7) from sklearn.linear\_model import LogisticRegression

# Sample data (features and labels for binary classification)

X = [[1, 2], [2, 3], [3, 4], [5, 6], [6, 7], [7, 8]]

y = [0, 0, 0, 1, 1, 1] # Binary labels (0 or 1)

# Create and train the Logistic Regression model

model = LogisticRegression()

model.fit(X, y)

# Make a prediction

prediction = model.predict([[4, 5]]) # Predict for a new data point

print("Predicted class:",prediction)

8) import math

def main():

# Get input from the user

numbers = int(input("How many numbers do you want to enter: "))

k = int(input("Enter value of k: "))

# Get numbers

nums = []

for i in range(numbers):

num = int(input(f"Enter Number {i + 1}: "))

nums.append(num)

# Set initial values for kvals (centroids)

kvals = nums[:k] # Initial centroids are the first k elements

prev\_kvals = [0] \* k # To store previous centroids

steps = 1

# Create matrices for calculations and group assignments

addition = [[0 for \_ in range(numbers)] for \_ in range(k)]

groups = [[] for \_ in range(k)] # Change to list of lists

# Show values entered by the user

print("You have entered:", ', '.join(map(str, nums)))

# While loop for iterations

ok = True

while ok:

print(f"\nIteration Number: {steps}")

# Make calculations for the distance between centroids and data points

for i in range(k):

for j in range(numbers):

addition[i][j] = abs(kvals[i] - nums[j])

# Group the numbers based on minimum distance from centroids

for i in range(numbers):

min\_val = math.inf

groupnum = -1

for j in range(k):

if addition[j][i] < min\_val:

min\_val = addition[j][i]

groupnum = j

groups[groupnum].append(nums[i])

# Show results of calculations

print("\nCalculations")

for i in range(numbers):

for j in range(k):

print(f"{addition[j][i]:.2f}\t", end="")

print()

# Show groups and calculate new centroids

print("\nGroups")

for i in range(k):

if groups[i]: # Check if the group is not empty

sum\_vals = sum(groups[i])

count = len(groups[i])

prev\_kvals[i] = kvals[i]

kvals[i] = sum\_vals / count # Update the centroid

print(f"Group {i + 1}: {', '.join(map(str, groups[i]))} = {kvals[i]:.2f}")

else:

print(f"Group {i + 1}: (empty group)")

# Empty the groups for the next iteration

groups = [[] for \_ in range(k)]

# Check if centroids have changed, stop if they haven't

ok = any(prev\_kvals[i] != kvals[i] for i in range(k))

steps += 1

if \_\_name\_\_ == "\_\_main\_\_":

main()