각 feature 값을 pandas에 저장하고, 범위별로 양자화했으면 좋겠어.

date는 연도-월별로 나눌꺼야. (일자는 사용하지 않아)

avg (temperature),max (temperature),min (temperature) 3가지 features는 5 단위로 쪼갤게. 예를 들어 0~5그룹, ....

avg (humidity),max (humidity),min (humidity) 3가지 features는 5단위로 쪼갤게.

power는 10단위로 쪼갤게.

label은 정상, 비정상이고 정상일 경우 0으로, 비정상인 경우 1로 표시해야 해.

avg(wind speed),avg(local pressure),avg(cloud cover) 3가지 features는 0.5 단위로 쪼갤게.

sun\_rise,sun\_set 2가지 features는 시간:분 단위로 되어 있어. 이거는 분을 10분 단위로 쪼갤게.

sun\_max는 시간:분:초 단위로 되어 있어. 이거는 분을 5분 단위로 쪼갤게.

day\_duration,night\_duration 2가지 features는 'n days 시간:분:초' 단위로 되어 있어. n은 무시하고, 분을 5분 단위로 쪼갤게.

day\_avg\_temperature\_app,night\_avg\_temperature\_app 2가지 features는 5 단위로 쪼갤게.

Nitrogen Dioxide Concentration (ppm),Ozone Concentration (ppm) 2가지 features는 0.005 단위로 쪼갤게.

Carbon Monoxide Concentration (ppm)는 0.1 단위로 쪼갤게.

Sulfur Dioxide Concentration (ppm)는 0.001 단위로 쪼갤게.

Particulate Matter (㎍/㎥),Fine Particulate Matter (㎍/㎥),AQI 3가지 features는 10 단위로 쪼갤게.

이렇게 쪼개서 pandas에 각 features 값 대신 그룹명으로 표기되도록 해 줘. 그룹명은 작은 값에서 큰 값으로 1부터 시작하는 자연수로 정하는 코드를 작성해 줘.

참고로 csv 파일 일부를 제시할게:

date,avg (temperature),max (temperature),min (temperature),avg (humidity),max (humidity),min (humidity),power,label,avg(wind speed),avg(local pressure),avg(cloud cover),avg(ground temperature),avg(evaporation volume),sun\_rise,sun\_max,sun\_set,day\_duration,night\_duration,day\_avg\_temperature\_app,night\_avg\_temperature\_app,Nitrogen Dioxide Concentration (ppm),Ozone Concentration (ppm),Carbon Monoxide Concentration (ppm),Sulfur Dioxide Concentration (ppm),Particulate Matter (㎍/㎥),Fine Particulate Matter (㎍/㎥),AQI

2021-01-01,-4.45,1.4,-9.7,64.87,75,52,242.75,0,2.7,1019.8,1.3,6.3,1.6,7:47,12:35:14,17:24,0 days 09:37:00,0 days 14:23:00,-1.525,-7.074999999999999,0.023,0.013,0.5,0.003,32.0,13.0,93.0

2021-01-02,-4.79,-1.5,-8.3,38.43,50,31,229.78,0,2.5,1021.0,0.0,6.2,1.3,7:47,12:35:42,17:25,0 days 09:38:00,0 days 14:22:00,-3.145,-6.545,0.015,0.019,0.4,0.003,44.0,9.0,122.0

2021-01-03,-5.59,-2.3,-9.0,44.39,58,30,252.5,0,1.8,1019.5,0.0,6.1,1.3,7:47,12:36:10,17:25,0 days 09:38:00,0 days 14:22:00,-3.945,-7.295,0.017,0.017,0.4,0.003,46.0,11.0,127.0

2021-01-04,-3.67,0.2,-8.3,48.91,69,40,298.45,0,1.9,1019.5,1.9,6.0,1.5,7:47,12:36:38,17:26,0 days 09:39:00,0 days 14:21:00,-1.735,-5.985,0.033,0.008,0.5,0.003,47.0,19.0,129.0

2021-01-05,-4.96,-2.5,-9.5,53.43,82,31,333.4,0,1.6,1019.6,3.6,5.9,0.9,7:47,12:37:05,17:27,0 days 09:40:00,0 days 14:20:00,-3.73,-7.23,0.018,0.018,0.4,0.003,33.0,12.0,95.0

**1. 이거는 각 feature의 연관성보다 feature 그룹과 label 간의 연관성이 중요하다 생각하기에 apriori의 강점을 포기하고 더 빠르고 compact한 FP-Growth로 진행**

**2. anomal인 부분을 찾아내는 것이 중요하므로 recall 보다는 precision이 중요하다 판단**

**양자화된 구간을 추출하는 과정**

양자화된 구간을 추출하는 과정은 training.csv에 포함된 데이터의 **값 범위**를 이용하여, 우리가 양자화를 할 때 사용할 **구간(bin)을 계산**하는 것입니다.

예를 들어, training.csv에서 'avg (temperature)' 컬럼이 있고, 이 컬럼의 값이 10에서 30 사이일 수 있다고 가정해보겠습니다. 이 값들을 구간(예: 5 단위)으로 나누면, 구간이 [10, 15), [15, 20), [20, 25), [25, 30] 이렇게 나뉘게 됩니다.

get\_bins\_from\_training 함수는 **training.csv에서 값의 최소값과 최대값을 구한 후**, 그 범위에 맞게 구간을 나누는 역할을 합니다.

**양자화 구간 추출 방법**

여기서 핵심은 **값의 최소값**과 **최대값**을 이용해서 양자화 구간을 나눈다는 점입니다. 이 구간은 training.csv에서 얻은 정보에 기반하여 testing.csv에도 동일한 구간을 적용하려는 목적입니다.

**예시로 보겠습니다.**

python

코드 복사

def get\_bins\_from\_training(df\_train, feature, step):

"""Training 데이터에서 feature의 구간을 얻어옵니다."""

min\_val = df\_train[feature].min() # 최소값

max\_val = df\_train[feature].max() # 최대값

bins = np.arange(min\_val, max\_val + step, step) # step 크기로 구간을 나누기

return bins

이 함수는 training.csv에서 특정 컬럼(예: avg (temperature))의 최소값과 최대값을 구하고, 그 범위에 맞춰 구간을 나눕니다. 예를 들어, avg (temperature)가 10에서 30 사이라면, 구간을 np.arange(10, 30+5, 5)로 만들 수 있습니다. 이렇게 하면 [10, 15), [15, 20), [20, 25), [25, 30] 구간이 나옵니다.

**예시**

**training.csv의 'avg (temperature)' 컬럼의 값이 다음과 같다고 가정:**

* 최소값: 10
* 최대값: 30

python

코드 복사

temperature\_bins = get\_bins\_from\_training(df\_train, 'avg (temperature)', 5)

이렇게 하면 temperature\_bins는 [10, 15, 20, 25, 30]과 같은 값으로 반환됩니다. 이 구간을 testing.csv의 같은 컬럼에 적용하여, 값들을 해당 구간으로 분류할 수 있습니다.

**전체 흐름**

1. **training.csv에서 구간 추출**: get\_bins\_from\_training() 함수를 사용하여 각 특성(feature)별로 구간을 추출합니다.
2. **testing.csv에 같은 구간 적용**: 추출한 구간을 testing.csv에 동일하게 적용하여, 양자화된 값을 할당합니다.

따라서, training.csv에서 구간을 추출하는 방법은 **단순히 데이터의 최솟값과 최댓값을 기준으로 범위를 설정**하는 것입니다. training.csv의 데이터 범위가 testing.csv에서도 동일한 방식으로 적용되기 때문에, **양자화된 구간을 유지하면서 새로운 데이터에 동일한 기준을 적용**할 수 있습니다.

이해가 되셨나요? 추가적으로 궁금한 부분이 있으면 언제든지 질문해 주세요!

1. 이걸 우선 pandas를 통해 label=1인 값만 모읍니다.

2. FP-Growth를 적용해서 label=1인 그룹을 딕셔너리로 모읍니다. feature:value의 형태로 모읍니다. confidence=0.7로 가정합니다.

클러스터링을 통해 데이터를 그룹화하고, **FP-Growth**와 같은 연관 규칙 알고리즘 대신 다른 클러스터링 방법을 사용하여 **특성 간의 관계**를 찾아내는 방법을 찾고 계신다면, \*\*비지도 학습(unsupervised learning)\*\*에서 사용할 수 있는 다양한 클러스터링 알고리즘들이 있습니다. 각 알고리즘은 서로 다른 특성을 가지며, 데이터의 특성에 따라 적합한 방법이 다를 수 있습니다.

여기서는 **군집화(clustering)** 알고리즘 중 몇 가지를 소개하고, 데이터를 그룹화하는 데 사용할 수 있는 방법을 설명하겠습니다.

**1. K-평균 군집화 (K-Means Clustering)**

K-Means는 가장 널리 사용되는 군집화 알고리즘 중 하나입니다. 이 알고리즘은 데이터를 K개의 군집으로 나누고, 각 군집은 그 군집 내 데이터들의 평균값(centroid)에 의해 정의됩니다. 각 데이터를 가장 가까운 군집의 중심으로 할당하며, 중심이 수렴할 때까지 반복합니다.

**K-평균 군집화 예시:**

python

코드 복사

import pandas as pd

from sklearn.cluster import KMeans

# 데이터 불러오기

df = pd.read\_csv('data.csv')

# 데이터 전처리 (숫자형 데이터만 추출하여 사용)

df\_encoded = df.select\_dtypes(include=['float64', 'int64'])

# K-평균 군집화

kmeans = KMeans(n\_clusters=3, random\_state=42) # 3개의 군집으로 클러스터링

df\_encoded['cluster'] = kmeans.fit\_predict(df\_encoded)

# 군집화 결과 확인

print(df\_encoded.head())

**2. 계층적 군집화 (Hierarchical Clustering)**

계층적 군집화는 **병합(hierarchical)** 방식 또는 **분할(divisive)** 방식으로 데이터를 계층적으로 그룹화하는 방법입니다. 군집을 트리 구조로 만들어서 각 군집이 서로 어떤 관계를 가지는지 쉽게 시각화할 수 있습니다.

**계층적 군집화 예시:**

python

코드 복사

import pandas as pd

from sklearn.cluster import AgglomerativeClustering

# 데이터 불러오기

df = pd.read\_csv('data.csv')

# 데이터 전처리 (숫자형 데이터만 추출하여 사용)

df\_encoded = df.select\_dtypes(include=['float64', 'int64'])

# 계층적 군집화 (Agglomerative Clustering)

agg\_clustering = AgglomerativeClustering(n\_clusters=3) # 3개의 군집으로 클러스터링

df\_encoded['cluster'] = agg\_clustering.fit\_predict(df\_encoded)

# 군집화 결과 확인

print(df\_encoded.head())

**3. DBSCAN (Density-Based Spatial Clustering of Applications with Noise)**

DBSCAN은 **밀도 기반 군집화** 방법으로, 데이터 포인트들이 밀집된 지역을 찾아 군집을 형성합니다. 군집을 형성할 수 없는 점들은 \*\*노이즈(noise)\*\*로 분류되며, 군집의 형태가 비구형일 때 유리합니다.

**DBSCAN 예시:**

python

코드 복사

import pandas as pd

from sklearn.cluster import DBSCAN

# 데이터 불러오기

df = pd.read\_csv('data.csv')

# 데이터 전처리 (숫자형 데이터만 추출하여 사용)

df\_encoded = df.select\_dtypes(include=['float64', 'int64'])

# DBSCAN 군집화

dbscan = DBSCAN(eps=0.5, min\_samples=5) # eps: 거리 기준, min\_samples: 군집의 최소 데이터 수

df\_encoded['cluster'] = dbscan.fit\_predict(df\_encoded)

# 군집화 결과 확인

print(df\_encoded.head())

**4. Gaussian Mixture Model (GMM)**

가우시안 혼합 모델(GMM)은 **확률 기반** 군집화 방법으로, 각 군집을 **가우시안 분포**(정규 분포)로 모델링합니다. GMM은 데이터가 여러 개의 가우시안 분포로부터 나온 것이라고 가정하고 군집을 형성합니다.

**GMM 예시:**

python

코드 복사

import pandas as pd

from sklearn.mixture import GaussianMixture

# 데이터 불러오기

df = pd.read\_csv('data.csv')

# 데이터 전처리 (숫자형 데이터만 추출하여 사용)

df\_encoded = df.select\_dtypes(include=['float64', 'int64'])

# GMM 군집화

gmm = GaussianMixture(n\_components=3) # 3개의 군집

df\_encoded['cluster'] = gmm.fit\_predict(df\_encoded)

# 군집화 결과 확인

print(df\_encoded.head())

**5. Self-Organizing Maps (SOM)**

Self-Organizing Maps(SOM)은 **신경망 기반의 군집화** 알고리즘으로, 고차원 데이터를 저차원으로 **차원 축소**하면서 군집을 찾습니다. SOM은 비선형적인 데이터 관계를 찾는 데 유리합니다. 이 방법은 군집화뿐만 아니라 데이터의 **특성 시각화**에도 유용합니다.

**SOM 예시 (라이브러리 필요):**

python

코드 복사

import pandas as pd

from minisom import MiniSom

# 데이터 불러오기

df = pd.read\_csv('data.csv')

# 데이터 전처리 (숫자형 데이터만 추출하여 사용)

df\_encoded = df.select\_dtypes(include=['float64', 'int64'])

# SOM 모델 생성

som = MiniSom(x=10, y=10, input\_len=df\_encoded.shape[1], sigma=1.0, learning\_rate=0.5)

som.train(df\_encoded.values, 100)

# 군집화 결과 추출

clusters = som.labels\_

# 군집화 결과 확인

df\_encoded['cluster'] = clusters

print(df\_encoded.head())

**클러스터링 알고리즘 선택 기준**

* **K-Means**: 데이터가 \*\*구형(cluster shape)\*\*으로 분포할 때 유리합니다. 미리 군집의 수(K)를 정해야 하며, 군집의 수를 어떻게 설정할지에 대한 고민이 필요합니다.
* **계층적 군집화**: 군집의 **계층적 관계**를 파악하고 싶을 때 유용합니다. 군집 수를 미리 정하지 않아도 되며, 트리 구조로 군집 관계를 시각화할 수 있습니다.
* **DBSCAN**: 군집의 형태가 **비구형**일 때 유리하며, 노이즈를 분리할 수 있는 장점이 있습니다.
* **GMM**: 군집이 **확률적**으로 분포한다고 가정할 때 유리합니다. 군집이 가우시안 분포에 가까운 데이터에 잘 적용됩니다.
* **SOM**: **비선형적 관계**를 파악하고 **고차원 데이터를 저차원으로 축소**하려는 경우 적합합니다.

**클러스터링 후 연관 규칙 찾기**

군집화를 통해 비슷한 특성을 가진 데이터 포인트를 찾은 후, 각 군집 내에서 FP-Growth와 같은 **연관 규칙 알고리즘**을 사용할 수 있습니다. 예를 들어, 각 군집에 대해 특정 특성들 간의 **연관 규칙을 추출**할 수 있습니다.

**결론**

* **FP-Growth**와 같은 연관 규칙 알고리즘을 사용하기 위해서는 데이터가 **불리언 형식**이어야 하기 때문에, 다른 방식으로 **클러스터링**을 사용하여 데이터를 그룹화하는 방법을 제시해 드렸습니다.
* 군집화 후 각 군집에 대해 연관 규칙을 추출하거나, 군집 내 특성 간의 관계를 분석하는 방법을 적용할 수 있습니다.
* 사용하려는 데이터의 성격과 군집화의 목적에 맞는 알고리즘을 선택하는 것이 중요합니다.

클러스터링 결과에 영향을 미친 \*\*특징(feature)\*\*들을 식별하는 데는 여러 방법이 있습니다. 주로 **특징의 중요도**를 평가하는 기법을 사용하거나, **상관 관계** 및 \*\*주성분 분석(PCA)\*\*을 통해 각 특성이 클러스터링에 얼마나 중요한지 알 수 있습니다. 여기서는 주요 방법을 소개하고, 각 방법을 Python 코드로 설명하겠습니다.

**1. 클러스터링 결과와의 상관 관계 분석**

클러스터링 결과 (cluster 열)와 각 feature의 상관 관계를 분석하여, 어떤 feature들이 군집을 잘 구분할 수 있는지 파악할 수 있습니다. 상관 관계가 높은 feature들은 클러스터링에 더 큰 영향을 미친다고 볼 수 있습니다.

**코드 예시:**

python

코드 복사

import pandas as pd

# CSV 파일 로드

df = pd.read\_csv('your\_file.csv')

# 클러스터링 결과와 특성들 간의 상관 관계 계산

correlation\_matrix = df.corr()

# 클러스터 열과 각 특성 간의 상관 관계만 추출

cluster\_correlation = correlation\_matrix['cluster'].sort\_values(ascending=False)

print(cluster\_correlation)

**2. 특징 중요도 (Feature Importance) 분석: Random Forest 사용**

랜덤 포레스트(Random Forest) 모델을 이용하면, 각 feature가 **클러스터링**을 얼마나 잘 구분하는지에 대한 중요도를 평가할 수 있습니다. RandomForestClassifier를 사용하여, 각 feature의 중요도를 계산할 수 있습니다. 이를 통해 어떤 feature가 클러스터링에 더 큰 영향을 미쳤는지 확인할 수 있습니다.

**코드 예시:**

python

코드 복사

from sklearn.ensemble import RandomForestClassifier

# 특성(클러스터 제외)과 레이블 설정

X = df.drop(columns=['cluster', 'label'])

y = df['cluster']

# 랜덤 포레스트 모델 학습

rf = RandomForestClassifier(n\_estimators=100)

rf.fit(X, y)

# feature 중요도 출력

feature\_importances = pd.Series(rf.feature\_importances\_, index=X.columns)

feature\_importances = feature\_importances.sort\_values(ascending=False)

# 중요도가 큰 순서대로 출력

print(feature\_importances)

**3. 주성분 분석(PCA)을 통한 특성 중요도 파악**

PCA는 데이터의 분산을 최대화하는 방식으로 특성을 차원 축소하는 기법입니다. PCA를 사용하면 **주성분**들이 데이터를 설명하는 정도를 알 수 있으며, 이를 통해 각 특성이 차원 축소 후 데이터에 얼마나 중요한지 파악할 수 있습니다. 주성분이 가장 많이 설명하는 특성들이 클러스터링에 중요한 특성일 가능성이 높습니다.

**코드 예시:**

python

코드 복사

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

# 특성들만 선택

X = df.drop(columns=['cluster', 'label'])

# 데이터 스케일링

scaler = StandardScaler()

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

# PCA 수행

pca = PCA()

pca.fit(X\_scaled)

# 각 주성분에 대한 설명된 분산 비율

explained\_variance\_ratio = pca.explained\_variance\_ratio\_

# 주성분에 대한 기여도 출력

components = pd.DataFrame(pca.components\_, columns=X.columns)

# 각 특성의 기여도 (각 주성분에 대한 기여도 합산)

feature\_contribution = components.abs().sum(axis=0)

# 중요도가 큰 순서대로 출력

feature\_contribution = feature\_contribution.sort\_values(ascending=False)

print(feature\_contribution)

**4. 클러스터링 중심 분석 (Centroid Analysis)**

KMeans와 같은 군집화 알고리즘을 사용할 경우, 각 클러스터의 중심(centroid)을 계산할 수 있습니다. 각 클러스터의 중심이 다른 특성들의 값에 어떻게 영향을 받는지 분석함으로써, 각 feature가 클러스터를 어떻게 구분하는지 평가할 수 있습니다. 클러스터별 중심값을 통해 중요한 특성을 파악할 수 있습니다.

**코드 예시:**

python

코드 복사

from sklearn.cluster import KMeans

# 클러스터링 모델 훈련 (클러스터 개수는 예시로 3으로 설정)

kmeans = KMeans(n\_clusters=3, random\_state=42)

df['cluster'] = kmeans.fit\_predict(X)

# 각 클러스터의 중심(centroid) 추출

centroids = kmeans.cluster\_centers\_

# 중심값을 DataFrame으로 변환

centroid\_df = pd.DataFrame(centroids, columns=X.columns)

# 각 클러스터의 중심값 출력

print(centroid\_df)

**5. 상관 분석을 통한 클러스터링 변수 중요도**

만약 **클러스터링 전**에 각 특성이 데이터의 군집 구조를 얼마나 잘 구분하는지 알고 싶다면, **상관 분석**을 활용할 수 있습니다. 각 특성 간의 상관관계를 분석하고, **클러스터 중심 간 차이**를 평가하는 방법이 있습니다.

**요약**

1. **상관 관계 분석**: cluster 열과 각 feature 간의 상관 관계를 계산하여 어떤 feature들이 클러스터를 구분하는 데 중요한 역할을 하는지 평가할 수 있습니다.
2. **랜덤 포레스트를 통한 중요도 분석**: 랜덤 포레스트 모델을 사용하여 각 feature의 중요도를 평가할 수 있습니다.
3. **PCA**: 주성분 분석을 통해 각 feature가 데이터 분산에 미치는 영향을 평가하고, 중요한 특성을 파악할 수 있습니다.
4. **클러스터 중심 분석**: 각 클러스터의 중심을 분석하여 클러스터를 구분하는 데 중요한 feature들을 식별할 수 있습니다.

이 방법들을 통해, 클러스터링 결과에 가장 큰 영향을 미친 **feature**들을 파악하고, 더 효율적인 데이터 분석과 모델링을 할 수 있을 것입니다.

**기여도가 큰 feature란?**

출력된 components 값에서 **절댓값이 큰 값**을 가진 feature들은 **PCA의 주성분에 중요한 영향을 미친 feature**들입니다. 이 값들이 크다는 것은 그 feature가 해당 주성분을 형성하는 데 큰 비중을 차지한다는 의미입니다.

**예시 분석**

1. \*\*주성분 1 (PCA1)\*\*에서 avg (temperature)\_group의 기여도가 0.3이라면, 이 feature는 첫 번째 주성분에 상당히 영향을 미친다는 뜻입니다.
2. \*\*주성분 2 (PCA2)\*\*에서 max (temperature)\_group의 기여도가 0.7이라면, max (temperature)\_group은 두 번째 주성분을 형성하는 데 매우 중요한 역할을 합니다.
3. \*\*주성분 3 (PCA3)\*\*에서 AQI\_group의 기여도가 0.5이라면, 이 feature는 세 번째 주성분에 큰 영향을 미칩니다.

**어떤 feature가 클러스터링에 강력한 영향을 미쳤는지는 어떻게 판단할 수 있을까?**

PCA 결과에서 특정 주성분을 구성하는 데 **기여도가 큰 feature**는, **클러스터링에서 중요한 역할을 했을 가능성이 큽니다**. 그 이유는 클러스터링은 데이터의 패턴을 잘 파악하는 것이므로, PCA에서 중요한 feature들이 클러스터를 잘 구분할 수 있는 정보들을 담고 있을 가능성이 높기 때문입니다.

**이를 활용한 방법**

1. **PCA에서 기여도가 큰 feature를 찾기**: components에서 각 주성분을 구성하는 feature들의 기여도를 보면, **PCA1, PCA2, PCA3 등에서 기여도가 높은 feature들**을 찾을 수 있습니다.
2. **클러스터링에 영향을 미친 feature 식별**: 클러스터링의 결과에 영향을 많이 미친 feature들은 보통 **PCA의 주성분을 잘 설명하는 feature들**입니다. PCA에서 높은 기여도를 가진 feature들은 클러스터 간 차이를 잘 구별하는 데 중요한 역할을 했을 가능성이 큽니다.

**예시: 특정 주성분의 기여도 분석**

예를 들어, PCA1에서 avg (temperature)\_group, max (temperature)\_group, min (temperature)\_group 등의 기여도가 높다면, 이러한 특성들이 **온도와 관련된 정보**로 클러스터링을 형성하는 데 중요한 역할을 했다고 해석할 수 있습니다. 반대로 AQI\_group이나 Particulate Matter 같은 대기질 관련 특성들이 주성분에 큰 영향을 미쳤다면, **대기질**이 클러스터를 구분하는 중요한 요소였다고 판단할 수 있습니다.

**결론**

* **PCA에서 기여도가 큰 feature**는 해당 주성분을 형성하는 데 중요한 역할을 하며, 클러스터링에서 중요한 특성일 가능성이 큽니다.
* **기여도가 높은 feature**들은 **클러스터 간 차이를 구분하는 데 중요한 요소**일 수 있으므로, 이를 바탕으로 클러스터링 결과를 해석하거나, 특정 feature들이 클러스터에 미치는 영향을 평가할 수 있습니다.

""" PCA 분석을 통해 각 주성분의 분산 설명력과 기여도를 평가하는 함수 Parameters: - X: DataFrame (특성 데이터) - n\_components: 사용할 주성분 개수 (None이면 가능한 모든 주성분 사용) - threshold: 기여도가 높은 feature를 추출할 기여도 임계값 (기본값 0.3) Returns: - explained\_variance: 각 주성분의 분산 설명 비율 - important\_features: 기여도가 높은 feature 리스트 - explained\_variance\_ratio: 각 주성분이 데이터의 변동성을 얼마나 설명하는지 비율 """

**설명:**

1. **상위 5개 피처 추출**:
   * feature\_component.csv에서 **component 값**이 가장 큰 상위 5개 feature를 추출합니다. 이를 위해 데이터를 내림차순으로 정렬하고 상위 5개 항목을 선택합니다.
2. **클러스터링 데이터에서 분산 계산**:
   * clustering\_data.csv에서 상위 5개 feature와 cluster 정보를 포함한 데이터를 선택하여, 각 클러스터별로 분산을 계산합니다. 이를 통해 각 feature가 클러스터 내에서 얼마나 분산되는지 알 수 있습니다.
3. **Precision 및 Recall 계산**:
   * 각 feature에 대해 **precision**과 **recall**을 계산합니다.
   * 예측 값은 해당 feature의 값이 중간값(임계값)을 초과하면 1 (positive cluster), 그렇지 않으면 0 (negative cluster)로 예측합니다. (이 방법은 예시이며, 더 정교한 예측 방법을 사용할 수 있습니다.)
   * precision\_score와 recall\_score는 실제 클러스터 값과 예측한 값을 비교하여 계산합니다.

**결과:**

* 각 feature에 대한 **precision**과 **recall** 값을 출력하여, 해당 feature가 **클러스터링에 얼마나 기여**하는지를 평가할 수 있습니다.
* 높은 precision과 recall 값은 해당 feature가 클러스터의 구분에 더 중요한 역할을 한다는 것을 의미합니다.

이렇게 하면 각 피처가 클러스터링 결과에 얼마나 기여하는지를 **precision**과 **recall** 값을 통해 평가할 수 있습니다.