**# # 패키지 불러오기**

import numpy as np

import pandas as pd

import json

import datetime as dt

import matplotlib.pyplot as plt

import matplotlib.ticker as ticker

import matplotlib.cm as cm

import math

get\_ipython().run\_line\_magic('matplotlib', 'inline')

import seaborn as sns

import seaborn as sns

plt.style.use('seaborn-whitegrid')

import missingno

import re

import glob

import os

from scipy import stats

from scipy.integrate import trapz

import missingno as msno

import sys

import warnings

warnings.filterwarnings('ignore')

pd.set\_option('max\_columns', 10, 'max\_rows', 5, 'max\_colwidth', 10)

# 전처리

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

from sklearn.preprocessing import StandardScaler, MinMaxScaler

from sklearn.preprocessing import Binarizer

from sklearn.decomposition import PCA

# 알고리즘(군집)

from sklearn.cluster import KMeans

from sklearn.cluster import MeanShift

from sklearn.cluster import estimate\_bandwidth # 최적의 bandwidth값을 estimate\_bandwidth()로 계산 한 뒤에 다시 군집화 수행

from sklearn.cluster import DBSCAN

from sklearn.mixture import GaussianMixture

## 평가

from sklearn.metrics import silhouette\_score, silhouette\_samples

from sklearn.metrics.pairwise import cosine\_similarity

# # 데이터 읽기

from sklearn.datasets import load\_iris

iris = load\_iris()

X\_features = pd.DataFrame(data=iris.data, columns=['sepal\_length','sepal\_width','petal\_length','petal\_width'])

X\_origin\_df = X\_features.copy()

X\_features

**# ## 전처리**

**# ### Scaler**

# 표준화

scaler = StandardScaler()

X\_features = scaler.fit\_transform(X\_features)

# 표준화 후 데이터 프레임으로 변경

X\_features = pd.DataFrame(X\_features)

X\_features.columns = X\_origin\_df.columns

X\_features

**# MinMax**

scaler = MinMaxScaler()

X\_features = scaler.fit\_transform(X\_features)

# 표준화 후 데이터 프레임으로 변경

X\_features = pd.DataFrame(X\_features)

X\_features.columns = X\_origin\_df.columns

X\_features

**# Log1p**

X\_features = np.log1p(X\_features)

X\_features

**# ### PCA를 활용한 x, y 좌표값 추출**

pca = PCA(n\_components=2)

pca\_transformed = pca.fit\_transform(X\_features)

X\_features['pca\_x'] = pca\_transformed[:,0]

X\_features['pca\_y'] = pca\_transformed[:,1]

X\_features.head(3)

**# # 계층적 군집화(군집수 구하기)**

import scipy.cluster.hierarchy as shc

plt.figure(figsize=(20, 14))

plt.title("Customer Dendograms")

dend = shc.dendrogram(shc.linkage(X\_features, method='ward'))

**# # 머신러닝 모델 만들기(비계층적 군집화)**

**# ## Kmeans**

kmeans = KMeans(n\_clusters=3, init='k-means++', max\_iter=300,random\_state=0)

kmeans\_cluster\_labels = kmeans.fit\_predict(X\_features)

# X\_df['kmeans\_cluster\_label'] = kmeans\_cluster\_labels

X\_features['kmeans\_cluster\_label'] = kmeans.labels\_

print('군집 유형 :',np.unique((kmeans.labels\_)), '\n')

print(kmeans.labels\_)

print(kmeans.predict(X\_origin\_df))

plt.scatter(x=X\_features.loc[:, 'pca\_x'], y=X\_features.loc[:, 'pca\_y'], c=X\_features['kmeans\_cluster\_label'],

cmap = 'winter\_r')

#setosa를 세모, versicolor를 네모, virginica를 동그라미로 표시

# markers=['o', 's', '^', 'x', '\*']

markers=['^', 's', 'o']

#pca\_component\_1 을 x축, pc\_component\_2를 y축으로 scatter plot 수행.

for i, marker in enumerate(markers):

x\_axis\_data = X\_features[X\_features['kmeans\_cluster\_label']==i]['pca\_x']

y\_axis\_data = X\_features[X\_features['kmeans\_cluster\_label']==i]['pca\_y']

plt.scatter(x\_axis\_data, y\_axis\_data, marker=marker,label=iris.target\_names[i])

plt.legend()

plt.xlabel('pca\_component\_1')

plt.ylabel('pca\_component\_2')

plt.show()

**# ## Mean Shift**

meanshift= MeanShift(bandwidth=1.211)

meanshift\_cluster\_labels = meanshift.fit\_predict(X\_features)

X\_features['meanshift\_cluster\_label'] = meanshift\_cluster\_labels

print('군집 유형 :', np.unique((meanshift.labels\_)), '\n')

print(meanshift.labels\_)

print(meanshift.predict(X\_origin\_df))

bandwidth = estimate\_bandwidth(X\_features, quantile=0.25)

# bandwidth = estimate\_bandwidth(X\_df,quantile=0.25)

print('bandwidth 값:', round(bandwidth,3))

plt.scatter(x=X\_features.loc[:, 'pca\_x'], y=X\_features.loc[:, 'pca\_y'], c=X\_features['meanshift\_cluster\_label'],

cmap='winter\_r')

#setosa를 세모, versicolor를 네모, virginica를 동그라미로 표시

markers=['^', 's', 'o']

#pca\_component\_1 을 x축, pc\_component\_2를 y축으로 scatter plot 수행.

for i, marker in enumerate(markers):

x\_axis\_data = X\_features[X\_features['meanshift\_cluster\_label']==i]['pca\_x']

y\_axis\_data = X\_features[X\_features['meanshift\_cluster\_label']==i]['pca\_y']

plt.scatter(x\_axis\_data, y\_axis\_data, marker=marker,label=iris.target\_names[i])

plt.legend()

plt.xlabel('pca\_component\_1')

plt.ylabel('pca\_component\_2')

plt.show()

# ## GaussianMixture

gmm = GaussianMixture(n\_components=3, random\_state=0)

gmm\_cluster\_labels = gmm.fit\_predict(X\_features)

X\_features['gmm\_cluster\_label'] = gmm\_cluster\_labels

print('군집 유형 :', np.unique((gmm.labels\_)), '\n')

print(gmm.labels\_)

print(gmm.predict(X\_origin\_df))

plt.scatter(x=X\_features.loc[:, 'pca\_x'], y=X\_features.loc[:, 'pca\_y'], c=X\_features['gmm\_cluster\_label'],

cmap='winter\_r')

#setosa를 세모, versicolor를 네모, virginica를 동그라미로 표시

markers=['^', 's', 'o']

#pca\_component\_1 을 x축, pc\_component\_2를 y축으로 scatter plot 수행.

for i, marker in enumerate(markers):

x\_axis\_data = X\_features[X\_features['gmm\_cluster\_label']==i]['pca\_x']

y\_axis\_data = X\_features[X\_features['gmm\_cluster\_label']==i]['pca\_y']

plt.scatter(x\_axis\_data, y\_axis\_data, marker=marker,label=iris.target\_names[i])

plt.legend()

plt.xlabel('pca\_component\_1')

plt.ylabel('pca\_component\_2')

plt.show()

**# ## DBSCAN**

dbscan = DBSCAN(eps=0.6, min\_samples=8, metric='euclidean')

dbscan\_labels = dbscan.fit\_predict(X\_features)

X\_features['dbscan\_cluster\_label'] = dbscan\_labels

print('군집 유형 :',np.unique((dbscan.labels\_)), '\n')

print(dbscan.labels\_)

print(dbscan.predict(X\_origin\_df))

plt.scatter(x=X\_features.loc[:, 'pca\_x'], y=X\_features.loc[:, 'pca\_y'], c=X\_features['dbscan\_cluster\_label'],

cmap='winter\_r')

#setosa를 세모, versicolor를 네모, virginica를 동그라미로 표시

markers=['^', 's', 'o']

#pca\_component\_1 을 x축, pc\_component\_2를 y축으로 scatter plot 수행.

for i, marker in enumerate(markers):

x\_axis\_data = X\_features[X\_features['dbscan\_cluster\_label']==i]['pca\_x']

y\_axis\_data = X\_features[X\_features['dbscan\_cluster\_label']==i]['pca\_y']

plt.scatter(x\_axis\_data, y\_axis\_data, marker=marker,label=iris.target\_names[i])

plt.legend()

plt.xlabel('pca\_component\_1')

plt.ylabel('pca\_component\_2')

plt.show()

**# # 평가**

**# ## 분류처럼 결과값이 있을 경우 비교(군집화는 원래 없다)**

# 타겟 은 실제 타겟이랑 비교하는 것으로 원래는 가지고 있지 않은 것이다.

X\_features['target'] = iris.target

iris\_result = X\_features.groupby(['target','cluster\_label'])['sepal\_length'].count()

print(iris\_result)

**# ## 실루엣계수 평가**

algo\_col\_list = ['kmeans\_cluster\_label', 'meanshift\_cluster\_label',

'gmm\_cluster\_label', 'dbscan\_cluster\_label']

for col in algo\_col\_list:

print(col)

print('실루엣 스코어 : {0:.3f}'.format(silhouette\_score(X\_origin\_df, X\_features[col])))

print('-------------------------------------------------------------------------------------------------\n')

**# ## 2차원 시각화 평가 1**

# if 'cluster\_label' in X\_df.columns:

# X\_df.drop('cluster\_label', axis=1, inplace=True)

# if 'target' in X\_df.columns:

# X\_df.drop('target', axis=1, inplace=True)

### 클러스터 결과를 담은 DataFrame과 사이킷런의 Cluster 객체등을 인자로 받아 클러스터링 결과를 시각화하는 함수

def visualize\_cluster\_plot(clusterobj, dataframe, label\_name, iscenter=True):

if iscenter :

centers = clusterobj.cluster\_centers\_

unique\_labels = np.unique(dataframe[label\_name].values)

markers=['o', 's', '^', 'x', '\*']

isNoise=False

for label in unique\_labels:

label\_cluster = dataframe[dataframe[label\_name]==label]

if label == -1:

cluster\_legend = 'Noise'

isNoise=True

else :

cluster\_legend = 'Cluster '+str(label)

plt.scatter(x=label\_cluster['pca\_x'], y=label\_cluster['pca\_y'], s=70, edgecolor='k', marker=markers[label], label=cluster\_legend)

if iscenter:

center\_x\_y = centers[label]

plt.scatter(x=center\_x\_y[0], y=center\_x\_y[1], s=250, color='white',

alpha=0.9, edgecolor='k', marker=markers[label])

plt.scatter(x=center\_x\_y[0], y=center\_x\_y[1], s=70, color='k', edgecolor='k', marker='$%d$' % label)

if isNoise:

legend\_loc='upper center'

else: legend\_loc='upper right'

plt.legend(loc=legend\_loc)

plt.show()

# 3개의 Cluster 기반 Kmeans 를 X\_aniso 데이터 셋에 적용

# kmeans = KMeans(3, random\_state=0)

# kmeans\_label = kmeans.fit\_predict(X\_aniso)

# X\_df['kmeans\_cluster\_label'] = kmeans\_label

visualize\_cluster\_plot(kmeans, X\_features, 'kmeans\_cluster\_label',iscenter=True)

**# # 추가 시각화 평가**

**# ### 실루엣계수 시각화 평가**

### 여러개의 클러스터링 갯수를 List로 입력 받아 각각의 실루엣 계수를 면적으로 시각화한 함수 작성

def visualize\_silhouette(cluster\_lists, X\_features):

# 입력값으로 클러스터링 갯수들을 리스트로 받아서, 각 갯수별로 클러스터링을 적용하고 실루엣 개수를 구함

n\_cols = len(cluster\_lists)

# plt.subplots()으로 리스트에 기재된 클러스터링 만큼의 sub figures를 가지는 axs 생성

fig, axs = plt.subplots(figsize=(4\*n\_cols, 4), nrows=1, ncols=n\_cols)

# 리스트에 기재된 클러스터링 갯수들을 차례로 iteration 수행하면서 실루엣 개수 시각화

for ind, n\_cluster in enumerate(cluster\_lists):

# KMeans 클러스터링 수행하고, 실루엣 스코어와 개별 데이터의 실루엣 값 계산.

clusterer = KMeans(n\_clusters = n\_cluster, max\_iter=500, random\_state=0)

cluster\_labels = clusterer.fit\_predict(X\_features)

sil\_avg = silhouette\_score(X\_features, cluster\_labels)

sil\_values = silhouette\_samples(X\_features, cluster\_labels)

y\_lower = 10

axs[ind].set\_title('Number of Cluster : '+ str(n\_cluster)+'\n' 'Silhouette Score :' + str(round(sil\_avg,3)) )

axs[ind].set\_xlabel("The silhouette coefficient values")

axs[ind].set\_ylabel("Cluster label")

axs[ind].set\_xlim([-0.1, 1])

axs[ind].set\_ylim([0, len(X\_features) + (n\_cluster + 1) \* 10])

axs[ind].set\_yticks([]) # Clear the yaxis labels / ticks

axs[ind].set\_xticks([0, 0.2, 0.4, 0.6, 0.8, 1])

# 클러스터링 갯수별로 fill\_betweenx( )형태의 막대 그래프 표현.

for i in range(n\_cluster):

ith\_cluster\_sil\_values = sil\_values[cluster\_labels==i]

ith\_cluster\_sil\_values.sort()

size\_cluster\_i = ith\_cluster\_sil\_values.shape[0]

y\_upper = y\_lower + size\_cluster\_i

color = cm.nipy\_spectral(float(i) / n\_cluster)

axs[ind].fill\_betweenx(np.arange(y\_lower, y\_upper), 0, ith\_cluster\_sil\_values, facecolor=color, edgecolor=color, alpha=0.7)

axs[ind].text(-0.05, y\_lower + 0.5 \* size\_cluster\_i, str(i))

y\_lower = y\_upper + 10

axs[ind].axvline(x=sil\_avg, color="red", linestyle="--")

visualize\_silhouette([2,3,4,5],X\_origin\_df)

**# ### 2차원 시각화 평가 3**

### 여러개의 클러스터링 갯수를 List로 입력 받아 각각의 클러스터링 결과를 시각화

def visualize\_cluster\_plot\_multi(cluster\_lists, X\_features):

# plt.subplots()으로 리스트에 기재된 클러스터링 만큼의 sub figures를 가지는 axs 생성

n\_cols = len(cluster\_lists)

fig, axs = plt.subplots(figsize=(4\*n\_cols, 4), nrows=1, ncols=n\_cols)

# 입력 데이터의 FEATURE가 여러개일 경우 2차원 데이터 시각화가 어려우므로 PCA 변환하여 2차원 시각화

pca = PCA(n\_components=2)

pca\_transformed = pca.fit\_transform(X\_features)

dataframe = pd.DataFrame(pca\_transformed, columns=['PCA1','PCA2'])

# 리스트에 기재된 클러스터링 갯수들을 차례로 iteration 수행하면서 KMeans 클러스터링 수행하고 시각화

for ind, n\_cluster in enumerate(cluster\_lists):

# KMeans 클러스터링으로 클러스터링 결과를 dataframe에 저장.

clusterer = KMeans(n\_clusters = n\_cluster, max\_iter=500, random\_state=0)

cluster\_labels = clusterer.fit\_predict(pca\_transformed)

dataframe['cluster']=cluster\_labels

unique\_labels = np.unique(clusterer.labels\_)

markers=['o', 's', '^', 'x', '\*']

# 클러스터링 결과값 별로 scatter plot 으로 시각화

for label in unique\_labels:

label\_df = dataframe[dataframe['cluster']==label]

if label == -1:

cluster\_legend = 'Noise'

else :

cluster\_legend = 'Cluster '+str(label)

axs[ind].scatter(x=label\_df['PCA1'], y=label\_df['PCA2'], s=70, edgecolor='k', marker=markers[label], label=cluster\_legend)

axs[ind].set\_title('Number of Cluster : '+ str(n\_cluster))

axs[ind].legend(loc='upper right')

plt.show()

visualize\_cluster\_plot\_multi([2,3,4,5], X\_origin\_df)