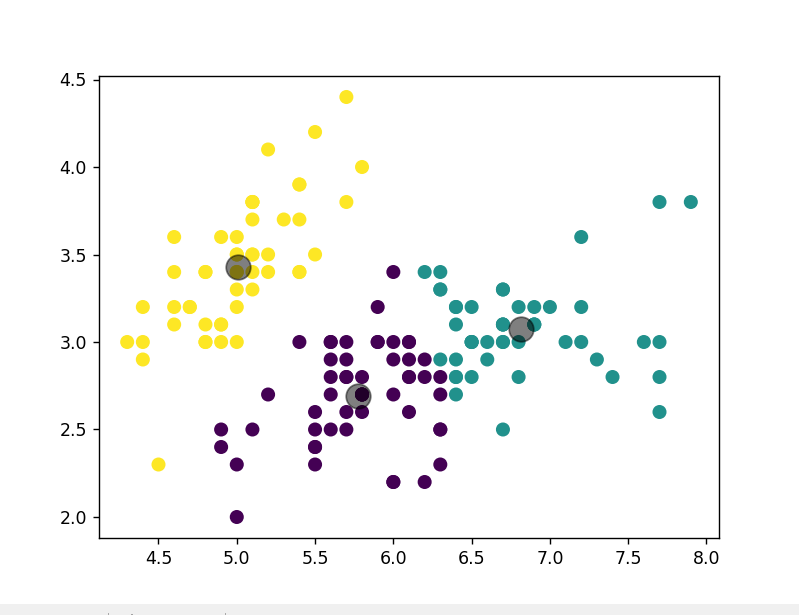
**Завдання 2.7. Кластеризація даних за допомогою методу k-середніх**

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  from sklearn.cluster import KMeans  X = np.loadtxt('data\_clustering.txt', delimiter=',')  num\_clusters = 5  plt.figure()  plt.scatter(X[:, 0], X[:, 1], marker='o', facecolors='none', edgecolors='black', s=80)  x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1  y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1  plt.title('Input data')  plt.xlim(x\_min, x\_max)  plt.ylim(y\_min, y\_max)  plt.xticks(())  plt.yticks(())  # Створення об'єкту КМеаns  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() |



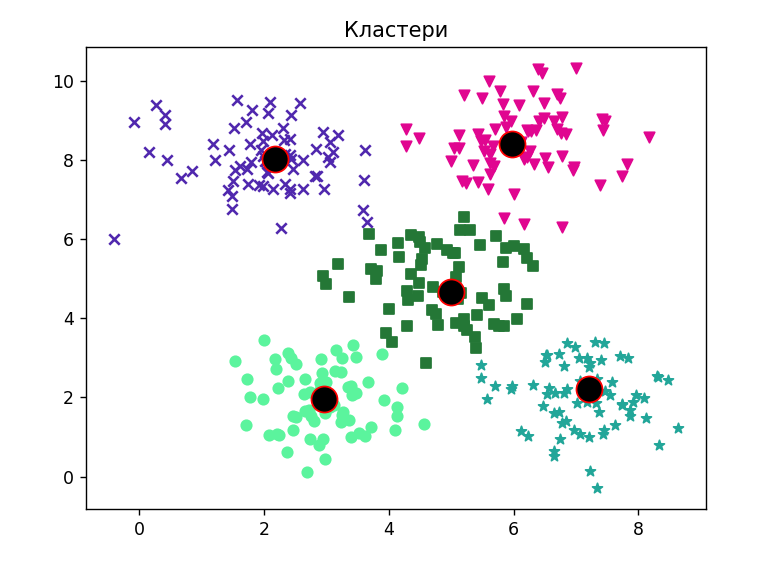
**Завдання 2.8. Кластеризація K-середніх для набору даних Iris**

|  |
| --- |
| 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  kmeans = KMeans(n\_clusters=Y.max() + 1, init='k-means++', n\_init=10, max\_iter=300,  tol=0.0001, verbose=0, random\_state=None, copy\_x=True)  kmeans.fit(X)  y\_pred = kmeans.predict(X)  print("n\_clusters: 3, n\_init: 10, max\_iter: 300, tol: 0.0001, verbose: 0, random\_state: None, copy\_x: True")  print(y\_pred)  plt.figure()  plt.scatter(X[:, 0], X[:, 1], c=y\_pred, s=50, cmap='viridis')  centers = kmeans.cluster\_centers\_  plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5)  plt.show()  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  print("using find\_clusters():")  centers, labels = find\_clusters(X, 3)  print("n\_clusters: 3, rseed: 2")  plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')  plt.show()  centers, labels = find\_clusters(X, 3, rseed=0)  print("n\_clusters: 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)  print("n\_clusters: 3, rseed: 0")  plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')  plt.show() |



**Завдання 2.9. Оцінка кількості кластерів з використанням методу зсуву середнього**

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  from sklearn.cluster import MeanShift, estimate\_bandwidth  X = np.loadtxt('data\_clustering.txt', delimiter=',')  bandwidth\_X = estimate\_bandwidth(X, quantile=0.1, n\_samples=len(X))  meanshift\_model = MeanShift(bandwidth=bandwidth\_X, bin\_seeding=True)  meanshift\_model.fit(X)  cluster\_centers = meanshift\_model.cluster\_centers\_  print('\nCenters of clusters:\n', cluster\_centers)  labels = meanshift\_model.labels\_  num\_clusters = len(np.unique(labels))  print("\nNumber of clusters in input data =", num\_clusters)  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=np.random.rand(3,))  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() |



***Висновки:***.в ході виконання дабораторної роботи я використовуючи спеціалізовані бібліотеки і мову програмування Python дослідив методи регресії та неконтрольованої класифікації даних у машинному навчанні.