머신러닝실습 레포트

-7주차-

제출일 : 2022.06.05.

학 번 : 2017142012

이 름 : 김효건

# **0. 목차**

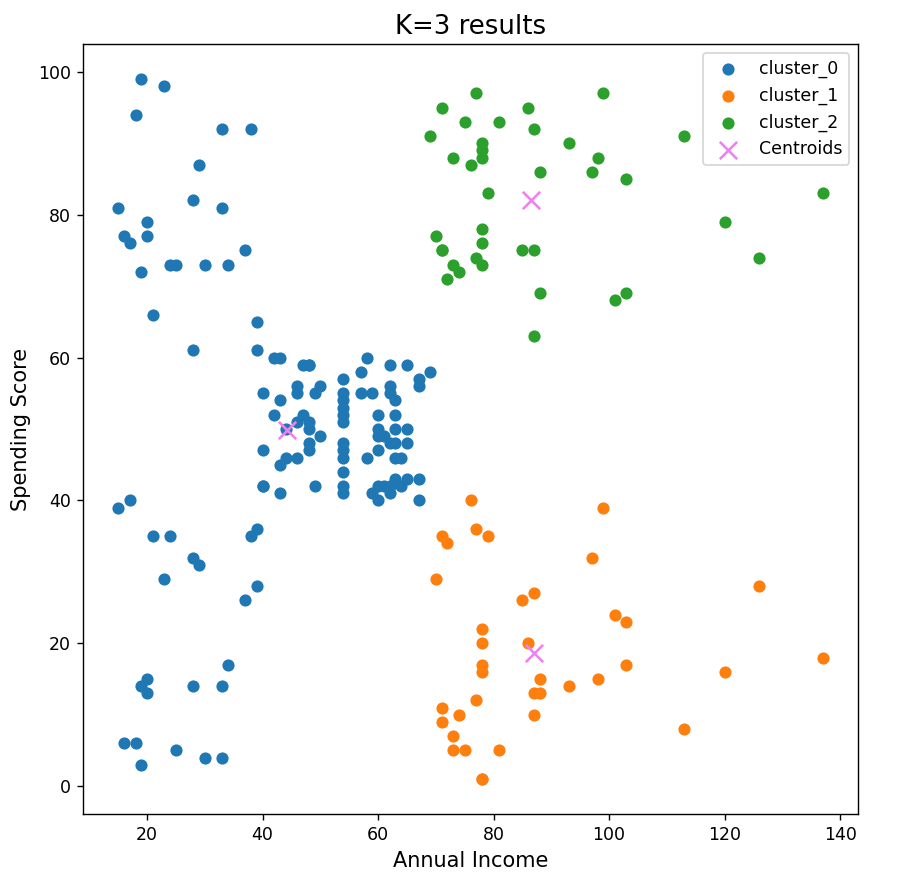
1. **목차**
2. **실습 #1**
   1. Scikit-learn라이브러리 사용하여 K-Means 클러스터링
   2. Elbow method 구현
3. **실습 #2**
4. Income, Spending Score 데이터 그래프 출력
5. **실습 #3**

3.1 K-Means Clustering Raw level 구현 및 결과 출력

# **1. 실습 #1**

**1.1 Scikit-learn 라이브러리 사용하여 K-Means 클러스터링**

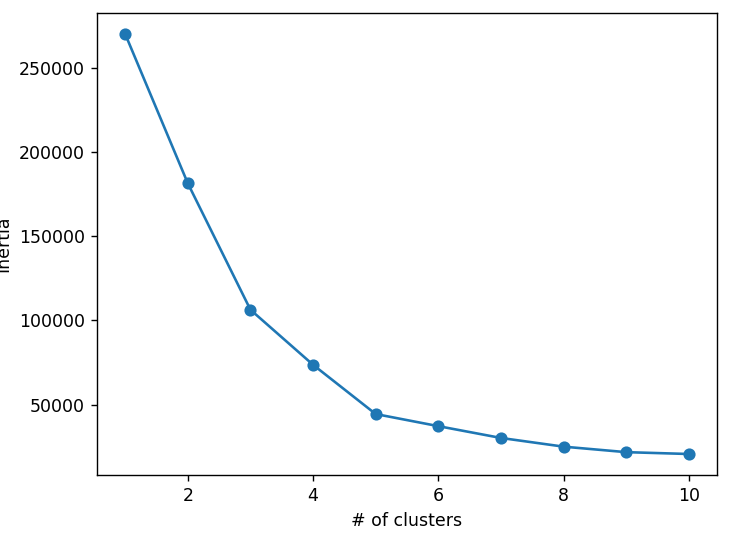
|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  import random  from sklearn.cluster import KMeans  from sklearn.preprocessing import MinMaxScaler  import os  main\_dir = 'C:\\Users\\parks\\Desktop\\대학\\4학년 1학기\\ml\\7week'  # DataFrame 가져오기  dataFrame = pd.read\_csv(os.path.join(main\_dir, "Mall\_Customers.csv"))  # 특성 데이터  data = dataFrame[['Annual Income (k$)', 'Spending Score (1-100)']]  k = 3 # Clustering 갯수  """\_summary\_      n\_clusters : 그룹 수 (k)      k-means++  : 방식, 랜덤값 지정      random\_state : ( 난수 고정)  """  model = KMeans(n\_clusters = k, init='k-means++', random\_state= 10)  # 데이터 프레임에 결과 값 추가  dataFrame['cluster'] = model.fit\_predict(data)  # 마지막 중심 값  # final\_centroid type : numpy. ndrray  final\_centroid = model.cluster\_centers\_  print(final\_centroid)  # 그래프 출력  plt.figure(figsize=(8,8))  for i in range(k):      plt.scatter(dataFrame.loc[dataFrame['cluster'] == i, 'Annual Income (k$)'], dataFrame.loc[dataFrame['cluster'] == i, 'Spending Score (1-100)'], label="cluster\_"+str(i))  plt.scatter(final\_centroid[:,0], final\_centroid[:,1], s=100, c='violet', marker='x', label="Centroids")  plt.legend()  plt.title(f'K={k} results', size=15)  plt.xlabel('Annual Income', size=12)  plt.ylabel('Spending Score', size=12)  plt.show() |



(그림1. 클러스터링 결과)

**1.2 Elbow method 구현**

|  |
| --- |
| # elbow method  def elbow(x):      """\_summary\_          x: data      """      sse = []      for i in range(1, 11):          km = KMeans(n\_clusters = i, init = 'k-means++', random\_state=0)          km.fit(x)          sse.append(km.inertia\_)          print(km.inertia\_)      plt.plot(range(1,11), sse, marker = 'o')      plt.xlabel("# of clusters")      plt.ylabel("Inertia")      plt.show()  elbow(data) |

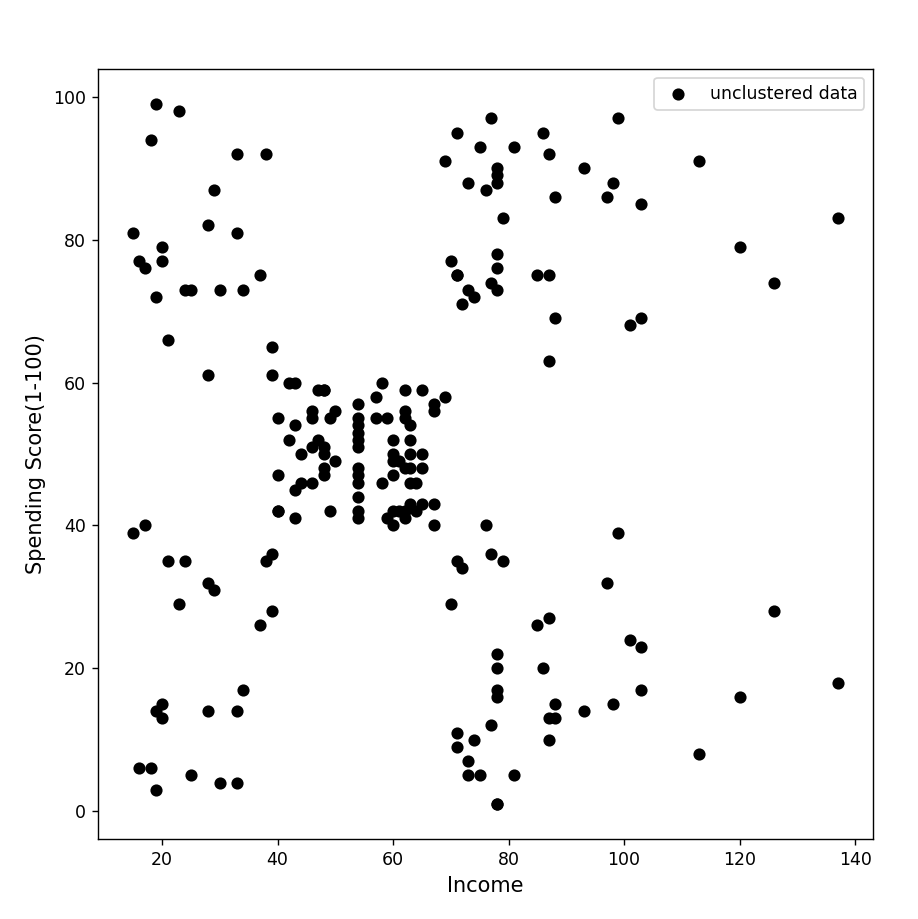


(그림2. 이너셔 그래프)

# **2. 실습 #2**

**2.0 Income, Spending Score 데이터 그래프 출력**

|  |
| --- |
| # Income, Spending Score(1~100) 그래프  plt.figure(figsize=(8,8))  plt.scatter(data['Annual Income (k$)'], data['Spending Score (1-100)'], color='black', label='unclustered data')  plt.legend(loc = 'upper right')  plt.xlabel('Income', size=12)  plt.ylabel('Spending Score(1-100)', size=12)  plt.show() |

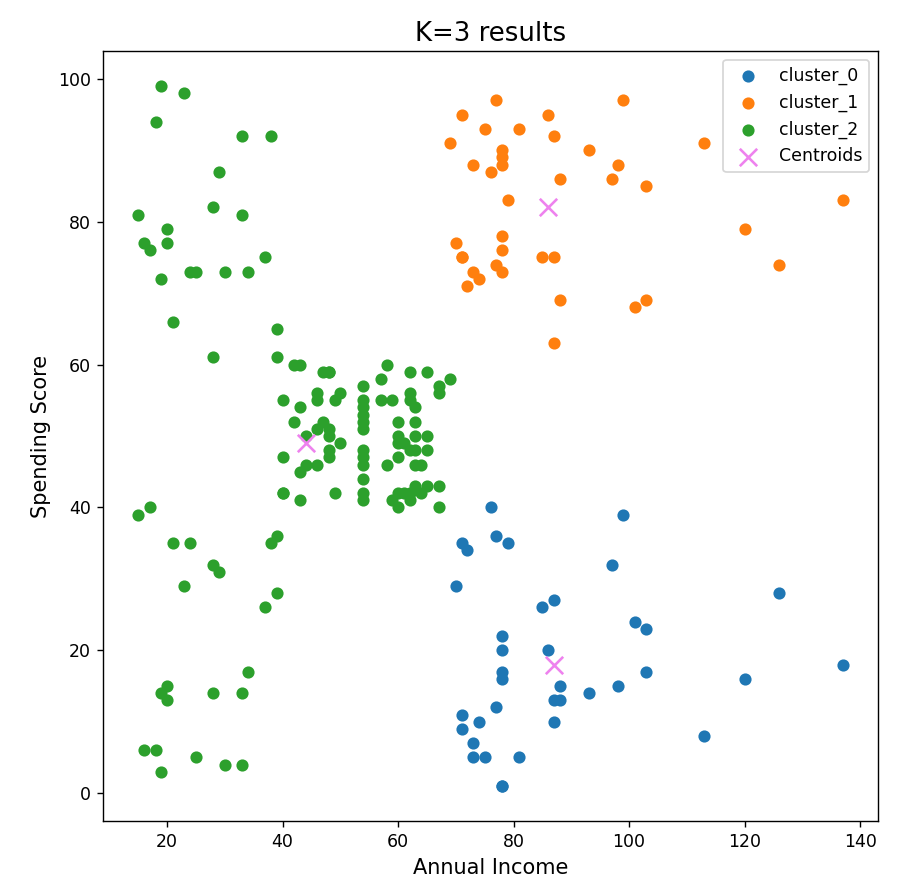


(그림 3. 클러스터링 안된 데이터)

# **3. 실습 #3**

**3.1 K-Means Clustering Raw level 구현 및 그래프**

|  |
| --- |
| K=3 # 클러스터 갯수  iteration = 100 # 반복 횟수  data\_set = data.to\_numpy() # Income, Spending Score 데이터셋 리스트로 변환  # 랜덤 클러스터 초기값 가져오는 함수  def get\_random\_clusters(k):      """\_summary\_          k : 클러스터 갯수          Returns: 클래스터 갯수와 특성에 맞게 반환      """      data\_length = len(data) # 데이터 갯수 저장        # 데이터 갯수만큼의 범위로 k개의 인덱스 랜덤으로 가져온다.      rand\_indexes = [random.randrange(0, data\_length) for \_ in range(k)]      print("rand indexes : ",rand\_indexes)      return data\_set[rand\_indexes] # 데이터 셋에서 랜덤 인덱스에 맞게 가져온다.  # 초기 클러스터 중심점 저장  cluster\_centroid =  get\_random\_clusters(k)  print(cluster\_centroid)  # p-norm 함수  # 유클리디안 거리  def euclidean\_dist(data, centroid):      result = list(map(lambda x: sum(x), (centroid - data) \*\* 2))      return result  # 맨해튼 거리  def manhattan\_dist(data, centroid):      result = list(map(lambda x: sum(x), abs(centroid - data)))      return result  # 클러스터 중심정들 중 가장 가까운 지점 찾아내는 함수  def find\_cluster\_index(result):      return np.argmin(result)  # 다음 클러스터 중심정르 위해 평균을 구하는 함수  def find\_mean(cluster\_list, features):      result = np.array([[0 for \_ in range(features)] for \_ in range(len(cluster\_list))])        # 클러스트 리스트 : 클러스터 벡터      for i in range(len(cluster\_list)):          length = len(cluster\_list[i])          for d in cluster\_list[i]: # 클러스터의 벡터 값들의 합              result[i] = [result[i][f] + d[f] for f in range(features)]          # 클러스타 벡터의 평균을 구함          result[i] = [result[i][f] / length for f in range(features)]      return result  # 클러스터링 동작하는 메인 함수  def clustering(data\_set, cluster\_centroid, k):      # 클러스터의 벡터가 변화했는지 감지하기 위한 변수      old\_list = [[] for \_ in range(k)]        # 데이터 셋의 특성 갯수      features = data\_set.shape[1]        # 100회 반복      for i in range(iteration):          cluster\_list = [[] for \_ in range(k)]            # 데이터 셋 전체          for d in data\_set:              # 각 데이터와 중심간 거리              result = euclidean\_dist(d,cluster\_centroid)              # 거리를 통한 가장 가까운 곳의 중심값 인덱스              cluster\_index = find\_cluster\_index(result)              # 중심값을 중심으로한 벡터에 포함              cluster\_list[cluster\_index].append(d)          # 기존 클러스터 벡터와 변경된 클러스터 벡터가 같으면 종료          if np.array\_equal(old\_list,cluster\_list):              return cluster\_list, cluster\_centroid, i            # 평균 구하는 함수          cluster\_centroid = find\_mean(cluster\_list, features)          old\_list = cluster\_list.copy()        return cluster\_list, cluster\_centroid, i  f\_cluster\_list, f\_cluster\_centroid, final\_index = clustering(data\_set, cluster\_centroid, k)  print(f\_cluster\_centroid)  # 구한 값들을 통해 결과 출력  plt.figure(figsize=(8,8))  for i, cluster in enumerate(f\_cluster\_list):      cluster = np.array(cluster).T      plt.scatter(cluster[0], cluster[1], label="cluster\_"+str(i))    final\_centroid = np.array(f\_cluster\_centroid).T  plt.scatter(final\_centroid[0], final\_centroid[1], s=100, c='violet', marker='x', label="Centroids")  plt.legend()  plt.title(f'K={k} results', size=15)  plt.xlabel('Annual Income', size=12)  plt.ylabel('Spending Score', size=12)  plt.show() |



(그림4. 클러스터링 결과)