**K-Means Clustering of People with COVID-19**

**July 2, 2020**

**DoYeong, Jeon  
Software Engineering Laboratory  
Soongsil University**

**Table of Contents**

[1. Source Code 3](#_Toc44616585)

[1.1. Code for Creating Database 3](#_Toc44616586)

[1.2. Code for Clustering 5](#_Toc44616587)

[2. Result of Clustering 10](#_Toc44616588)

[2.1. Loaded Dataset 10](#_Toc44616589)

[2.2. K-Means 11](#_Toc44616590)

# Source Code

## Code for Creating Database

* CreatingDB Class

|  |
| --- |
| class CreatingDB:  *"""  Class for creating random database  """* num\_people = 0 *# number of people to create* base\_date = None *# the base date of data* def \_\_init\_\_(self, num\_people, base\_date):  self.num\_people = num\_people  self.base\_date = base\_date   def generate\_incurred\_date(self):  *"""  function to create random incurred date  :return:  incurred\_date: string, the day of infection or contact  elapsed\_days: int, the difference between base date and incurred date  """* elapsed\_days = random.randint(0, 14) *# the valid day period is 0~14  # extracting the incurred day using periods and base date* incurred\_date = (self.base\_date - timedelta(days=elapsed\_days)). \  strftime(**"%Y %m %d"**)  return incurred\_date, elapsed\_days   def generate\_address\_list(self):  *"""  function to get one address randomly from the adress list  :return: the randomly generated address list  """* with open(**'./Address\_Part.txt'**, **'r'**, encoding=**'utf-8'**) as add\_file:  *# add\_file = add\_file.encoding* address\_list = add\_file.readlines()   random\_address\_list = [] *# list to store addresses   # extract addresses as many as the number of recipients* for \_ in range(1, self.num\_people + 1):  random\_address\_list.append(random.choice(address\_list))   return random\_address\_list   def generate\_csv\_data(self):  *"""  function to create .csv file with randomly generated records  :return: None  """* num\_healthy = round(self.num\_people / 3) *# 1/3 is healthy* num\_contacted = round(self.num\_people / 3) *# 1/3 is contacted  # 1/3 is confirmed* num\_confirmed = self.num\_people - num\_healthy - num\_contacted   id\_list = list(range(1, self.num\_people + 1)) *# ID as many as people* random.shuffle(id\_list) *# shuffle list   # age records as many as people* age\_list = list(random.randint(1, 100)  for \_ in range(1, self.num\_people + 1))  *# address records as many as people* address\_list = self.generate\_address\_list()   severity\_list = [] *# severity records as many as people* incurred\_date\_list = [] *# incurred date list including 'None'(healthy)* status\_list = [] *# status(Healthy, Contacted, and Confirmed) list   # Entire people num = healthy + contacted + confirmed  # Repeat as many healthy people* for \_ in range(num\_healthy):  *# severity\_list.append(0)* status\_list.append(**'Healthy'**)  incurred\_date\_list.append(**'None'**)   *# Repeat as many contacted people* for count in range(num\_contacted):  date, days = self.generate\_incurred\_date()  status\_list.append(**'Contacted'**)  *# severity\_list.append(round(self.compute\_severity('contacted', days), 2))* incurred\_date\_list.append(date)   *# Repeat as many confirmed people* for \_ in range(num\_confirmed):  date, days = self.generate\_incurred\_date()  status\_list.append(**'Confirmed'**)  *# severity\_list.append(round(self.compute\_severity('confirmed', days), 2))* incurred\_date\_list.append(date)   *# converting as pandas DataFrame data type to save .csv* df = pd.DataFrame({  **"ID"**: id\_list,  **"Age"**: age\_list,  **"Address"**: address\_list,  **"Covid Status"**: status\_list,  *# "Severity": severity\_list,* **"Incurred Date"**: incurred\_date\_list,  })  df = df.sort\_values([**'ID'**], ascending=[True])  df.reset\_index(drop=True, inplace=True)   *# saving as .csv file* df.to\_csv(**"corona\_data.csv"**, mode=**'w'**, encoding=**'utf-8-sig'**) |

## Code for Clustering

* ClusteringPeople Class

|  |
| --- |
| from datetime import datetime, timedelta from dateutil.parser import parse import random import pandas as pd import numpy as np import sklearn from sklearn import cluster from sklearn.metrics import silhouette\_score import matplotlib.pyplot as plt from matplotlib import cm   class CreatingDB:  *"""  Class for creating random database  """* num\_people = 0 *# number of people to create* base\_date = None *# the base date of data* def \_\_init\_\_(self, num\_people, base\_date):  self.num\_people = num\_people  self.base\_date = base\_date   def generate\_incurred\_date(self):  *"""  function to create random incurred date  :return:  incurred\_date: string, the day of infection or contact  elapsed\_days: int, the difference between base date and incurred date  """* elapsed\_days = random.randint(0, 14) *# the valid day period is 0~14  # extracting the incurred day using periods and base date* incurred\_date = (self.base\_date - timedelta(days=elapsed\_days)). \  strftime(**"%Y %m %d"**)  return incurred\_date, elapsed\_days   def generate\_address\_list(self):  *"""  function to get one address randomly from the adress list  :return: the randomly generated address list  """* with open(**'./Address\_Part.txt'**, **'r'**, encoding=**'utf-8'**) as add\_file:  *# add\_file = add\_file.encoding* address\_list = add\_file.readlines()   random\_address\_list = [] *# list to store addresses   # extract addresses as many as the number of recipients* for \_ in range(1, self.num\_people + 1):  random\_address\_list.append(random.choice(address\_list))   return random\_address\_list   def generate\_csv\_data(self):  *"""  function to create .csv file with randomly generated records  :return: None  """* num\_healthy = round(self.num\_people / 3) *# 1/3 is healthy* num\_contacted = round(self.num\_people / 3) *# 1/3 is contacted  # 1/3 is confirmed* num\_confirmed = self.num\_people - num\_healthy - num\_contacted   id\_list = list(range(1, self.num\_people + 1)) *# ID as many as people* random.shuffle(id\_list) *# shuffle list   # age records as many as people* age\_list = list(random.randint(1, 100)  for \_ in range(1, self.num\_people + 1))  *# address records as many as people* address\_list = self.generate\_address\_list()   severity\_list = [] *# severity records as many as people* incurred\_date\_list = [] *# incurred date list including 'None'(healthy)* status\_list = [] *# status(Healthy, Contacted, and Confirmed) list   # Entire people num = healthy + contacted + confirmed  # Repeat as many healthy people* for \_ in range(num\_healthy):  *# severity\_list.append(0)* status\_list.append(**'Healthy'**)  incurred\_date\_list.append(**'None'**)   *# Repeat as many contacted people* for count in range(num\_contacted):  date, days = self.generate\_incurred\_date()  status\_list.append(**'Contacted'**)  *# severity\_list.append(round(self.compute\_severity('contacted', days), 2))* incurred\_date\_list.append(date)   *# Repeat as many confirmed people* for \_ in range(num\_confirmed):  date, days = self.generate\_incurred\_date()  status\_list.append(**'Confirmed'**)  *# severity\_list.append(round(self.compute\_severity('confirmed', days), 2))* incurred\_date\_list.append(date)   *# converting as pandas DataFrame data type to save .csv* df = pd.DataFrame({  **"ID"**: id\_list,  **"Age"**: age\_list,  **"Address"**: address\_list,  **"Covid Status"**: status\_list,  *# "Severity": severity\_list,* **"Incurred Date"**: incurred\_date\_list,  })  df = df.sort\_values([**'ID'**], ascending=[True])  df.reset\_index(drop=True, inplace=True)   *# saving as .csv file* df.to\_csv(**"corona\_data.csv"**, mode=**'w'**, encoding=**'utf-8-sig'**)   class ClusteringPeople:  df\_corona = None  cluster\_result\_dic = {}   def \_\_init\_\_(self, file\_path):  self.load\_data(file\_path)   def load\_data(self, file\_path):  *"""  method to load .csv file  :param file\_path: string, the path of file  :return:  """* self.df\_corona = pd.read\_csv(file\_path)   def display\_load\_data(self):  print(**f"**{**'ID'**:**<4**}{**'Age'**:**<4**}{**'Covid Status'**:**<13**}{**'Severity'**:**<9**}{**'Address'**:**<10**}**"**)  for i in range(len(self.df\_corona)):  print(**f"**{self.df\_corona[**'ID'**][i]:**<4**}**"  f"**{self.df\_corona[**'Age'**][i]:**<4**}**"  f"**{self.df\_corona[**'Covid Status'**][i]:**<13**}**"  f"**{round(self.df\_corona[**'Severity'**][i], 3):**<9**}**"  f"**{self.df\_corona[**'Address'**][i].split()[0]:**<10**}**"** )  print() *# float 1 line* def preprocess(self):  *"""  method to preprocess the data for distance function  :return: None  """* col\_num = len(self.df\_corona) *# the number of rows from loaded data* today = datetime.now().date() *# date of today, YEAR-MONTH-DAY   # selecting specific column to compute 'severity'* incur\_date\_col = self.df\_corona[**'Incurred Date'**]  status = self.df\_corona[**'Covid Status'**]   severity\_list = [] *# list for storing severity result* for i in range(col\_num):  severity = 0 *# default is healthy, 0.* if status[i] == **'Contacted'**: *# contacted person?  # formula for contacted person:  # x = 1 - ((today's date) - (infected date)) \* 0.05)* elapsed\_days = (today - parse(incur\_date\_col[i]).date()).days  severity = 1 - (elapsed\_days \* 0.05)  elif status[i] == **'Confirmed'**: *# confirmed person?  # formula for confirmed person:  # x = (1 - ((today's date) - (infected date)) \* 0.05)) / 2* elapsed\_days = (today - parse(incur\_date\_col[i]).date()).days  severity = (1 - (elapsed\_days \* 0.05)) \* 0.5   severity\_list.append(severity) *# add the value to the list* self.df\_corona[**"Severity"**] = severity\_list   def cluster(self):  sse\_list = [] *# list for storing SSE(Sum of squares errors)* silhouette\_score\_list = [] *# list for storing silhouette scores* for i in range(2, 10): *# number of clusters 2 to 9  # load the k-means model* km = cluster.KMeans(  n\_clusters=i, *# the number of cluster* init=**'k-means++'**, *# how to initial cluster centers* max\_iter=300, *# maximum number of iterations* algorithm=**'auto'** *# three choices: auto, full, and elkan.* )   *# changing the shape of data* severity\_list = self.df\_corona[**"Severity"**].values.tolist()  severity\_list = np.array(severity\_list)   *# cluster* cluster\_predicted\_list = km.fit\_predict(severity\_list.reshape(-1, 1))   *# storing SSE value to get the optimal number of cluster* sse\_list.append(km.inertia\_)   *# storing silhouette score to get optimal number of cluster* silhouette\_score\_list.append(silhouette\_score(severity\_list.reshape(-1, 1), cluster\_predicted\_list))   cluster\_list = [j for j in range(i)] *# cluster list  # display the reuslt of cluster* self.print\_result\_of\_cluster(cluster\_list, cluster\_predicted\_list)   *# store the prediction result* self.cluster\_result\_dic[i] = cluster\_predicted\_list   def draw\_elbow\_method(self, sse\_list):  *"""  method to draw elbow graph using SSE(Sum of Squares Error)  :param sse\_list: list of SSE  :return: None  """* plt.plot(range(2, 10), sse\_list, marker=**'o'**)  plt.xlabel(**"The Number of Cluster"**)  plt.ylabel(**"SSE"**)  plt.show()   def print\_result\_of\_cluster(self, cluster\_list, cluster\_predicted\_list):  *"""  form  Cluster 1:  Number of people: n  Severity Values: [...]  Average of severities: n  Cluster 2:  ...   :return:  """* severity\_list = self.df\_corona[**"Severity"**].values.tolist()  id\_list = self.df\_corona[**"ID"**].values.tolist()   cluster\_predicted\_list = cluster\_predicted\_list.tolist()  print(**f"The number of Cluster:** {len(cluster\_list)}**"**)  for cluster\_idx in cluster\_list:  num\_people = cluster\_predicted\_list.count(cluster\_idx)  id\_severity\_tuple\_list = []  sum\_of\_severities = 0  for person\_idx in range(len(cluster\_predicted\_list)):  if cluster\_idx == cluster\_predicted\_list[person\_idx]:  sum\_of\_severities += severity\_list[person\_idx]  id\_severity\_tuple\_list.append((person\_idx+1, round(severity\_list[person\_idx], 2)))  print(**f"**\t**Cluster** {cluster\_idx}**:"**)  print(**f"**\t\t**Number of People:** {num\_people}**"**)  print(**f"**\t\t**People list with Severity Values:"**)  print(**f"**\t\t\t{**'ID'**:**<4**}{**'Severity Value'**}**"**)  for person\_in\_cluster in id\_severity\_tuple\_list:  if id\_severity\_tuple\_list.index(person\_in\_cluster) % 2 == 0:  print(**f"**\t\t\t{person\_in\_cluster[0]:**<4**}{person\_in\_cluster[1]}**"**)   print(**f"**\t\t**Average of severities:** {round(sum\_of\_severities / len(id\_severity\_tuple\_list), 2)}**"**)  print() *# float 1 line* def draw\_silhouette(self):  *"""  method to draw graph using silhouette scores  :return: None  """* pass   def draw\_graph(self):  *"""  method to draw clustering result  :return: None  """* pass   if \_\_name\_\_ == **'\_\_main\_\_'**:  *# CODE FOR CREATING DATABASE  # require the number of people and base date  # num\_people = int(input("Enter the number of people: "))  # date\_input = input("Enter the base date(Year-Month-Day): ")  # if date\_input == '':  # print("The base date is set as today.")  # date = datetime.now().date()  # else:  # date = parse(date\_input).date()  #  # cdb = CreatingDB(num\_people, date) # creating instance  # cdb.generate\_csv\_data() # creating .csv file   # CODE FOR CLUSTERING* file\_path = **'./corona\_data.csv'** cp = ClusteringPeople(file\_path)  cp.preprocess()  cp.display\_load\_data()  cp.cluster()  *# cp.draw\_graph()* |

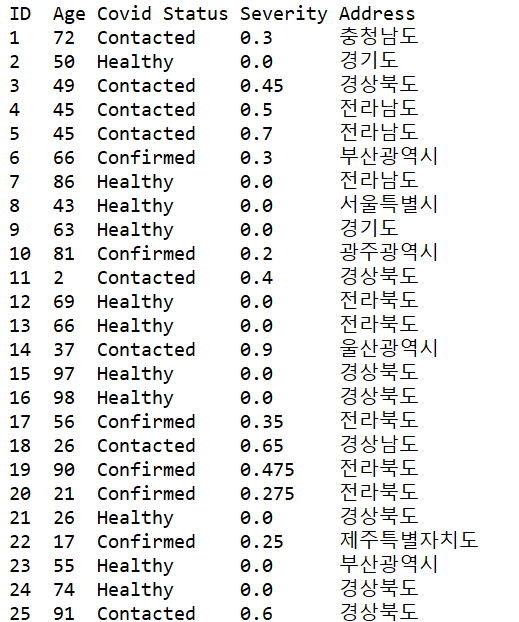
* main

|  |
| --- |
| if \_\_name\_\_ == **'\_\_main\_\_'**:  *# CODE FOR CLUSTERING* file\_path = **'./corona\_data.csv'** cp = ClusteringPeople(file\_path)  cp.preprocess()  cp.draw\_graph()  cp.cluster() |

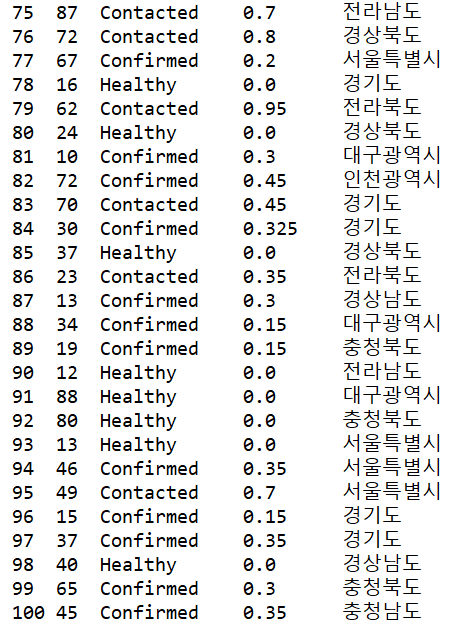
# Result of Clustering

## Loaded Dataset

* Top 25 lines

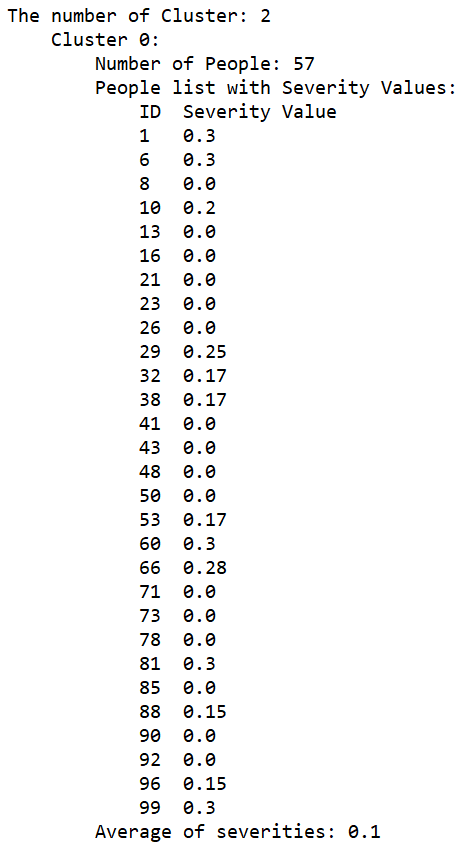


* Last 25 lines

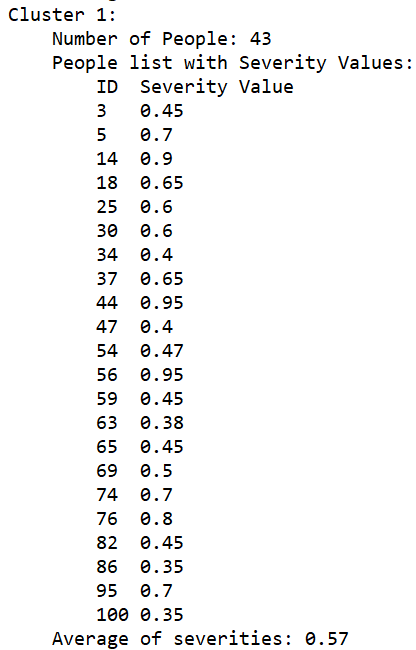


## K-Means

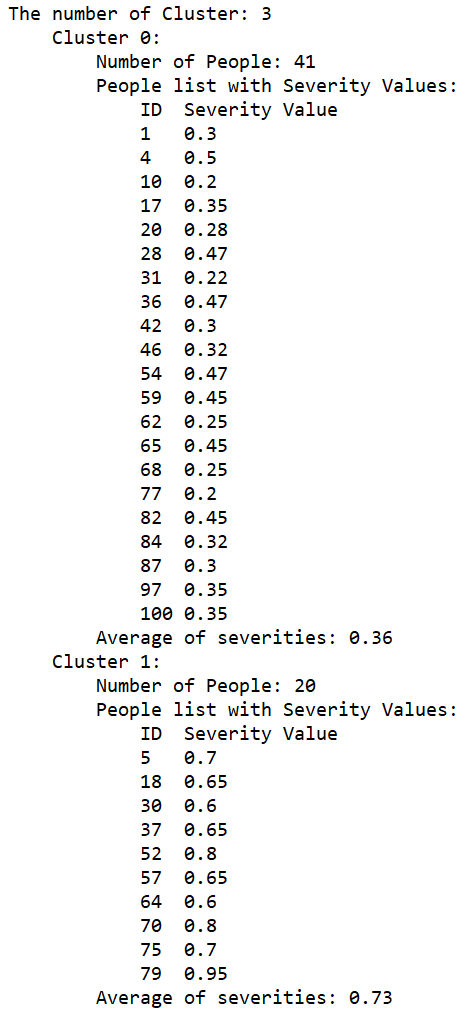
* The number of Cluster: 2
* Cluster 0



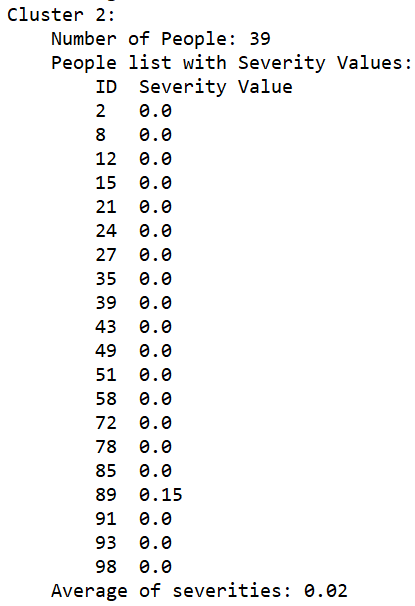
* Cluster 1



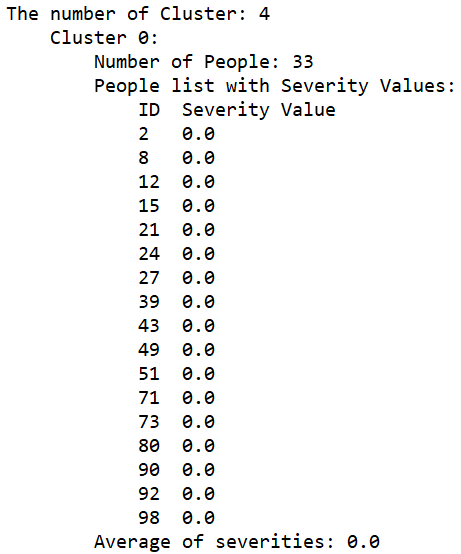
* The number of Cluster: 3
* Cluster 0, 1



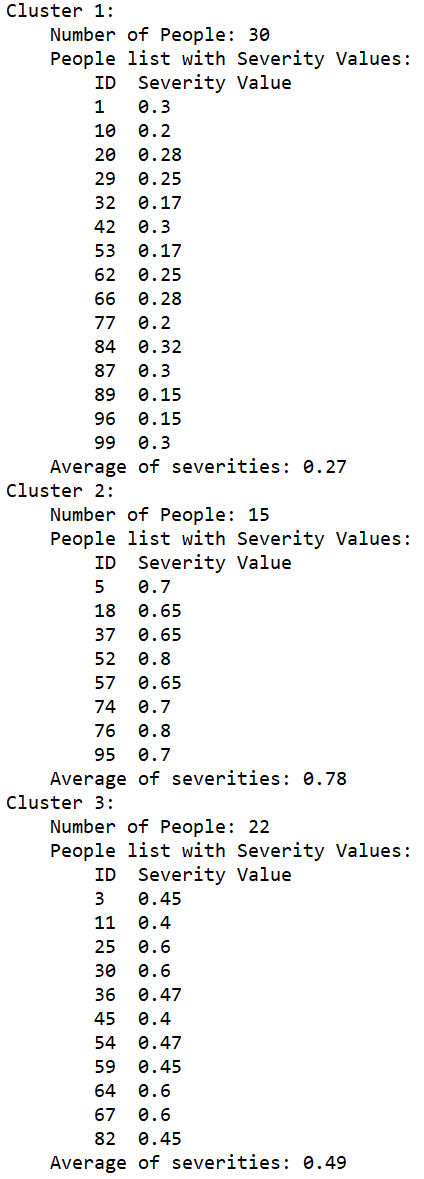
* Cluster 2



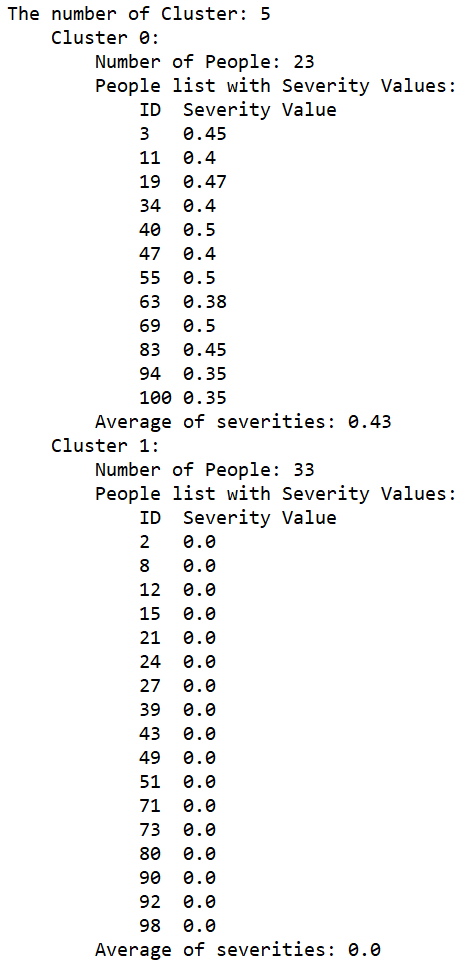
* The number of Cluster: 4
* Cluster 0



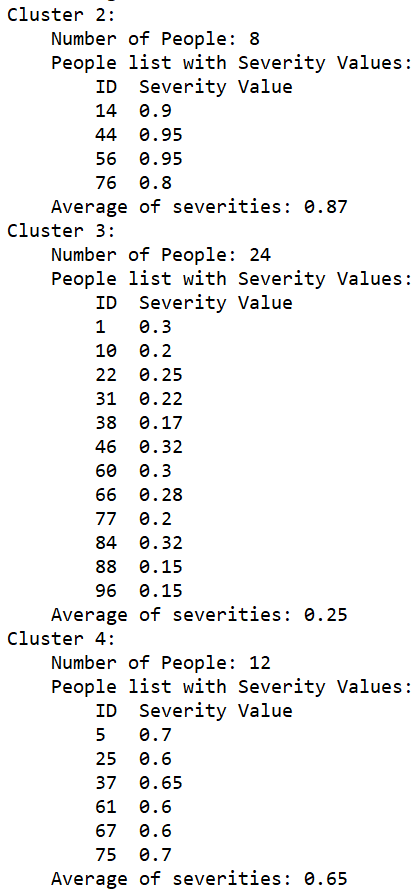
* Cluster 1, 2, 3



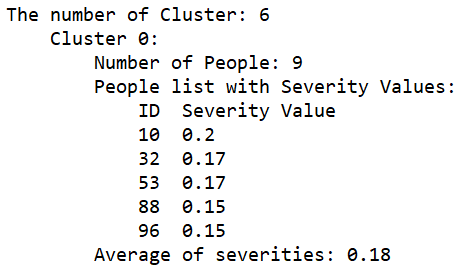
* The number of Cluster: 5
* Cluster 0, 1



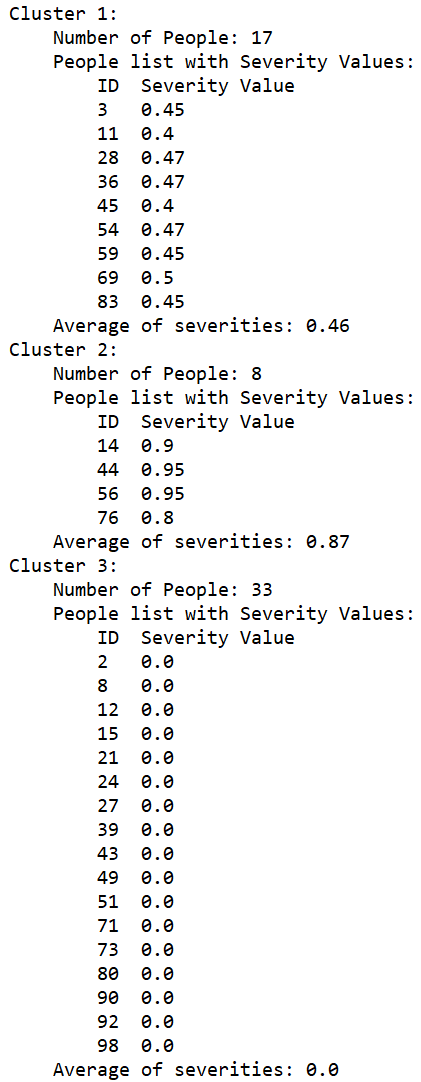
* Cluster 2, 3, 4



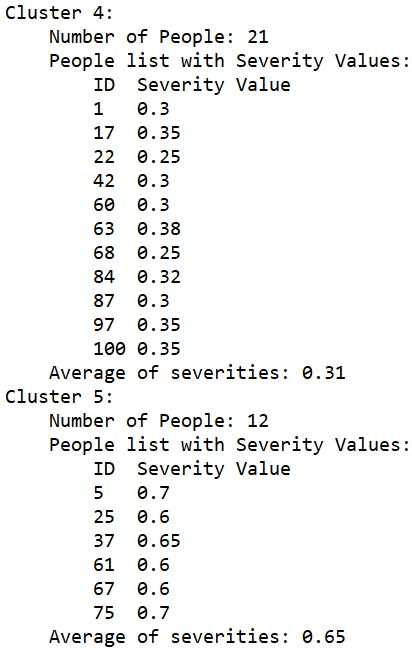
* The number of Cluster: 6
* Cluster 0



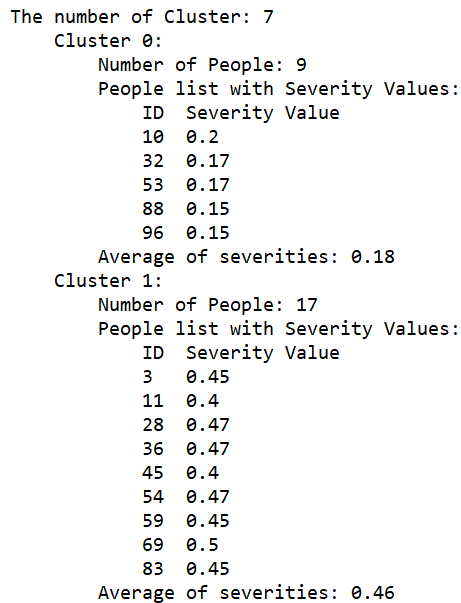
* Cluster 1, 2, 3



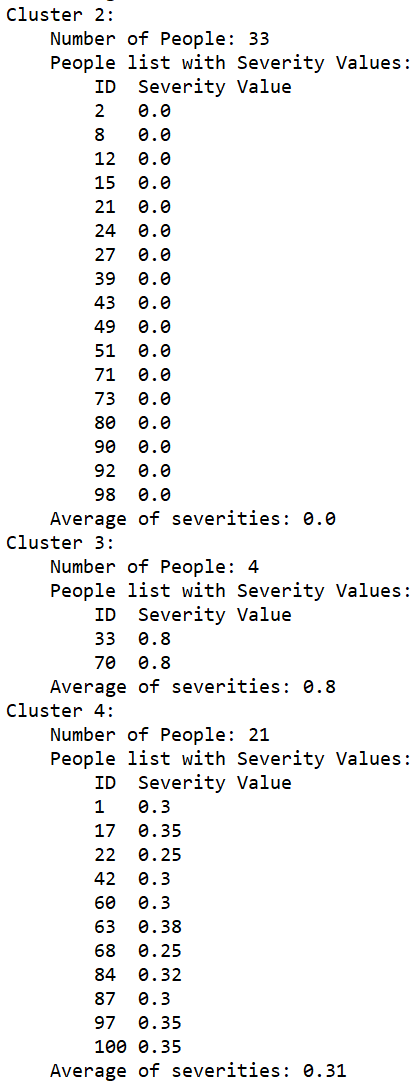
* Cluster 4, 5



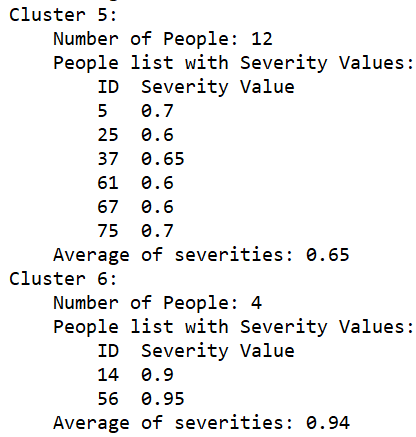
* The number of Cluster: 7
* Cluster 0, 1



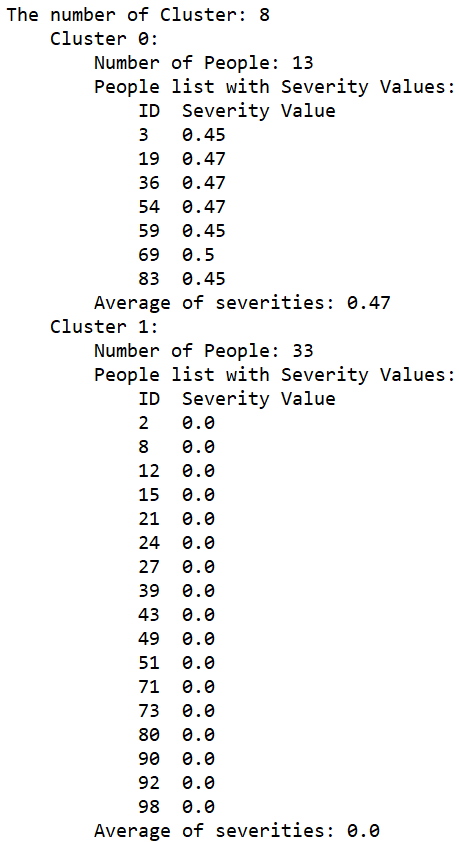
* Cluster 2, 3, 4



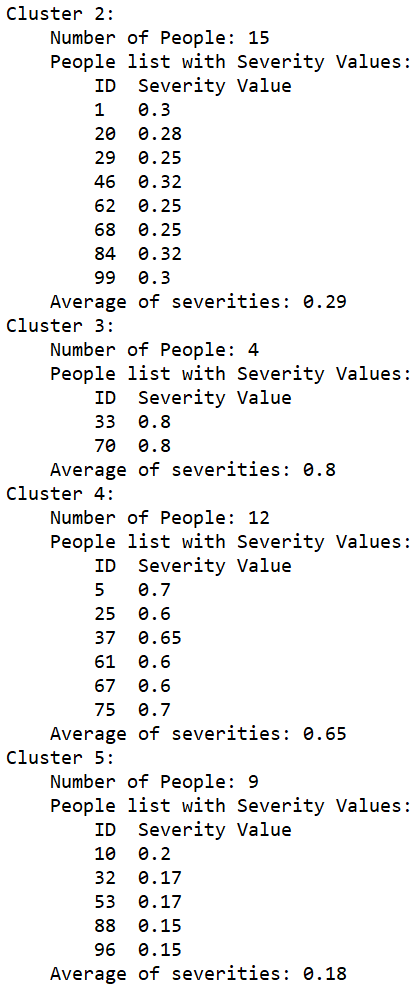
* Cluster 5, 6



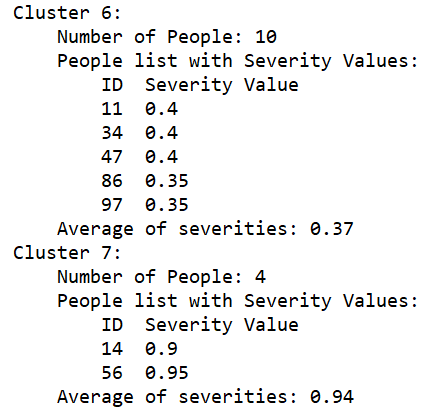
* The number of Cluster: 8
* Cluster 0, 1



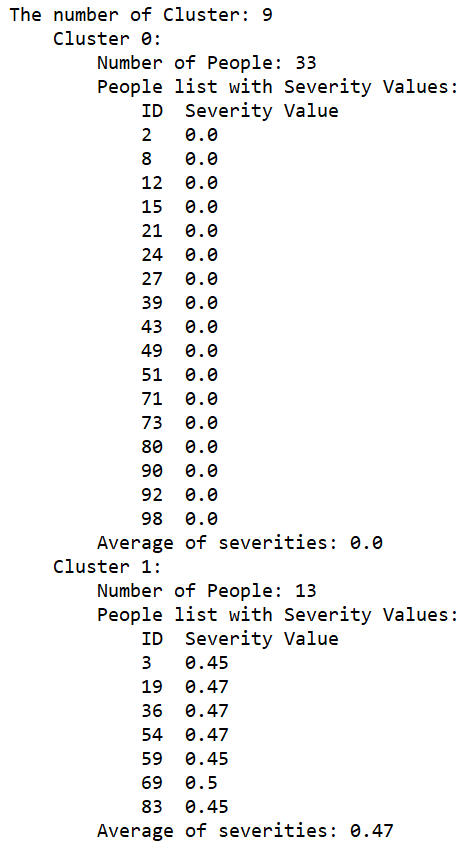
* Cluster 2, 3, 4, 5



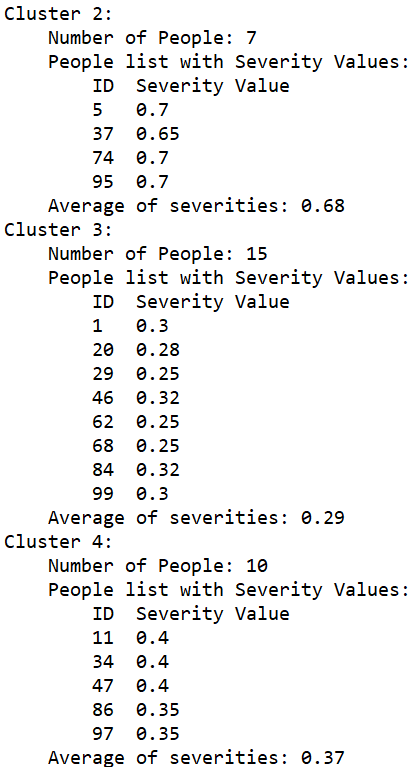
* Cluster 6, 7



* The number of Cluster: 9
* Cluster 0, 1



* Cluster 2, 3, 4



* Cluster 5, 6, 7, 8

