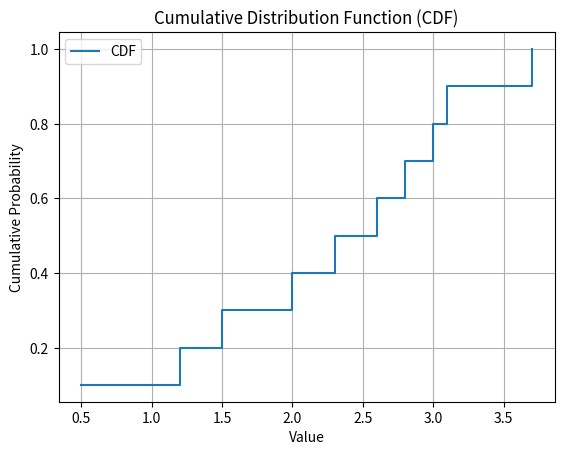
題目 1: 繪製累積分布函數 (CDF)

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  *# 給定的數據*  data = [0.5, 1.2, 2.3, 3.1, 2.8, 1.5, 2.0, 3.7, 2.6, 3.0]  *# 計算累積分布函數 (CDF)*  sorted\_data = np.sort(data)  cdf = np.arange(1, len(sorted\_data) + 1) / len(sorted\_data)  *# 繪圖*  plt.step(sorted\_data, cdf, where='post', label="CDF") *# 使用階梯圖*  plt.title("Cumulative Distribution Function (CDF)")  plt.xlabel("Value")  plt.ylabel("Cumulative Probability")  plt.legend()  plt.grid()  plt.show() |



題目 2: 蒙地卡羅模擬估算圓周率 (π)

|  |
| --- |
| import random  import matplotlib.pyplot as plt  def monte\_carlo\_pi(radius, num\_samples):  inside\_circle = 0  for \_ in range(num\_samples):  x, y = random.uniform(-radius, radius), random.uniform(-radius, radius)  if x\*\*2 + y\*\*2 <= radius\*\*2:  inside\_circle += 1  *# 圓的面積與正方形面積的比例為 πr² / (2r)² = π / 4，因此估算公式不變*  return (inside\_circle / num\_samples) \* 4  *# 設定半徑與測試樣本數*  radius = 50  n\_values = [100, 1000, 10000, 100000, 1000000]  results = []  print("Monte Carlo Simulation for Estimating π (Radius = 50):")  for n in n\_values:  estimated\_pi = monte\_carlo\_pi(radius, n)  results.append((n, estimated\_pi))  print(f"Samples: {n}, Estimated π: {estimated\_pi:.6f}") |

Monte Carlo Simulation for Estimating π (Radius = 50):

Samples: 100, Estimated π: 2.960000

Samples: 1000, Estimated π: 3.156000

Samples: 10000, Estimated π: 3.139600

Samples: 100000, Estimated π: 3.138400

Samples: 1000000, Estimated π: 3.139480

(額外) 題目 2: 蒙地卡羅模擬估算圓周率 (π)

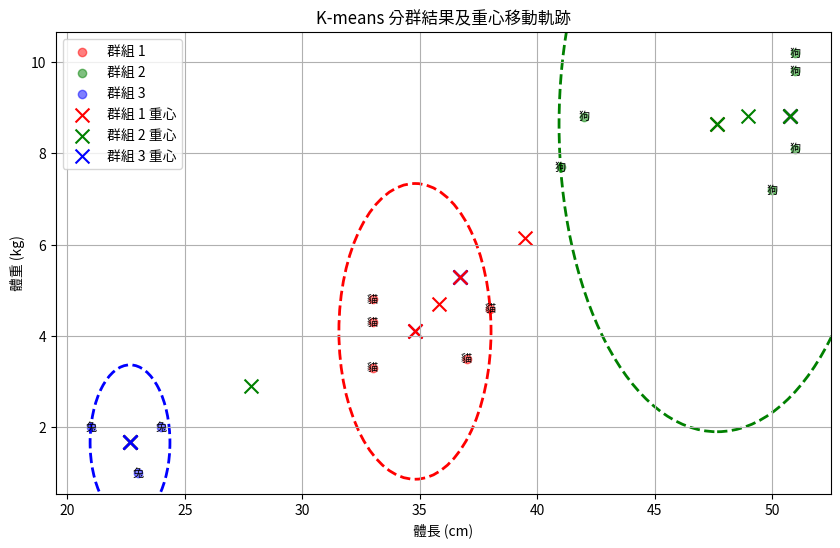
|  |
| --- |
| import random  import matplotlib.pyplot as plt  def monte\_carlo\_pi\_visualized(radius, num\_samples):  inside\_circle = 0  points\_x\_in, points\_y\_in = [], [] *# 圓內點*  points\_x\_out, points\_y\_out = [], [] *# 圓外點*  for \_ in range(num\_samples):  x, y = random.uniform(-radius, radius), random.uniform(-radius, radius)  if x\*\*2 + y\*\*2 <= radius\*\*2:  inside\_circle += 1  points\_x\_in.append(x)  points\_y\_in.append(y)  else:  points\_x\_out.append(x)  points\_y\_out.append(y)  *# 計算 π*  estimated\_pi = (inside\_circle / num\_samples) \* 4  *# 繪圖*  fig, ax = plt.subplots(figsize=(8, 8))  *# 繪製圓形*  circle = plt.Circle((0, 0), radius, color='blue', fill=False, label="Circle (R=50)")  ax.add\_artist(circle)  *# 繪製外接正方形*  square = plt.Rectangle((-radius, -radius), 2\*radius, 2\*radius, color='green', fill=False, label="Bounding Square")  ax.add\_artist(square)  *# 繪製隨機點*  ax.scatter(points\_x\_in, points\_y\_in, color='red', s=1, label="Points Inside Circle")  ax.scatter(points\_x\_out, points\_y\_out, color='blue', s=1, label="Points Outside Circle")  *# 設定圖表屬性*  ax.set\_xlim(-radius - 10, radius + 10)  ax.set\_ylim(-radius - 10, radius + 10)  ax.set\_aspect('equal', adjustable='box')  plt.title(f"Monte Carlo Simulation (Radius = {radius}, Samples = {num\_samples})")  plt.xlabel("X-axis")  plt.ylabel("Y-axis")  plt.legend()  plt.grid(alpha=0.5)  plt.tight\_layout()  plt.show()  return estimated\_pi  *# 執行模擬與繪圖*  *# 設定半徑與測試樣本數*  radius = 50  n\_values = [100, 10000, 100000]  results = []  print("Monte Carlo Simulation for Estimating π (Radius = 50):")  for n in n\_values:  print(f"Estimated π using {n} samples: {estimated\_pi:.6f}")  estimated\_pi = monte\_carlo\_pi\_visualized(radius, n) |

Monte Carlo Simulation for Estimating π (Radius = 50):

|  |  |  |
| --- | --- | --- |
| Estimated π using 100 samples: 3.140512 | Estimated π using 10000 samples: 3.104000 | Estimated π using 100000 samples: 3.158800 |
|  |  |  |

題目 3: K-Means 分群分析

|  |
| --- |
| import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  from sklearn.cluster import KMeans  *# 1. 載入資料*  data = pd.read\_csv('/content/drive/MyDrive/Colab Notebooks/機率學/data/data.csv')  length = data['體長(cm)']  weight = data['體重(kg)']  species = data['動物種類']  *# 2. 執行k-means分群，初始重心隨機選擇*  X = np.array(list(zip(length, weight)))  k = 3 *# 設定群組數*  kmeans = KMeans(n\_clusters=k, init='k-means++', max\_iter=10)  *# 記錄每次迭代的重心位置*  centroids\_history = []  *# 定義 KMeans 的自定義 fit 方法來捕捉每次迭代的重心位置*  def custom\_fit(kmeans, X):  for \_ in range(kmeans.max\_iter):  kmeans.fit(X) *# 執行 KMeans 以更新標籤和重心*  centroids\_history.append(kmeans.cluster\_centers\_) *# 記錄當前重心*  *# 執行自定義 fit 方法*  custom\_fit(kmeans, X)  *# 3. 繪製分群結果的散點圖及重心移動軌跡*  plt.figure(figsize=(10, 6))  colors = ['red', 'green', 'blue'] *# 每個群組的顏色*  center\_colors = ['red', 'green', 'blue'] *# 重心顏色與群組顏色相同*  *# 繪製散點圖*  for i in range(k):  plt.scatter(X[kmeans.labels\_ == i][:, 0], X[kmeans.labels\_ == i][:, 1], color=colors[i], label=f'群組 {i+1}', alpha=0.5)  *# 標記每個數據點的標籤*  for j in range(len(X[kmeans.labels\_ == i])):  plt.text(X[kmeans.labels\_ == i][j][0], X[kmeans.labels\_ == i][j][1], species.iloc[kmeans.labels\_ == i].iloc[j],  color='black', fontsize=8, ha='center', va='center')  *# 4. 繪製每次迭代的重心點，並標註群組名稱*  for iteration, centers in enumerate(centroids\_history):  for i in range(k):  plt.scatter(centers[i][0], centers[i][1], color=center\_colors[i], marker='x', s=100, label=f'群組 {i+1} 重心' if iteration == 0 else "")  *# 5. 用圓圈表示各群最後一次的範圍*  for i in range(k):  *# 取得每個群組的資料點*  group\_data = X[kmeans.labels\_ == i]  *# 計算資料點到重心的最大距離 (半徑)*  centroid = kmeans.cluster\_centers\_[i]  distances = np.linalg.norm(group\_data - centroid, axis=1) *# 計算每個點到重心的距離*  max\_distance = np.max(distances) *# 取最大距離作為圓的半徑*  *# 繪製圓圈 (以重心為圓心，半徑為最大距離)*  circle = plt.Circle((centroid[0], centroid[1]), max\_distance, color=colors[i], fill=False, linestyle='--', linewidth=2)  plt.gca().add\_artist(circle)  plt.xlabel('體長 (cm)')  plt.ylabel('體重 (kg)')  plt.title('K-means 分群結果及重心移動軌跡')  plt.legend()  plt.grid()  plt.show()  *# 6. 輸出每群的最終重心座標及群內樣本數量*  for i in range(k):  print(f'群組 {i+1} 重心: {kmeans.cluster\_centers\_[i]}, 樣本數量: {sum(kmeans.labels\_ == i)}')  *# 7. 輸出每次迭代的重心值*  for iteration, centers in enumerate(centroids\_history):  print(f'第 {iteration + 1} 次迭代的重心: {centers}') |



群組 1 重心: [34.8 4.1], 樣本數量: 5

群組 2 重心: [47.66666667 8.63333333], 樣本數量: 6

群組 3 重心: [22.66666667 1.66666667], 樣本數量: 3

模擬結果說明

1. **題目 1: 繪製累積分布函數 (CDF)**  
   使用 numpy 計算並排序輸入數據，繪製累積分布函數。模擬結果成功展示了數據累積概率隨值的變化，圖形使用階梯圖顯示累積分布的特性，並直觀反映數據分佈情況。
2. **題目 2: 蒙地卡羅模擬估算圓周率 (π)**
   * **非可視化模擬結果**  
     在樣本數分別為 100、1000、10000、100000 和 1000000 時，估算的 π 值逐步接近數學常數 π，顯示蒙地卡羅方法的準確性會隨樣本數增加而提升。
   * **可視化模擬結果**  
     透過繪製隨機點及圓形邊界，清楚顯示了圓內外點的分佈情況，並通過模擬結果進一步驗證 π 值的估算過程。隨著樣本數的增加，模擬結果的分佈越均勻，估算值越接近實際。
3. **題目 3: K-Means 分群分析**  
   使用 K-Means 方法將數據分為三組，並在每次迭代中更新重心。模擬結果展示了分群後各數據點與重心的對應關係，以及每群的數量和最終重心座標。圖形直觀呈現了群組邊界、資料分佈及重心移動軌跡，清晰說明了 K-Means 演算法的收斂過程。

作業心得

透過本次作業，學到了以下幾點：

1. **程式設計能力的提升**  
   本次作業的三個題目涵蓋了數據分析與模擬的不同應用場景。從累積分布函數的計算到蒙地卡羅模擬，再到 K-Means 分群分析，我熟悉了使用 Python 進行數值運算及視覺化的完整流程，也加深了對 numpy、matplotlib 和 sklearn 等工具的理解。
2. **蒙地卡羅方法的實踐**  
   在估算 π 的過程中，體會到了蒙地卡羅方法的簡單與有效性，並學會了如何結合隨機數與幾何方法解決數學問題。通過可視化的點分佈，對圓內外關係的直觀理解也得到了強化。
3. **K-Means 分群的應用**  
   分群分析題目幫助我深入理解了 K-Means 的原理，尤其是重心移動與群組範圍的更新過程。通過多次迭代觀察模型的收斂，對分群演算法的實現與優化有了更深入的體會。
4. **挑戰與改進**  
   作業中最大的挑戰是確保程式碼的邏輯性和穩定性，例如處理輸入數據的異常情況及可視化的美觀性。未來可以進一步優化程式碼，提高執行效率，並嘗試應用更複雜的數據集來驗證演算法的普適性。

整體而言，本次作業不僅鞏固了課堂所學，還增強了實際應用的能力，為未來的專題研究與工作奠定了良好的基礎。