한 파일에서 군집 골라내기

1. csv 파일에서

```
      frameNum  pointNum   Range  Azimuth  Elevation  Doppler       SNR
```

총 6개의 column을 가진 데이터를 받아와서 DataFrame에 저장한다.

frameNum: 55 밀리초 단위의 프레임을 1씩 증가하는 순서로 저장,

pointNum: 각 프레임에서 센서를 통해 얻어진 point를 나타내는 단위이다. point 여러개가 모여서 움직이는 객체를 나타낸다. 한 frameNum에는 여러 pointNum들이 나타난다.

Range, Azimuth, Elevation: 각 point의 위치를 나타내는 변수이다.

Doppler: 각 point의 속도와 관련한 변수

SNR: 각 point의 신호의 세기에 관련한 변수

- 센서 오작동으로 인해 DataFrame의 frameNum이 순서대로 증가하지 않거나 1부터 시작하지 않는 경우가 발생 → frameNum을 1부터 1씩 증가시키게 정렬

```
#frameNum 1부터 1씩 증가시키게 정렬
data = pd.read_csv("/content/cloud_2024_12_31_10_59_4.csv")
unique_frame_numbers = {frame: idx + 1 for idx, frame in enumerate(data['frameNum'].unique())}
data['frameNum'] = data['frameNum'].map(unique_frame_numbers)
```

data에 csv 파일을 DataFrame으로 저장한다.

그리고 frameNum을 다른 프레임 마다 1부터 시작하게 다 다른 수로 저장한다.

- 한 프레임 내에서 여러 cluster가 나올 때 이들을 알아내야함

```
#한 프레임 내에서 클러스터링
clustered_data = pd.DataFrame(columns = data.columns)
for frame, group in data.groupby("frameNum"):
  group_data = group[["Range", "Azimuth", "Elevation"]]
  scaler = StandardScaler()
  scaled_data = scaler.fit_transform(group_data)

  dbscan = DBSCAN(eps=1, min_samples=3)
  clusters = dbscan.fit_predict(scaled_data)
  group['cluster'] = clusters
  group = group.sort_values(by='cluster')
  clustered_data = pd.concat([clustered_data, group])

clustered_data = clustered_data[clustered_data['cluster'] != -1].reset_index().drop(columns='index')
```

위에서 정렬된 data를 가지고 한 프레임 내의 여러 cluster들을 알아내기 위해 data를 groupby('frameNum')하여 각 프레임마다 cluster 구분을 수행한다. 데이터에서 Doppler, SNR 특징은 포

인트마다 너무 큰 차이가 나서 클러스터링에 방해가 되고 클러스터링에는 위치 정보만 이용하는게 좋다고 판단해서 Range, Azimuth, Elevation 3가지 특징만 사용했다. 사이킷런의 DBSCAN을 사용하기 위해 StandardScaler로 정규화하고 DBSCAN 클러스터링 했다. 매개변수는 휴리스틱한 방식으로 찾았는데 수집한 데이터가 많아지면 그에 맞게 바꿔주길 바란다. 그리고 클러스터된 각 프레임을 정렬해준다. 나중에 global_cluster를 맞춰주는데 필요하기 때문이다(뒤에서 설명). 그리고 빈 데이터프레임에 concat하여 하나씩 붙인다. 그렇게 모든 원래 데이터의 모든 행에 cluster column이 추가된 데이터인 clustered_data를 만든다. 이는 각 point들에 대해 이 데이터가 한 프레임에서 어떤 cluster에 들어가있는지를 나타낸다. 맨 마지막 줄은 이상치로 감지된 클러스터 포인트들을 제외하는 행이다. 나중에는 이 코드를 빼야한다. 왜냐하면 우리의 목표가 이상치로 감지된 포인트들 까지 제대로 레이블링을해서 쓸 수 있는 데이터로 만들어야 하기 때문이다.

```
clustered_data['x'] = clustered_data['Range'] * np.cos(clustered_data['Elevation']) * np.cos(clustered_data['Azimuth'])
clustered_data['y'] = clustered_data['Range'] * np.cos(clustered_data['Elevation']) * np.sin(clustered_data['Azimuth'])
clustered_data['z'] = clustered_data['Range'] * np.sin(clustered_data['Elevation'])
```

Range, Elevation, Azimuth 값들을 변형 식을 이용해서 x,y,z 컬럼을 만들었다.

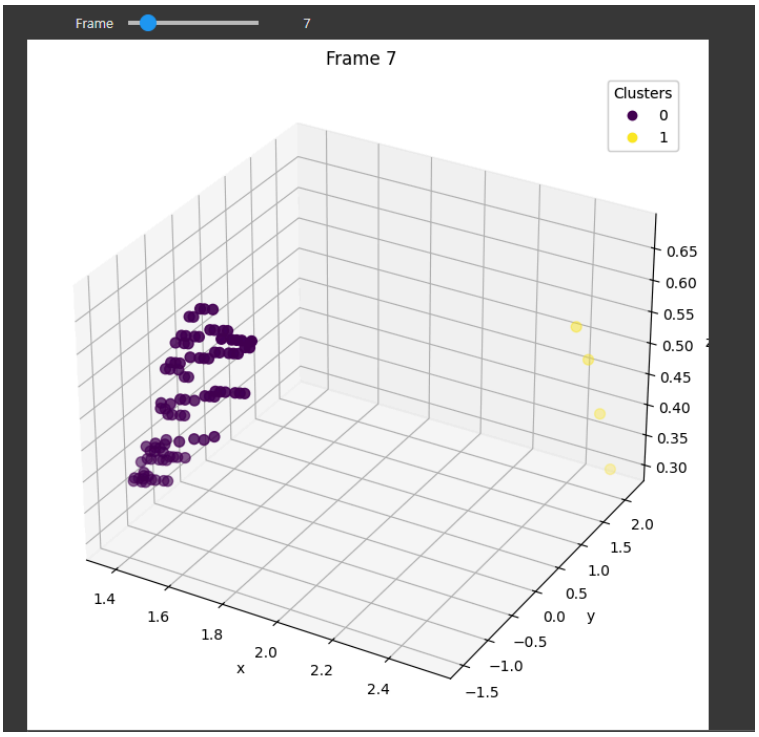
```
def plot_frame(frame):
  fig = plt.figure(figsize=(8, 8))
  ax = fig.add_subplot(111, projection='3d')
  clustered_data_frame = clustered_data[clustered_data['frameNum'] == frame]
  df_scale = clustered_data_frame[["x", "y", "z", "cluster"]]
  scatter = ax.scatter(
      df_scale['x'],
      df_scale['y'],
      df_scale['z'],
      c=df_scale['cluster'],
      cmap='viridis',
      marker='o',
      s=50)

  legend = ax.legend(*scatter.legend_elements(), title="Clusters")
  ax.add_artist(legend)
  ax.set_xlabel('x')
  ax.set_ylabel('y')
  ax.set_zlabel('z')
  ax.set_title(f'Frame {frame}')
  plt.show()

#슬라이더
frame_slider = widgets.IntSlider(
    value=1,
    min=clustered_data["frameNum"].min(),
    max=clustered_data["frameNum"].max(),

    step=1,
    description='Frame'
)
interact(plot_frame, frame=frame_slider)
```

각 프레임별로 point가 각 프레임에서 어떤 형태로 분포되어 있는지 시각화 하는 코드이다.



이런 식으로 슬라이더를 사용해서 데이터의 분포를 확인했다.

- clustered_data 데이터프레임의 모습이다.

	frameNum	pointNum	Range	Azimuth	Elevation	Doppler	SNR	cluster	x	y	z
0	1	0	2.46575	-1.17	0.07	0.06972	9.240000	0.0	0.959661	-2.264781	0.172462
1	1	61	2.59225	-0.63	0.28	0.06972	7.120000	0.0	2.013036	-1.467734	0.716383
2	1	60	2.52900	-0.63	0.25	-0.20888	27.119999	0.0	1.979974	-1.443628	0.625685
3	1	59	2.46575	-0.63	0.25	-0.20888	52.639999	0.0	1.930455	-1.407523	0.610036
4	1	58	2.27625	-0.61	0.17	-0.20888	30.999999	0.0	1.838829	-1.285192	0.385101
...	...	...	...	...	...	...	...	...	...	...	...
3564	43	76	3.79350	0.60	0.12	0.55664	18.120000	1.0	3.108395	2.126568	0.454128
3565	43	77	3.85675	0.60	0.12	0.48720	6.360000	1.0	3.160222	2.162024	0.461700
3566	43	78	3.60400	0.61	0.15	0.55664	26.999999	1.0	2.920841	2.041431	0.538575
3567	43	70	3.66725	0.59	0.15	0.20888	34.639999	1.0	3.013050	2.017404	0.548027
3568	43	101	3.73025	0.68	0.15	0.48720	19.520000	1.0	2.867971	2.319217	0.557442

- 위에서 한 방법은 각 프레임에서의 cluster가 다음 프레임에서의 cluster와 아무 연관이 없다는 문제가 있다. 이를 처리하기 위해 각 프레임마다 클러스터들이 이전 프레임의 어떤 클러스터와 연결되어 있고 이를 나타내기로 했다.

```
clustered_data_agg = clustered_data.groupby(['frameNum', 'cluster']).agg({'frameNum': 'first',
                                                                          'cluster': 'first',
                                                                          'Range': 'mean',
                                                                          'Azimuth': 'mean',
                                                                          'Elevation': 'mean',
                                                                          'Doppler': 'mean',
                                                                          'SNR': 'mean'}).reset_index(drop=True)
```

클러스터된 데이터를 'frameNum', 그 안에서 'cluster' 기준으로 나머지 특징들을 평균한다. 그렇게 한 프레임에서 각 클러스터를 대표하는 값들을 가진 행들을 담은 clustered_data_agg를 만든다

	frameNum	cluster	x	y	z	global_cluster
0	1	0.0	1.705726	-1.601542	0.410367	0.0
1	1	1.0	4.184639	1.469327	0.343905	1.0
2	1	2.0	2.834504	1.657704	0.651839	2.0
3	2	0.0	1.887561	-1.477973	0.454277	0.0
4	2	1.0	4.148217	1.509764	0.703590	1.0
...	...	...	...	...	...	...
89	41	1.0	2.863386	1.968592	0.513841	19.0
90	42	0.0	2.069335	-1.096709	0.559938	17.0
91	42	1.0	3.051533	1.883823	0.552724	19.0
92	43	0.0	2.090437	-1.097339	0.675624	17.0
93	43	1.0	3.042724	2.068884	0.518595	19.0

하고자 하는 것은 원래 clustered_data의 각 행의 클러스터가 이전 프레임의 n번째 클러스터('global cluster' 값이 n)와 이어졌을 때 그 n을 column 'global_cluster'에 담는 global_clustered_data 데이터프레임을 만드는 것이다. 만약 새로운 클러스터로 생각되는 클러스터가 생성된다면 지금 까지 없던 클러스터 번호를 지금 까지 있던 최대 클러스터 번호에 1을 더해서 할당한다.

```python
import numpy as np

def kth_smallest_value(matrix, k):
    flattened = np.ravel(matrix)
    sorted_values = np.sort(flattened)
    return sorted_values[k]

from scipy.spatial import distance
clustered_data_agg = clustered_data.groupby(['frameNum', 'cluster']).agg({'frameNum': 'first',
                                                                          'cluster': 'first',
                                                                          'x': 'mean',
                                                                          'y': 'mean',
                                                                          'z': 'mean',
                                                                          }).reset_index(drop=True)
global_clustered_data = clustered_data_agg[clustered_data_agg['frameNum'] == 1]
global_clustered_data['global_cluster'] = global_clustered_data['cluster']
frame_len = len(clustered_data_agg['frameNum'].unique())
generated_cluster_n = int(global_clustered_data['global_cluster'].max()) + 1

for i in range(1, frame_len):
  current_f = global_clustered_data[global_clustered_data['frameNum'] == i].reset_index().drop(columns='index')
  next_f = clustered_data_agg[clustered_data_agg['frameNum'] == i + 1].reset_index().drop(columns='index')
  used_current_f = [0 for i in range(len(current_f))]
  distance_matrix = distance.cdist(current_f[['x', 'y', 'z']], next_f[['x', 'y', 'z']], metric='euclidean')
  for j in range(distance_matrix.size):
    min_value = kth_smallest_value(distance_matrix, j)
    [row], [col] = np.where(distance_matrix == min_value)
    if (used_current_f[row] == 1):
      continue
    used_current_f[row] = 1
    if (min_value >= 3.5):
      next_f.loc[col, 'global_cluster'] = None
    else:
      next_f.loc[col, 'global_cluster'] = current_f.loc[row, 'global_cluster']
  g_arr = [i for i in range (generated_cluster_n, generated_cluster_n + next_f['global_cluster'].isna().sum())]
  generated_cluster_n = generated_cluster_n + next_f['global_cluster'].isna().sum()
  next_f.loc[next_f['global_cluster'].isna(), 'global_cluster'] = g_arr[:]
  global_clustered_data = pd.concat([global_clustered_data, next_f]).reset_index().drop(columns='index')
```

알고리즘 설명: global_clustered_data에 첫번째 frameNum의 클러스터들을 채우고 시작. 반복문 안에서 global_clustered_data의 i번째 frame에 해당하는 행들과 clustered_data_agg의 i + 1번째 frame에 해당하는 행들을 뽑아낸다. 전자는 current_f라는 변수에, 후자는 next_f라는 변수에 저장한다. 이제 next_f라는 변수에 있는 클러스터들에 대해 current_f에 있는 글로벌 클러스터를 연결 시켜주면 된다.

연결을 하는 방법은 예시로 설명하겠다. current_f에 3개의 global_cluster가 각각 1, 2, 3인 클러스터들이 있고 next_f에는 global_cluster가 아직 할당되지 않은 2개의 클러스터가 있다고 하자. 사이킷런의 distance 함수를 써서 current_f의 각 클러스터 중심의 x,y,z와 next_f의 각 클러스터 중심의 x,y,z의 거리 행렬을 구한다. 거기서 거리가 작은 순서대로 할당을 한다. next_f의 두번째 클러스터가 current_f의 global_cluster가 1인 클러스터와 거리 행렬의 값이 가장 작다고 하자. 그러면 next_f의 두번째 클러스터의 global_cluster 값을 1로 하고 다음으로 거리 행렬의 값이 가장 작은 값으로 넘어가서 계속 할당하는 것이다. 그런데 이렇게 하면 next_f의 클러스터들이 current_f의 클러스터보다 많을 때 무조건 원래 있는 next_f의 클러스터에 할당이 되는 문제가 생긴다. 이를 막기 위해 거리 행렬의 값이 일정 이상이면 global_cluster 할당을 하지 못하게 하는 코드를 추가했다. min_value >= 3.5가 이 부분인데 이 값도 휴리

스틱하게 찾아서 넣었다. 그리고 global_cluster가 할당 되지 않은 클러스터들은 지금까지 나오지 않은 클러스터 값으로 나오게 하도록 만들었다. 예를 들어 지금까지 나온 클러스터중 가장 큰 클러스터가 5면 아직 global_cluster가 할당되지 않은 global_cluster를 6부터 할당하는 것이다.

	frameNum	cluster	x	y	z	global_cluster
0	1	0.0	1.705726	-1.601542	0.410367	0.0
1	1	1.0	4.184639	1.469327	0.343905	1.0
2	1	2.0	2.834504	1.657704	0.651839	2.0
3	2	0.0	1.887561	-1.477973	0.454277	0.0
4	2	1.0	4.148217	1.509764	0.703590	1.0
...	...	...	...	...	...	...
89	41	1.0	2.863386	1.968592	0.513841	19.0
90	42	0.0	2.069335	-1.096709	0.559938	17.0
91	42	1.0	3.051533	1.883823	0.552724	19.0
92	43	0.0	2.090437	-1.097339	0.675624	17.0
93	43	1.0	3.042724	2.068884	0.518595	19.0

이렇게 나온 global_cluster 컬럼이 추가된 데이터이다. 이전 프레임에서의 연결되지 않은 클러스터는 계속해서 새로운 번호로 할당되기 때문에 클러스터 번호가 계속해서 늘어날 것이다.

```
clustered_data = pd.merge(clustered_data, global_clustered_data[['cluster','frameNum','global_cluster']], on=['cluster', 'frameNum'], how='left')
```

```
clustered_data.drop(columns=['cluster'])
```

	frameNum	pointNum	Range	Azimuth	Elevation	Doppler	SNR	x	y	z	global_cluster
0	1	0	2.46575	-1.17	0.07	0.06972	9.240000	0.959661	-2.264781	0.172462	0.0
1	1	61	2.59225	-0.63	0.28	0.06972	7.120000	2.013036	-1.467734	0.716383	0.0
2	1	60	2.52900	-0.63	0.25	-0.20888	27.119999	1.979974	-1.443628	0.625685	0.0
3	1	59	2.46575	-0.63	0.25	-0.20888	52.639999	1.930455	-1.407523	0.610036	0.0
4	1	58	2.27625	-0.61	0.17	-0.20888	30.999999	1.838829	-1.285192	0.385101	0.0
...	...	...	...	...	...	...	...	...	...	...	...
3564	43	76	3.79350	0.60	0.12	0.55664	18.120000	3.108395	2.126568	0.454128	19.0
3565	43	77	3.85675	0.60	0.12	0.48720	6.360000	3.160222	2.162024	0.461700	19.0
3566	43	78	3.60400	0.61	0.15	0.55664	26.999999	2.920841	2.041431	0.538575	19.0
3567	43	70	3.66725	0.59	0.15	0.20888	34.639999	3.013050	2.017404	0.548027	19.0
3568	43	101	3.73025	0.68	0.15	0.48720	19.520000	2.867971	2.319217	0.557442	19.0

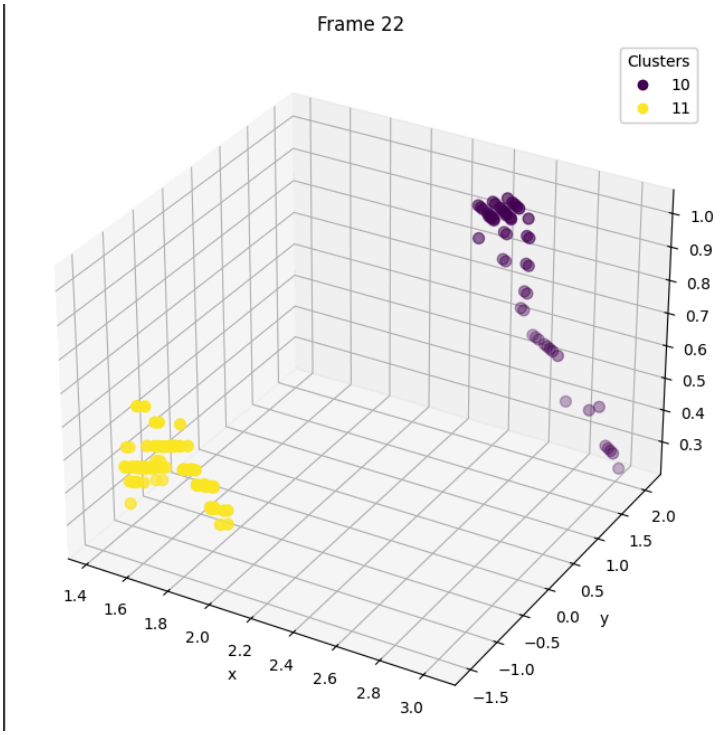
3569 rows × 11 columns

이 과정을 통해 모든 pointNum들에 global_cluster를 할당해준다.

```python
def plot_frame(frame):
  fig = plt.figure(figsize=(8, 8))
  ax = fig.add_subplot(111, projection='3d')
  clustered_data_frame = clustered_data[clustered_data['frameNum'] == frame]
  df_scale = clustered_data_frame[["x", "y", "z", "global_cluster"]]
  scatter = ax.scatter(
      df_scale['x'],
      df_scale['y'],
      df_scale['z'],
      c=df_scale['global_cluster'],
      cmap='viridis',
      marker='o',
      s=50)

  legend = ax.legend(*scatter.legend_elements(), title="Clusters")
  ax.add_artist(legend)
  ax.set_xlabel('x')
  ax.set_ylabel('y')
  ax.set_zlabel('z')
  ax.set_title(f'Frame {frame}')
  plt.show()
#슬라이더
frame_slider = widgets.IntSlider(
    value=1,
    min=clustered_data["frameNum"].min(),
    max=clustered_data["frameNum"].max(),
    step=1,
    description='Frame'
)
```

그러면 위와 같은 코드로 global_cluster을 보는 plot을 그릴 수 있다.



이것이 우리가 사용했던 클러스터링 방식이다.