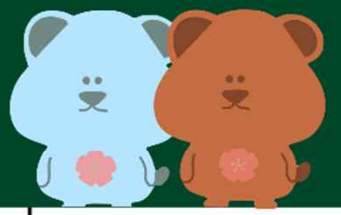
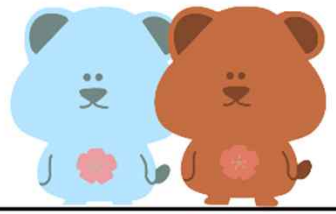


Kaggle 필사 (군집화)

1팀 최은빈 강민정 이채원

EWHA EURON

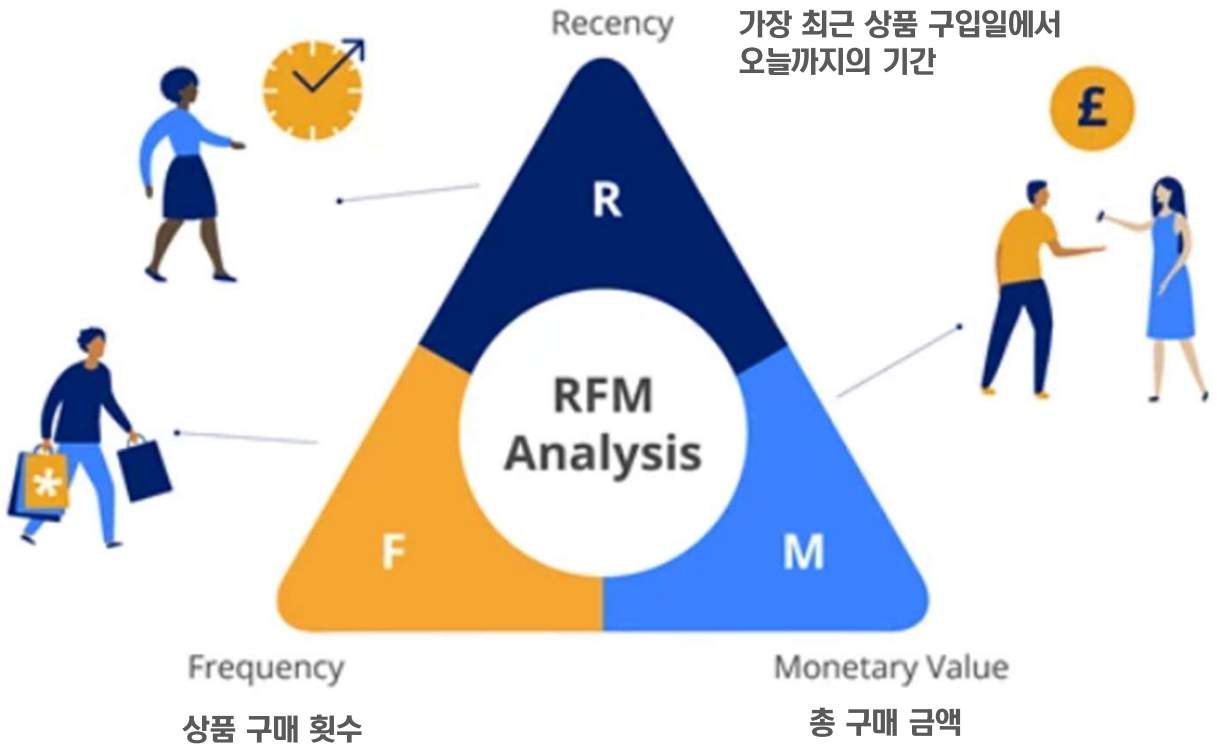
목차



EWHA EURON

1. 고객 세그먼테이션

#01 고객 세그먼테이션 기법



출처: https://velog.io/@vive0508/rfm

#02 데이터 EDA

```
retail_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541909 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   InvoiceNo    541909 non-null  object
 1   StockCode    541909 non-null  object
 2   Description  540455 non-null  object
 3   Quantity     541909 non-null  int64
 4   InvoiceDate  541909 non-null  datetime64[ns]
 5   UnitPrice    541909 non-null  float64
 6   CustomerID   406829 non-null  float64
 7   Country      541909 non-null  object
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 33.1+ MB
```

-customerID의 null 값이 많음

-description의 null 값도 존재

EWHA EURON

#02 데이터 EDA

```
#quantity >0 , unitprice >0, customerid가 not null 인 것만 필터링
retail_df = retail_df[retail_df['Quantity']>0]
retail_df = retail_df[retail_df['UnitPrice']>0]
retail_df = retail_df[retail_df['CustomerID'].notnull()]

#country는 영국인 것만
retail_df = retail_df[retail_df['Country']== 'United Kingdom']

print(retail_df.shape)
retail_df.isnull().sum()
```

```
(397884, 8)

InvoiceNo      0
StockCode      0
Description    0
Quantity       0
InvoiceDate    0
UnitPrice      0
CustomerID     0
Country        0
dtype: int64
```

- data 행이 397884로 줄어듦

-더 이상 null 값이 존재하지 않음

EWHA EURON

#03 RFM 기반 데이터 가공

```
#총 구매금액 칼럼 만들기
retail_df['sale_amount'] = retail_df['Quantity'] * retail_df['UnitPrice']
retail_df['CustomerID'] = retail_df['CustomerID'].astype(int)
```

```
aggregations = {
    'InvoiceDate' :'max',
    'InvoiceNo' : 'count',
    'sale_amount':'sum'
}

cust_df = retail_df.groupby('CustomerID').agg(aggregations)

#groupby된 결과 칼럼 값을 Recency, Frequency, Monetary로 변경
cust_df = cust_df.rename(columns = {'InvoiceDate' :'Recency',
                                    'InvoiceNo' : 'Frequency',
                                    'sale_amount': 'Monetary'
                                   }
                         )

cust_df = cust_df.reset_index()
cust_df.head()
```

	CustomerID	Recency	Frequency	Monetary
0	12346	2011-01-18 10:01:00	1	77183.60
1	12347	2011-12-07 15:52:00	182	4310.00
2	12348	2011-09-25 13:13:00	31	1797.24
3	12349	2011-11-21 09:51:00	73	1757.55
4	12350	2011-02-02 16:01:00	17	334.40

EWHA EURON

#03 RFM 기반 데이터 가공

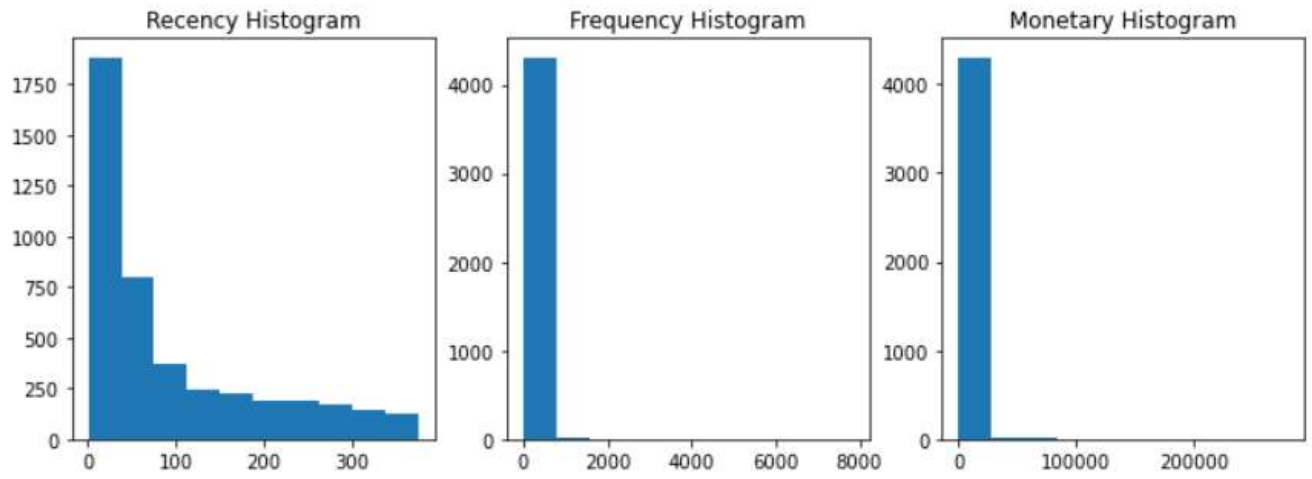
```
import datetime as dt
cust_df['Recency'] = dt.datetime(2011,12,10) - cust_df['Recency']
cust_df['Recency'] = cust_df['Recency'].apply(lambda x: x.days+1)
print('cust_df row와 column 건수는', cust_df.shape)
cust_df.head()
```

```
cust_df row와 column 건수는 (4338, 4)
```

	CustomerID	Recency	Frequency	Monetary
0	12346	326	1	77183.60
1	12347	3	182	4310.00
2	12348	76	31	1797.24
3	12349	19	73	1757.55
4	12350	311	17	334.40

#03 RFM 기반 데이터 가공

```
#hist()를 이용하여 각 칼럼의 값 분포도 확인
import matplotlib.pyplot as plt

fig, (ax1, ax2, ax3) = plt.subplots(figsize=(12,4), nrows=1, ncols=3)
ax1.set_title('Recency Histogram')
ax1.hist(cust_df['Recency'])

ax2.set_title('Frequency Histogram')
ax2.hist(cust_df['Frequency'])

ax3.set_title('Monetary Histogram')
ax3.hist(cust_df['Monetary'])

plt.show()
```



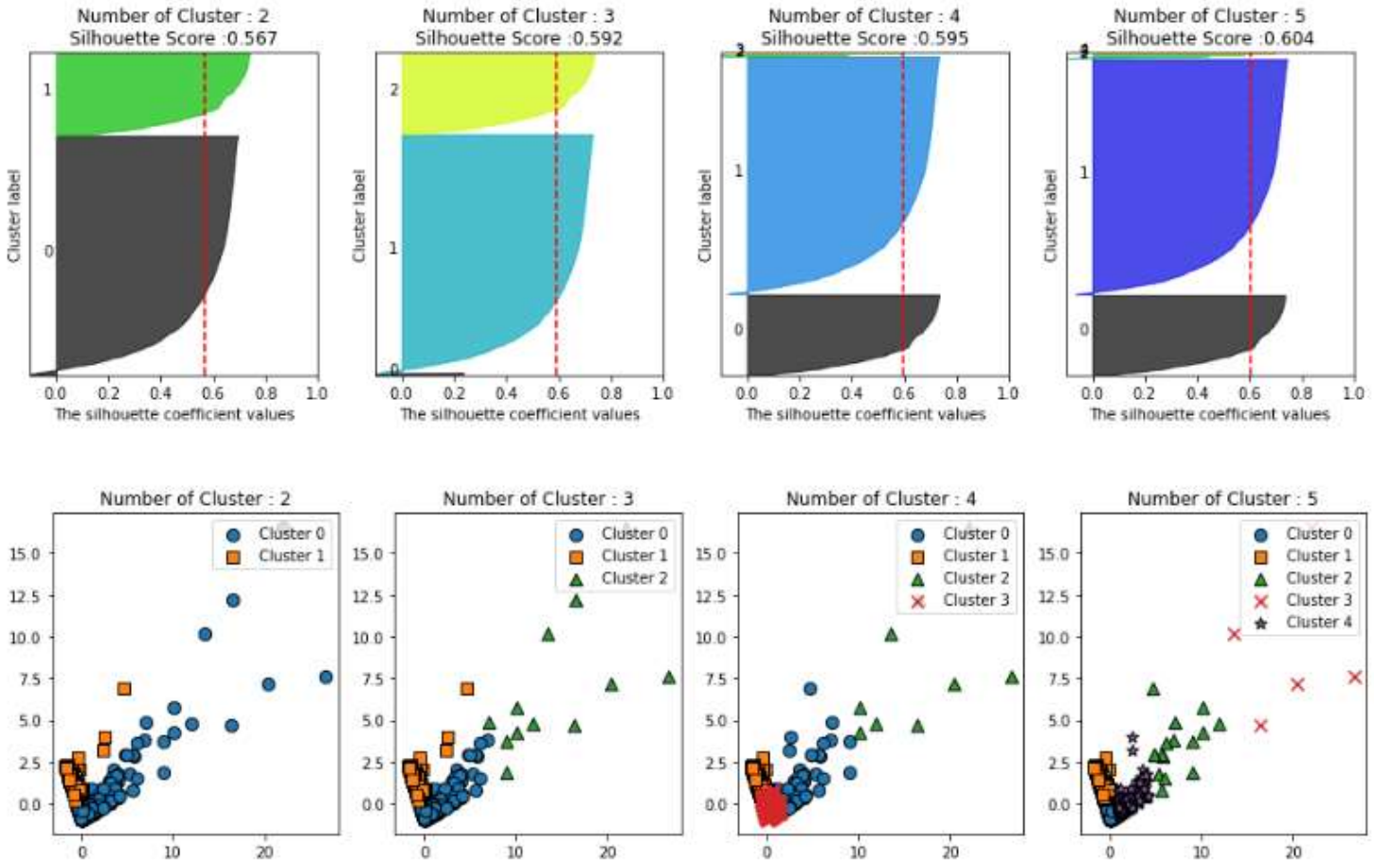
#03 RFM 기반 데이터 가공

```
#백분위 값으로 데이터 분포도 확인
cust_df[['Recency', 'Frequency', 'Monetary']].describe()
```

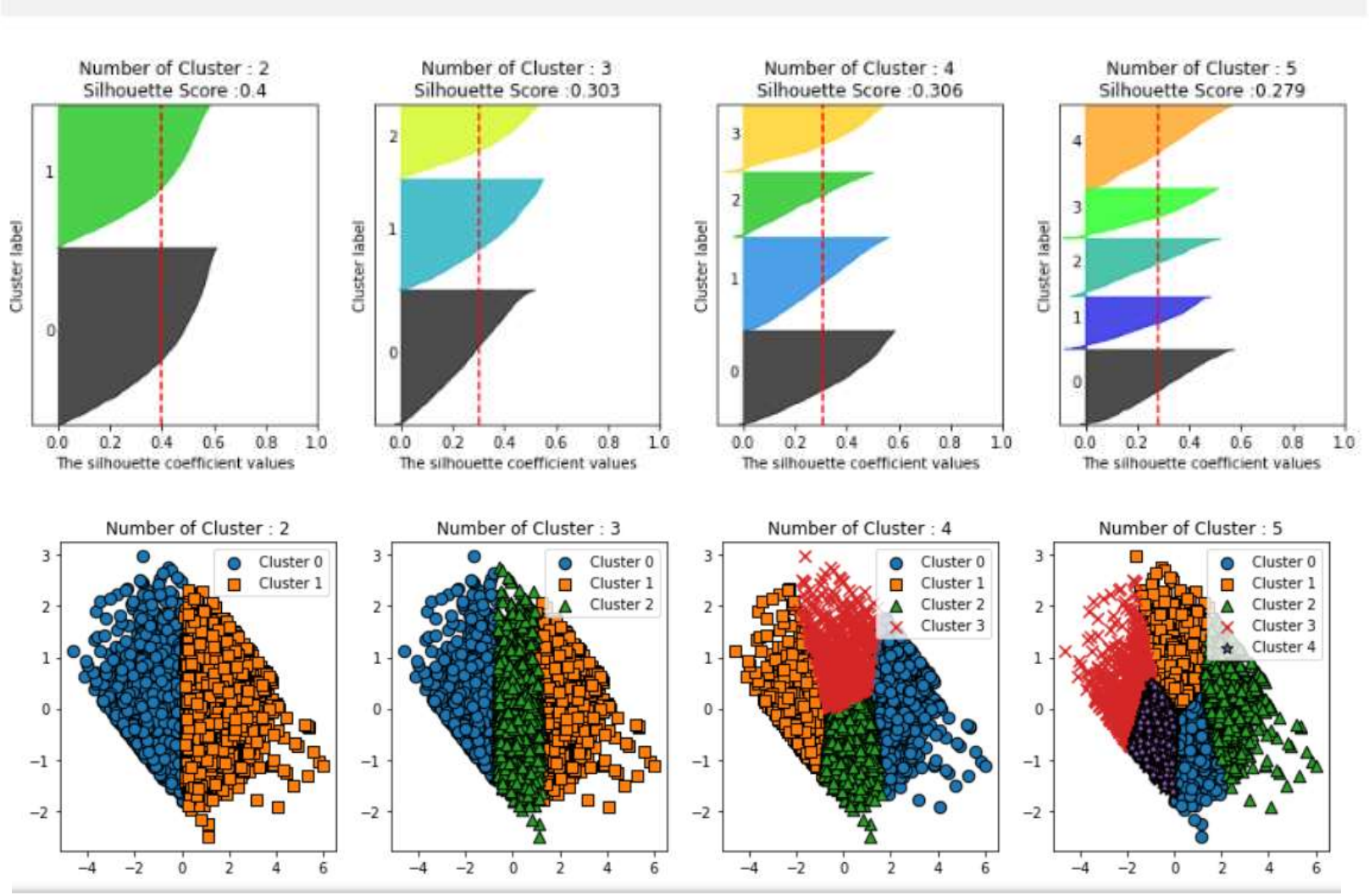
	Recency	Frequency	Monetary
count	4338.000000	4338.000000	4338.000000
mean	93.059474	91.720609	2054.266460
std	100.012264	228.785094	8989.230441
min	1.000000	1.000000	3.750000
25%	18.000000	17.000000	307.415000
50%	51.000000	41.000000	674.485000
75%	142.750000	100.000000	1661.740000
max	374.000000	7847.000000	280206.020000

EWHA EURON

#03 RFM 기반 데이터 가공



#03 RFM 기반 데이터 가공



2. Popular Unsupervised Clustering Algorithms

EWHA EURON

고객 세그먼테이션 실습

여러가지 비지도학습 군집화 방법으로 고객 세그먼테이션을 실습하는 캐글 노트북

- 교재에서 다룬 K-Means, DBSCAN, 평균이동
- 계층적 군집화인 Agglomerative Hierarchical Clustering (병합 군집)

방식을 사용

데이터셋 준비

① 데이터 불러오기

```
df = pd.read_csv('/content/Mall_Customers.csv')
df.head()
```

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype 
---  ------                  --------------  ----- 
 0   CustomerID              200 non-null    int64 
 1   Gender                  200 non-null    object
 2   Age                     200 non-null    int64 
 3   Annual Income (k$)      200 non-null    int64 
 4   Spending Score (1-100)  200 non-null    int64 
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```

② 컬럼명 변경

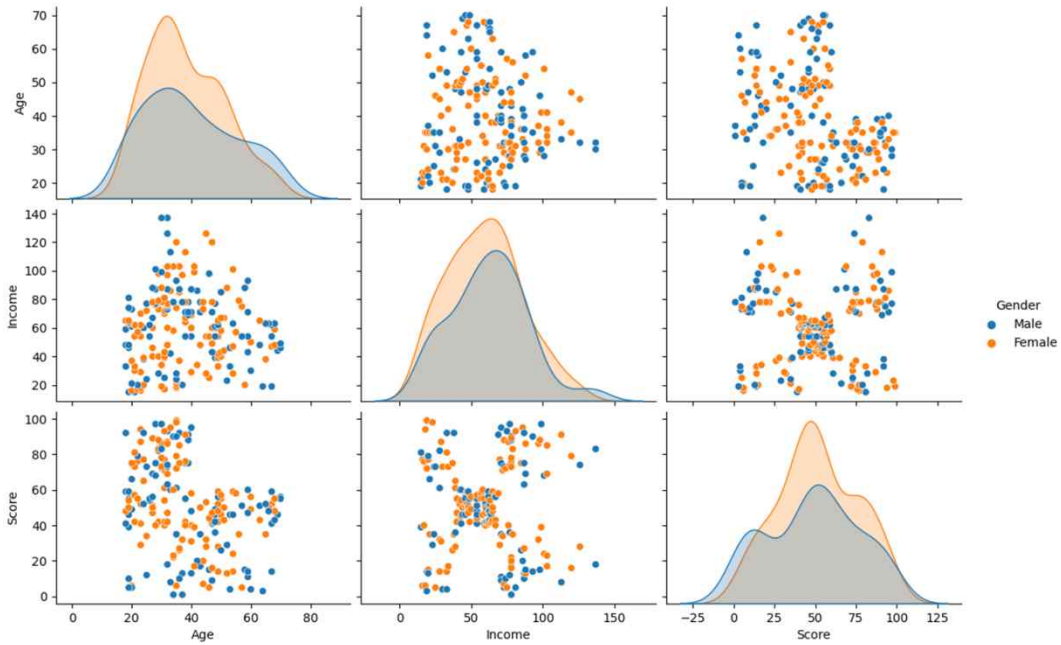
```
df.rename(index=str, columns={'Annual Income (k$)': 'Income',
                              'Spending Score (1-100)': 'Score'}, inplace=True)
df.head()
```

	CustomerID	Gender	Age	Income	Score
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

#01. 데이터 살펴보기

컬럼별 시각화

```
1 X = df.drop(['CustomerID', 'Gender'], axis=1)
2 sns.pairplot(df.drop('CustomerID', axis=1), hue='Gender', aspect=1.5)
3 plt.show()
```



- pairplot 이용하여 시각화
- 성별이 고객 세그먼테이션에 직접적인 영향을 미치지 않는 것을 볼 수 있음

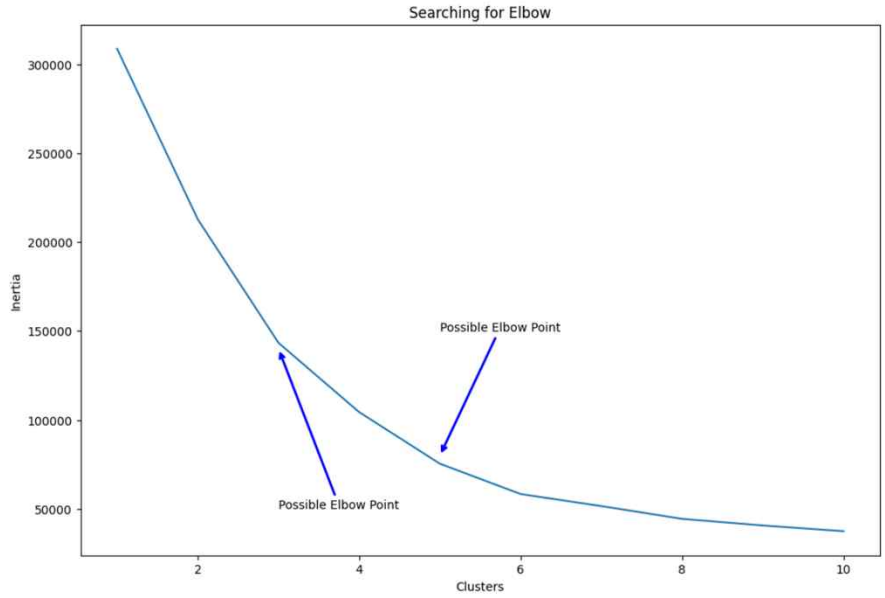
  => 고객ID와 성별 컬럼을 제거한 X를 이용

#02. 클러스터링 - (1) K-Means

 # K-Means: 거리 기반 작동 알고리즘

- 임의의 군집 중심점(centroid)을 선택해 해당 중심에 가장 가까운 포인트들을 선택하는 군집화 기법

```python
from sklearn.cluster import KMeans

clusters = []

for i in range(1, 11): # 군집 개수 1~10까지 올려가며 확인
    km = KMeans(n_clusters=i).fit(X) # 객체 생성하고 i개 군집으로 학습
    clusters.append(km.inertia_) # inertia 값을 clusters 리스트에 삽입

fig, ax = plt.subplots(figsize=(12, 8))
sns.lineplot(x=list(range(1, 11)), y=clusters, ax=ax)
ax.set_title('Searching for Elbow')
ax.set_xlabel('Clusters')
ax.set_ylabel('Inertia')

ax.annotate('Possible Elbow Point', xy=(3, 140000), xytext=(3, 50000), xycoords='data',
            arrowprops=dict(arrowstyle='->', connectionstyle='arc3', color='blue', lw=2))

ax.annotate('Possible Elbow Point', xy=(5, 80000), xytext=(5, 150000), xycoords='data',
            arrowprops=dict(arrowstyle='->', connectionstyle='arc3', color='blue', lw=2))

plt.show()
```



- **Inertia**: 클러스터 간의 거리의 합. *inertia_* 속성으로 확인
- **Elbow Method**

  : inertia가 급격히 떨어지는 구간이 생길 때,

  이 지점의 K값을 군집의 개수로 사용하는 방법

- 3과 5에서 변화가 커짐

  => 군집 개수를 3개, 5개로 각각 수행한 뒤, 결과 보고 결정

EWHA EURON

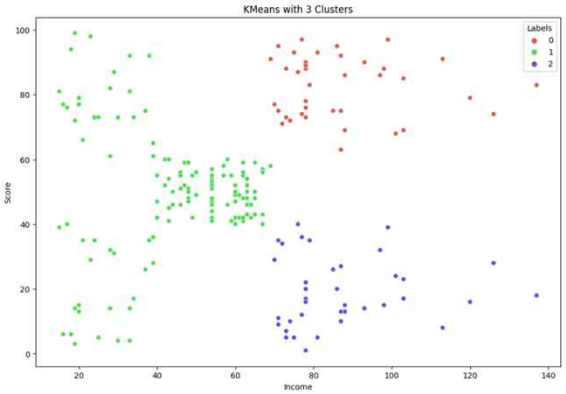
#02. 클러스터링 - (1) K-Means

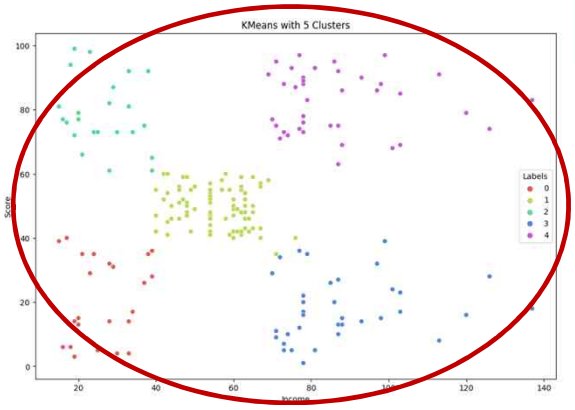
군집 개수 = 3

```
km3 = KMeans(n_clusters=3).fit(X)

X['Labels'] = km3.labels_
plt.figure(figsize=(12, 8))
sns.scatterplot(x=X['Income'], y=X['Score'], hue=X['Labels'], palette=sns.color_palette('hls', 3))
plt.title('KMeans with 3 Clusters')
plt.show()
```

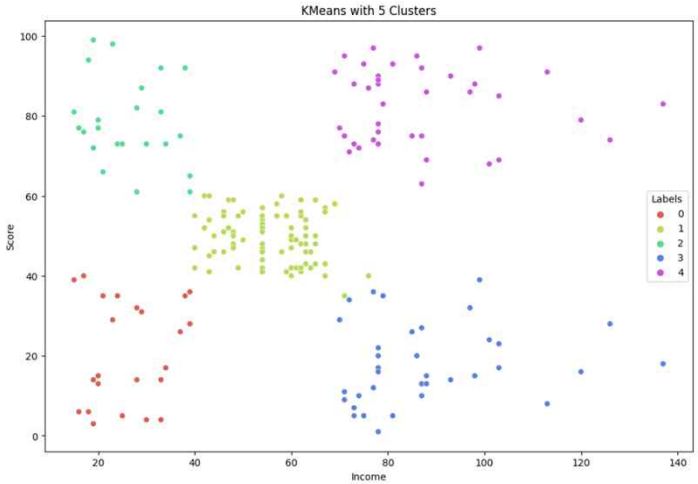
군집 개수 = 5

```
km5 = KMeans(n_clusters=5).fit(X)

X['Labels'] = km5.labels_
plt.figure(figsize=(12, 8))
sns.scatterplot(x=X['Income'], y=X['Score'], hue=X['Labels'], palette=sns.color_palette('hls', 5))
plt.title('KMeans with 5 Clusters')
plt.show()
```





=> 시각화 비교 결과 군집 개수 5개 선택



 # 군집 분석

- label0: 낮은 수입, 낮은 소비
- label1: 중간 수입, 중간 소비
- label2: 낮은 수입, 높은 소비
- label3: 높은 수입, 낮은 소비
- label4: 높은 수입, 높은 소비

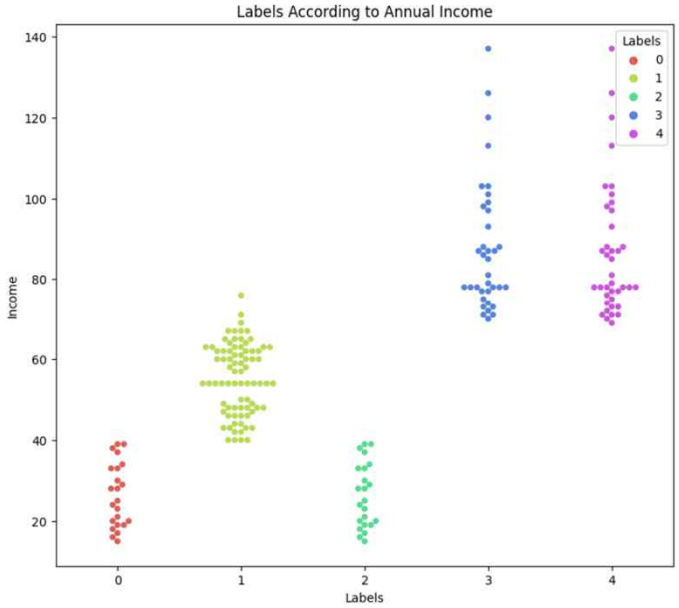
#02. 클러스터링 - (1) K-Means

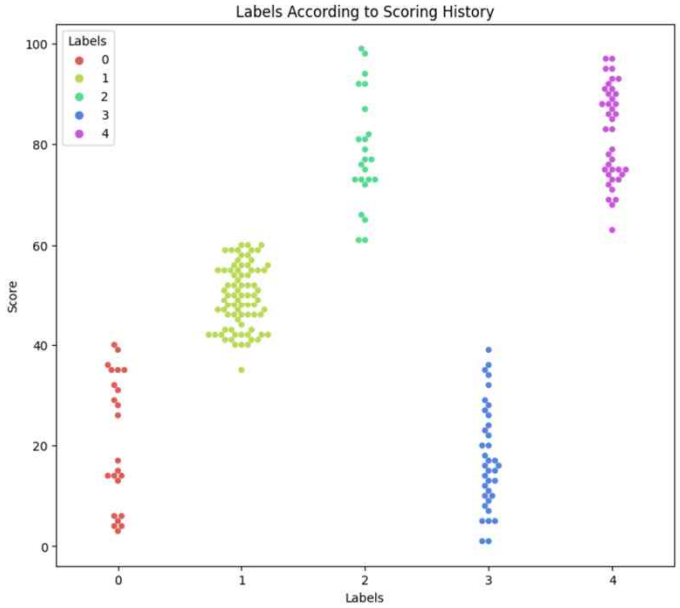
swarm plot

```
fig = plt.figure(figsize=(20,8))
ax = fig.add_subplot(121)
sns.swarmplot(x='Labels', y='Income', data=X, ax=ax, hue=X['Labels'], palette=sns.color_palette('hls', 5))
ax.set_title('Labels According to Annual Income')

ax = fig.add_subplot(122)
sns.swarmplot(x='Labels', y='Score', data=X, ax=ax, hue=X['Labels'], palette=sns.color_palette('hls', 5))
ax.set_title('Labels According to Scoring History')

plt.show()
```

- 군집별로 수입/소비 분포 산점도를 그려서 분포 확인

  => 구분 명확 -> 군집화 잘 된 것 확인

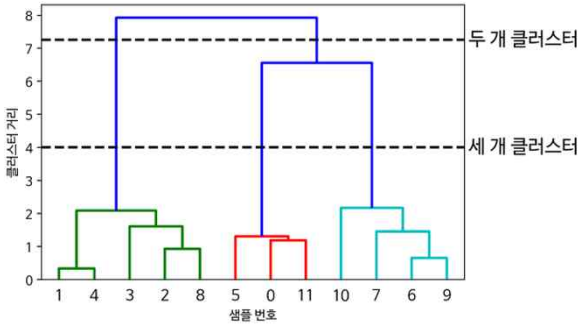




#02. 클러스터링 - (2) 병합 군집

- **계층적 군집(Hierarchical Clustering)**: 계층적 트리 모형을 이용하여 순차적, 계층적으로 유사한 클러스터끼리 통합하여 최종적으로 하나의 군집이 될 때까지 군집화
  - K-Means와 달리 군집 수를 사전에 정하지 않아도 학습 수행 가능
- **병합 군집(Agglomerative Clustering)**: 각각의 데이터 포인트를 하나의 클러스터로 지정하고, 지정된 클러스터 개수가 남을 때까지 비슷한 두 클러스터를 병합하는 알고리즘 => 상향식
  - **병합 방식**
    - Ward: 모든 클러스터 내의 분산을 가장 작게 증가시키는 두 클러스터 선택
    - Average: 클러스터 포인트 사이의 평균 거리가 가장 짧은 두 클러스터 선택
    - Complete: 클러스터 포인트 사이의 최대 거리가 가장 짧은 두 클러스터 선택
- **덴드로그램(Dendrogram)**: 계층적 군집화를 시각화할 때 사용하는 그래프
  - 덴드로그램을 가로선으로 분할하여 클러스터를 임의로 나눌 수 있음

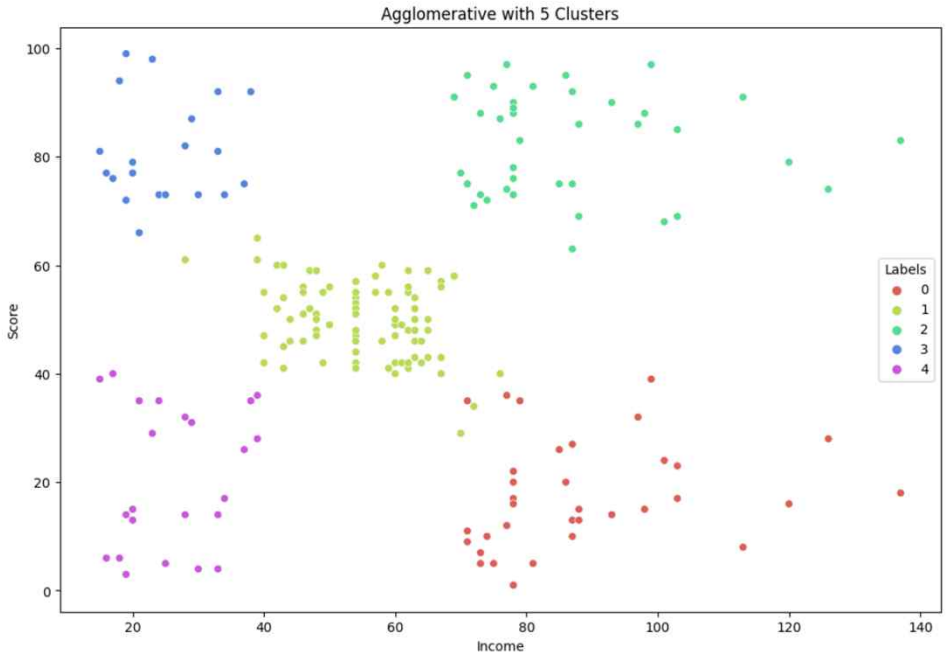
    ⇒ 사전에 클러스터 개수를 정하지 않더라도, 시각화된 덴드로그램을 보고 클러스터를 나눌 수 있음
  - y축 좌표: 군집간의 거리를 의미



#02. 클러스터링 - (2) 병합 군집

클러스터링 수행

```
from sklearn.cluster import AgglomerativeClustering

agglom = AgglomerativeClustering(n_clusters=5, linkage='complete').fit(X)

X['Labels'] = agglom.labels_
plt.figure(figsize=(12, 8))
sns.scatterplot(x=X['Income'], y=X['Score'], hue=X['Labels'], palette=sns.color_palette('hls', 5))
plt.title('Agglomerative with 5 Clusters')
plt.show()
```



- n_clusters=5 ; 군집 개수 설정
- linkage= 'complete' ; 병합 방식 설정

  (최대 거리가 가장 짧은 두 개의 클러스터 선택)

#02. 클러스터링 - (2) 병합 군집

덴드로그램

① 모든 데이터 포인트 간의 거리를 계산한 거리행렬 계산

```
from scipy.cluster import hierarchy
from scipy.spatial import distance_matrix

dist = distance_matrix(X, X)
print(dist)
```

```
[[  0.          42.05948169  33.03028913 ... 117.12813496 124.53915047
  130.17296186]
 [ 42.05948169   0.          75.01999733 ... 111.76761606 137.77880824
  122.35195135]
 [ 33.03028913  75.01999733   0.         ... 129.89226305 122.24974438
  143.78456106]
 ...
 [117.12813496 111.76761606 129.89226305 ...   0.          57.10516614
   14.35270009]
 [124.53915047 137.77880824 122.24974438 ...  57.10516614   0.
   65.06150936]
 [130.17296186 122.35195135 143.78456106 ...  14.35270009  65.06150936
    0.        ]]
```

② linkage 함수 파라미터로 거리행렬과 병합 방식(‘complete’) 전달

```
Z = hierarchy.linkage(dist, 'complete')
```

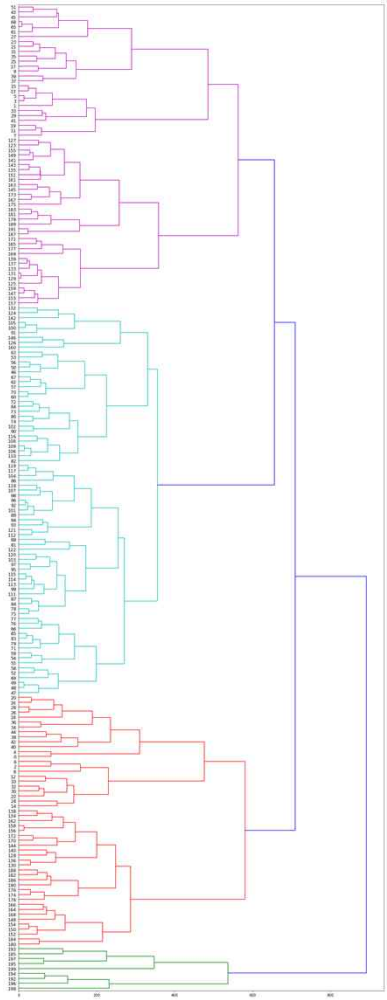
③ 덴드로그램 그리기

```
plt.figure(figsize=(18, 50))
dendro = hierarchy.dendrogram(Z, leaf_rotation=0, leaf_font_size=12, orientation='right')
```

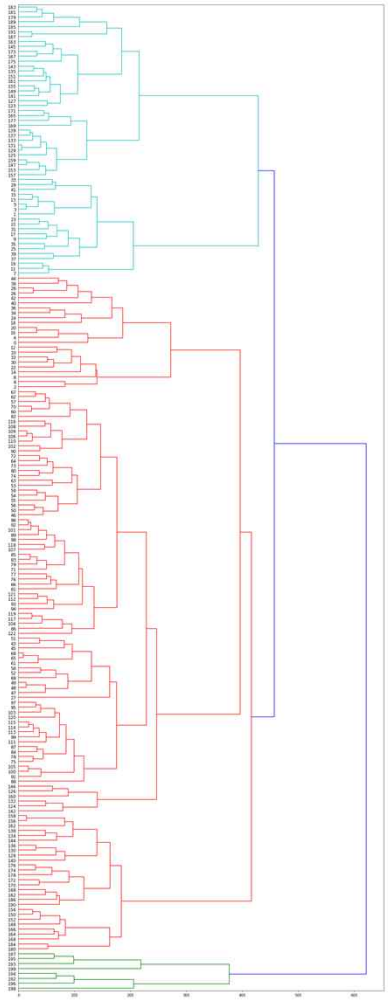
(+) 동일한 방식으로 병합 방식 ‘average’ 으로 설정

```
Z = hierarchy.linkage(dist, 'average')
plt.figure(figsize=(18, 50))
dendro = hierarchy.dendrogram(Z, leaf_rotation=0, leaf_font_size =12, orientation = 'right')
```

* Complete Linkage



* Average Linkage



EWHA EURON

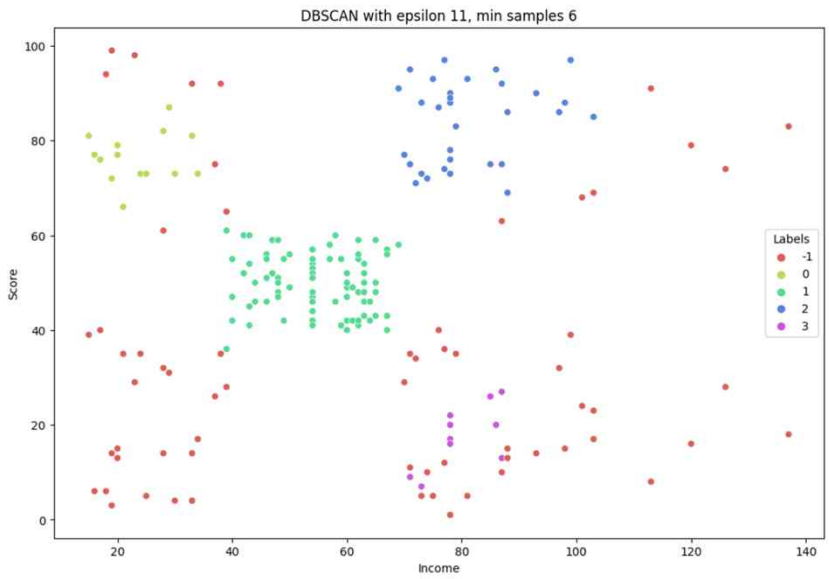
#02. 클러스터링 - (3) DBSCAN

DBSCAN: 어느 점을 기준으로 반경 x 내에 점이 n개 이상 있으면 하나의 군집으로 인식하는 방식.

- 밀도 기반 작동 알고리즘

클러스터링 수행

```
from sklearn.cluster import DBSCAN

db = DBSCAN(eps=11, min_samples=6).fit(X)

X['Labels'] = db.labels_
plt.figure(figsize=(12, 8))
sns.scatterplot(x=X['Income'], y=X['Score'], hue=X['Labels'], palette=sns.color_palette('hls', np.unique(db.labels_).shape[0]))
plt.title('DBSCAN with epsilon 11, min samples 6')
plt.show()
```



결과

- 레이블 -1은 이상치를 의미하는데, 이상치가 가장 많음
- DBSCAN: 밀도 기반 작동 방식

  => 해당 데이터는 밀도가 높지 않기 때문에 성능이 좋지 않게 나옴
- 더 큰 데이터가 있었다면 성능이 더 좋았을 것

EWHA EURON

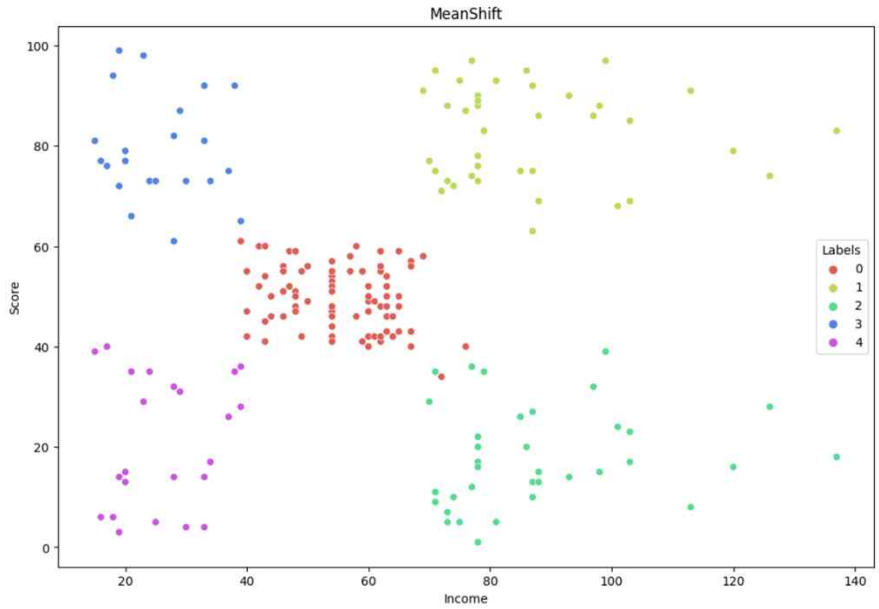
#02. 클러스터링 - (4) 평균 이동

 # 평균이동 K-Means와 유사한 작동 방식

- K-Means: 소속된 데이터의 평균 거리 중심으로 이동
- 평균 이동: 데이터가 모여 있는 밀도가 가장 높은 곳으로 이동

 # 클러스터링 수행

```
from sklearn.cluster import MeanShift, estimate_bandwidth

# The following bandwidth can be automatically detected using
best_bandwidth = estimate_bandwidth(X, quantile=0.1)
ms = MeanShift(bandwidth=best_bandwidth).fit(X)

X['Labels'] = ms.labels_
plt.figure(figsize=(12, 8))
sns.scatterplot(x=X['Income'], y=X['Score'], hue=X['Labels'],
                palette=sns.color_palette('hls', np.unique(ms.labels_).shape[0]))
plt.plot()
plt.title('MeanShift')
plt.show()
```



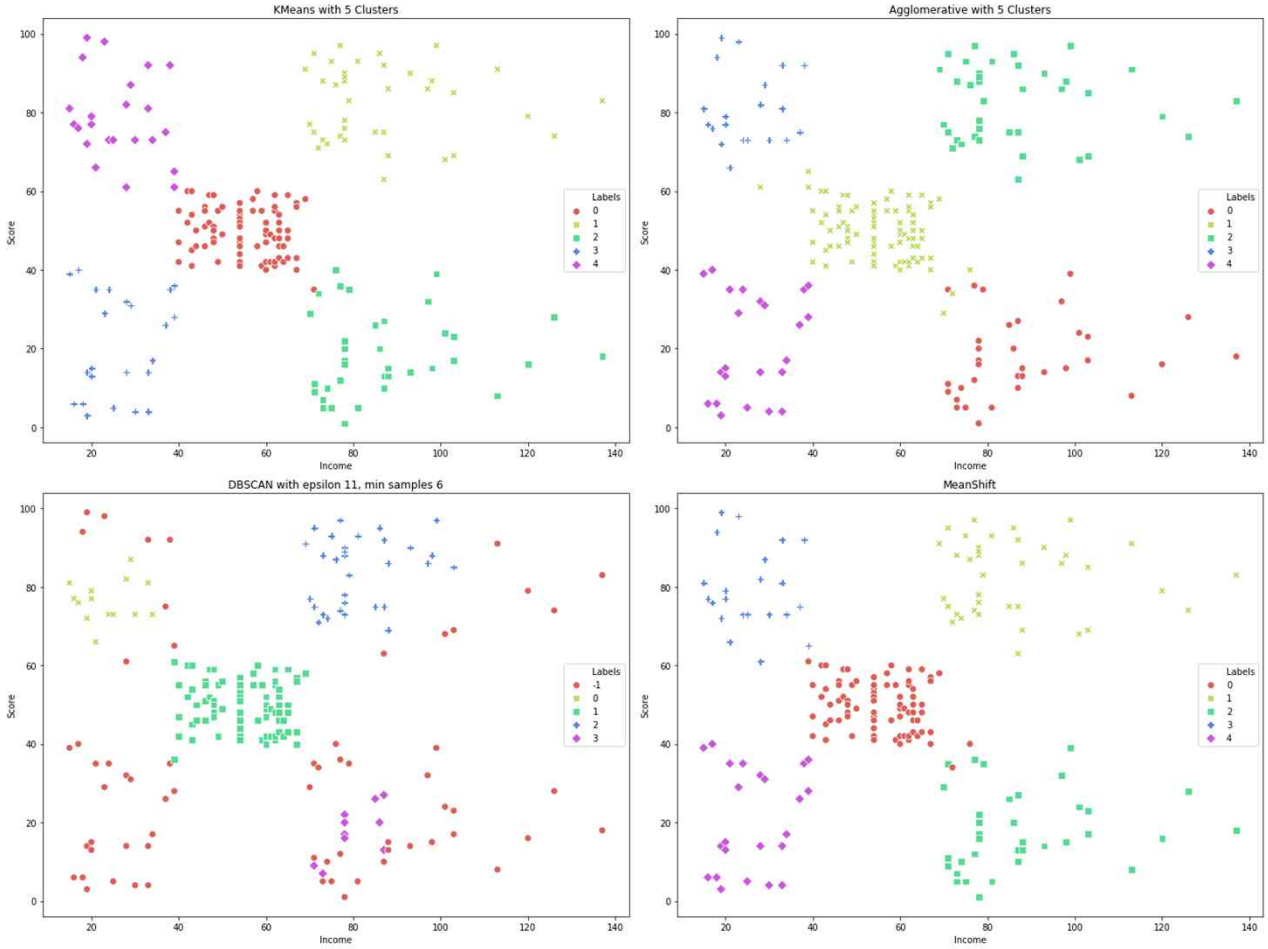
- 평균 이동의 가장 중요한 파라미터: bandwidth (대역폭)
- estimate_bandwidth 이용하여 최적의 bandwidth 값 찾기
  - quantile: 해당 비율만큼의 데이터 건수로 KNN 수행

* 일반적으로 quantile이 크면 bandwidth 값이 커져서 클러스터 개수 작아짐

EWHA EURON

#02. 클러스터링

전체 시각화



3. 군집화 추가 노트북 필사

https://www.kaggle.com/code/aditiani/kmeans-clustering-and-dbscan-for-country-data/notebook

1팀

EWHA EURON

3. 군집화 추가 노트북 필사

#01 데이터 전처리

#02 KMeans

#03 DBSCAN

EWHA EURON

#01 데이터 전처리

피처 확인

사회 경제, 건강 요인을 사용하여 국가 분류하기

Feature: child_mort, health, income, inflation, life_expec, total_fer, gdpp

Target: country

	country	child_mort	exports	health	imports	income	inflation	life_expec	total_fer	gdpp
0	Afghanistan	90.2	10.0	7.58	44.9	1610	9.44	56.2	5.82	553
1	Albania	16.6	28.0	6.55	48.6	9930	4.49	76.3	1.65	4090
2	Algeria	27.3	38.4	4.17	31.4	12900	16.10	76.5	2.89	4460

#01 데이터 전처리

피처 확인

결측치 및 각 피처 타입 확인 info()

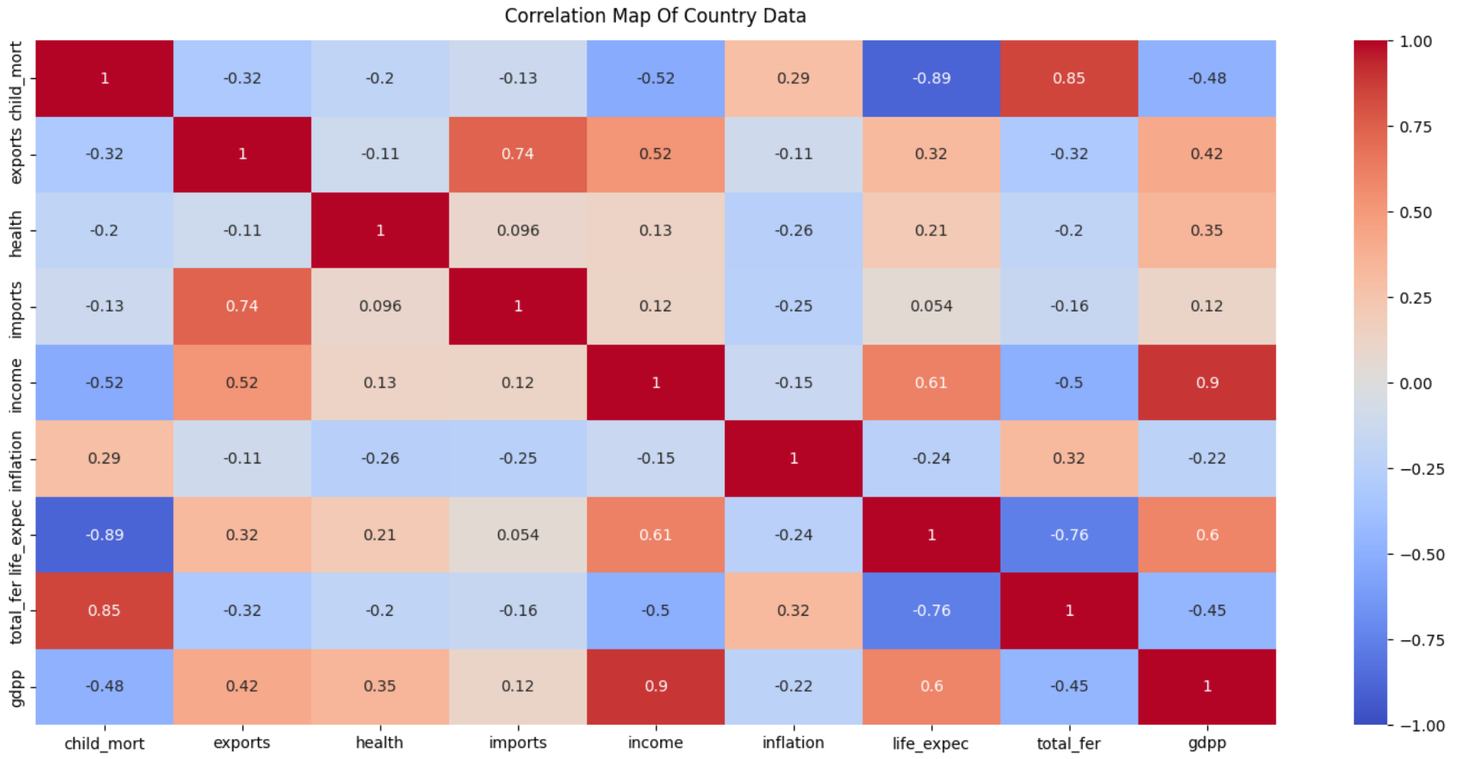
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 167 entries, 0 to 166
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   country     167 non-null    object 
 1   child_mort  167 non-null    float64
 2   exports     167 non-null    float64
 3   health      167 non-null    float64
 4   imports     167 non-null    float64
 5   income      167 non-null    int64  
 6   inflation   167 non-null    float64
 7   life_expec  167 non-null    float64
 8   total_fer   167 non-null    float64
 9   gdpp        167 non-null    int64  
dtypes: float64(7), int64(2), object(1)
memory usage: 13.2+ KB
```

#01 데이터 전처리

Heat map

피처간 상관 관계 확인

```
plt.figure(figsize=(18, 8))
sns.heatmap(country_data.corr(), vmin = -1, vmax = 1, annot = True, cmap = 'coolwarm')
plt.title('Correlation Map Of Country Data', fontdict={'fontsize':12}, pad=12);
```

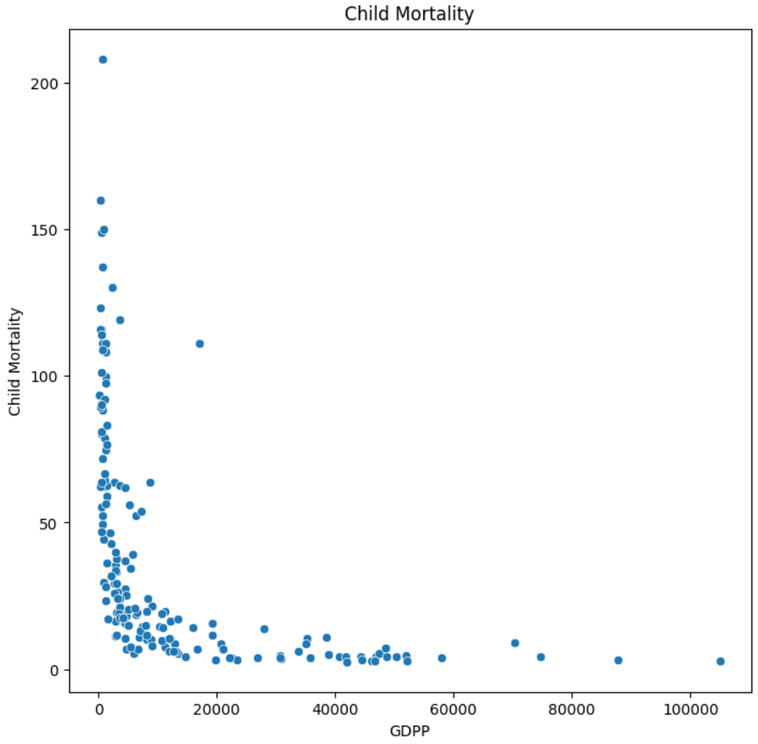


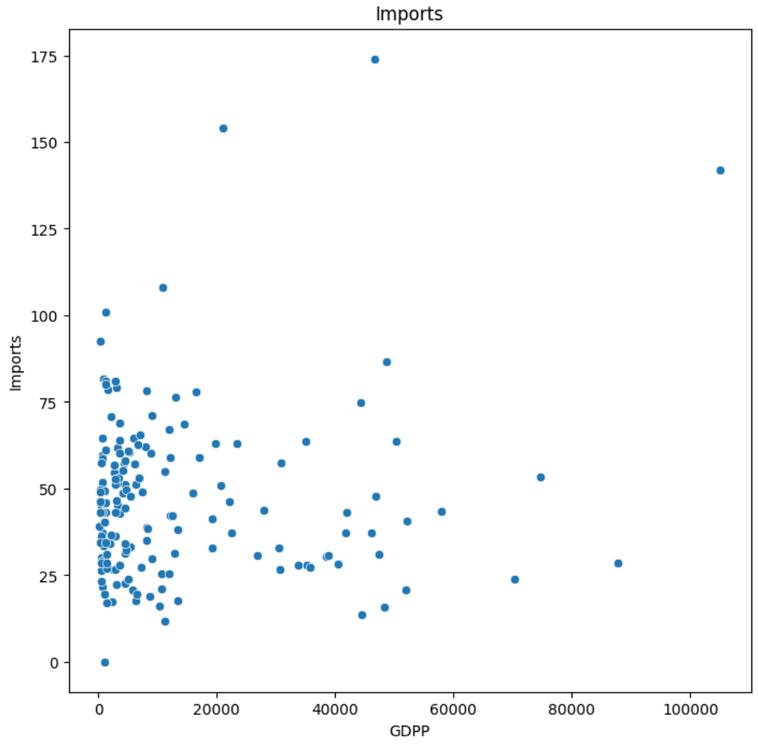
#01 데이터 전처리

Scatter plot

선택한 3개의 피처 관계를 산점도로 나타냄

gdpp & child_mort, gdpp & imports





#02 Kmeans clustering

Cluster 개수 정하기

K = 1 ~ 10, 클러스터의 개수에 따른 WSS (Within Sum of Squares) 값 변화 확인

```
[ ]  number_of_cluster = range(1,11)
     clusterings = [KMeans(n_clusters = k).fit(count_data) for k in number_of_cluster]
     centroids = [k.cluster_centers_ for k in clusterings]

     D_k = [cdist(count_data, cent, 'euclidean') for cent in centroids]
     cIdx = [np.argmin(D, axis = 1) for D in D_k]
     dist = [np.min(D, axis = 1) for D in D_k]
     avg_withinSS = [sum(d)/count_data.shape[0] for d in dist]
```

WSS = 데이터 포인트 – Centroid 간 거리 제곱의 합,

작을 수록 중심에 가까이 모여있음

#02 Kmeans clustering

Cluster 개수 정하기

클러스터 개수 k 값 증가에 따른 WSS 감소 추이를 그래프로 확인

elbow point = WSS 감소가 더디게 일어나는 지점 기준

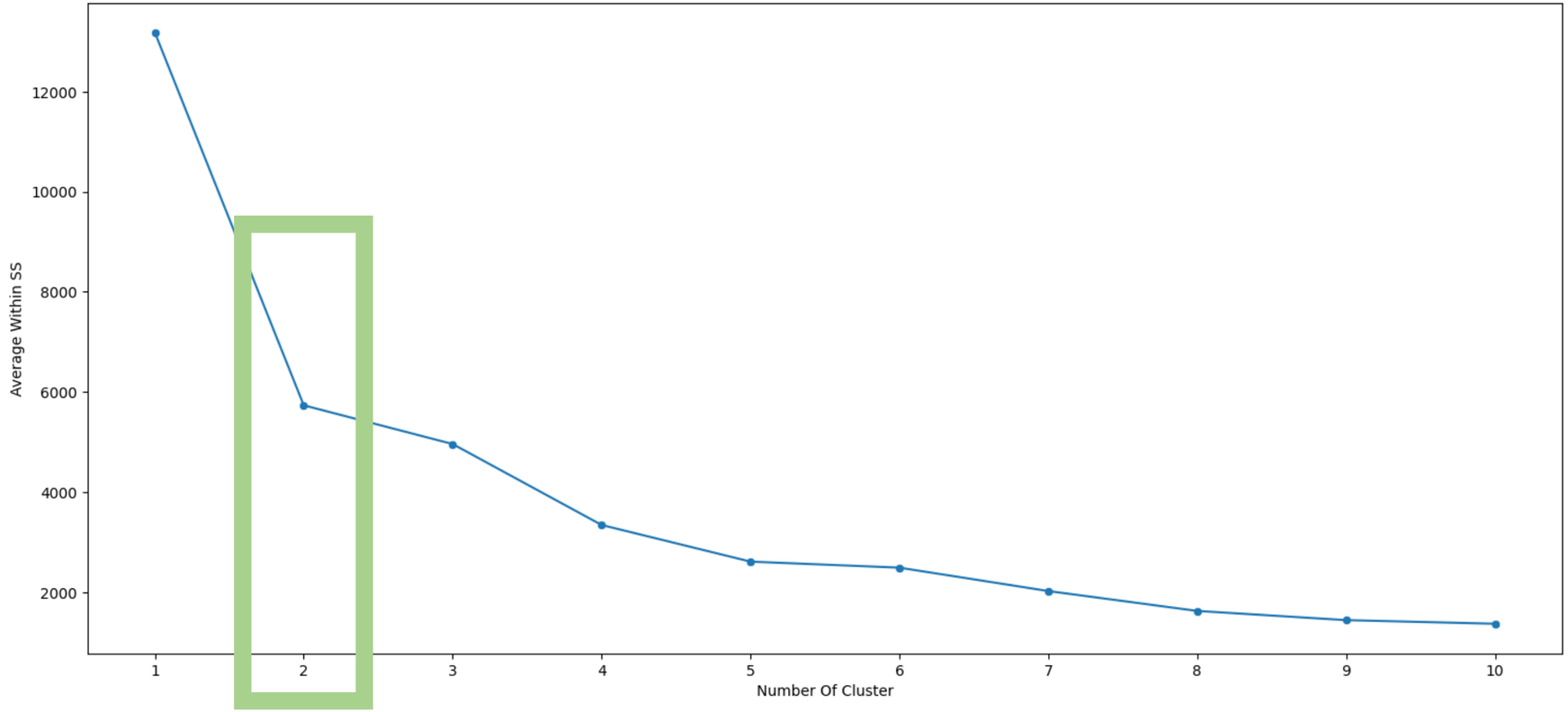
⭐ elbow point일 때 클러스터 개수가 적절하다고 판단

```
plt.figure(figsize=(18,8))
sns.lineplot(x=number_of_cluster, y=avg_withinSS)
sns.scatterplot(x=number_of_cluster, y=avg_withinSS)
plt.xticks(number_of_cluster)
plt.xlabel('Number Of Cluster')
plt.ylabel('Average Within SS')
plt.show()
```

#02 Kmeans clustering

Cluster 개수 정하기

Elbow point = 2, 클러스터의 개수는 2개가 적절



#02 Kmeans clustering

군집화

Cluster 2개로 군집화 진행 후 산점도 확인

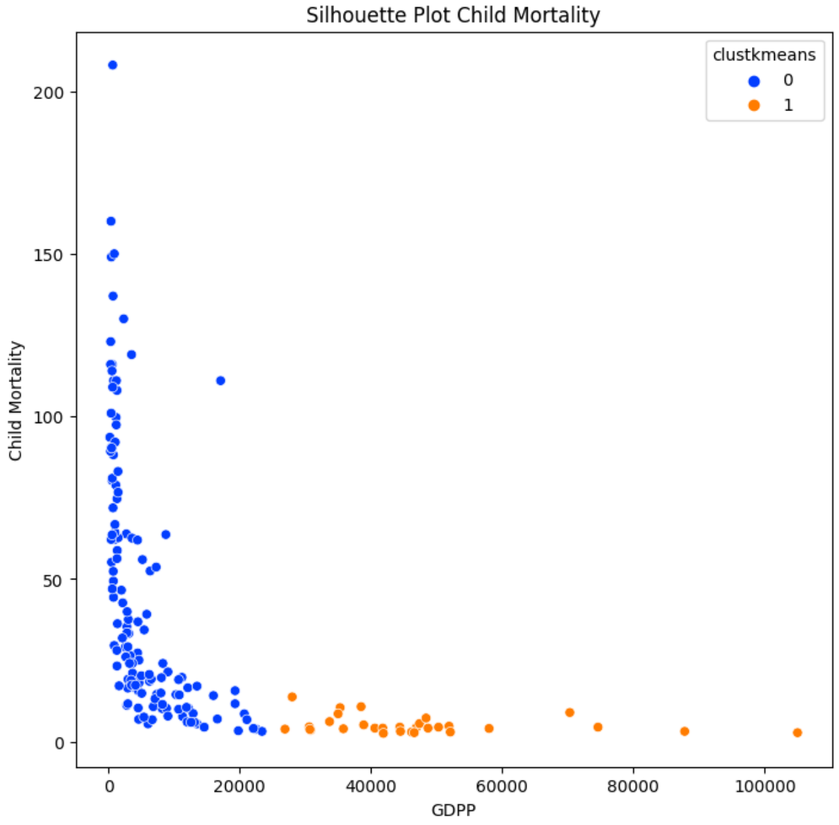
```
[ ]  kmeans = KMeans(n_clusters = 2)
     kmeans.fit(count_data)
     country_clust = count_data.copy()
     country_clust['clustkmeans'] = kmeans.labels_
```

```
[ ]  plt.figure(figsize = (18,8))
     plt.subplot(1, 2, 1)
     sns.scatterplot(x = 'gdpp', y = 'child_mort', data = country_clust, hue = 'clustkmeans', palette = 'bright')
     plt.title('Silhouette Plot Child Mortality')
     plt.xlabel('GDPP')
     plt.ylabel('Child Mortality')
     plt.subplot(1, 2, 2)
     sns.scatterplot(x = 'gdpp', y = 'imports', data = country_clust, hue = 'clustkmeans', palette = 'bright')
     plt.title('Silhouette Plot Imports')
     plt.xlabel('GDPP')
     plt.ylabel('Imports')
     plt.show()
```

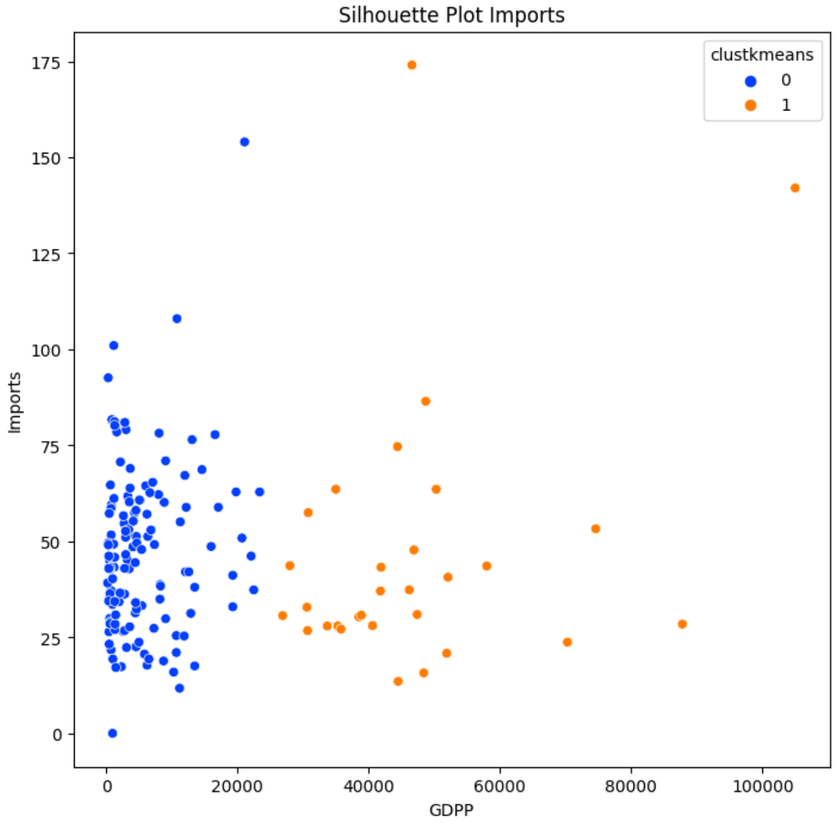
#02 Kmeans clustering

군집화

Gdpp - child_motality



Gdpp - imports



#03 DBSCAN clustering

하이퍼 파라미터 최적화

eps (입실론 반경), min_samples (클러스터가 되기 위해 필요한 최소 데이터 수)

하이퍼 파라미터에 따라 silhouette score 출력

Silhouette score가 높을 수록 좋은 클러스터링

```
scaler = StandardScaler()
country_clust_scaled = scaler.fit_transform(country_clust)

for eps in [i/10 for i in range(2,5)]:
    for min_samples in range (7,9):
        print(f'\neps {eps}')
        print(f'\min samples {min_samples}')

        dbscan = DBSCAN(eps = eps, min_samples = min_samples)
        labels = dbscan.fit_predict(country_clust_scaled)
        score = silhouette_score(country_clust_scaled, labels)

        print(f'clusters present: {np.unique(labels)}')
        print(f'clusters sizes: {np.bincount(labels + 1)}')
        print(f'Silhouette Score: {score}')
```

#03 DBSCAN clustering

하이퍼 파라미터 최적화

```
eps 0.2
\min samples 8
clusters present: [-1  0]
clusters sizes: [156  11]
Silhouette Score: -0.2393615335452477

eps 0.3
\min samples 7
clusters present: [-1  0  1  2]
clusters sizes: [119  30  11   7]
Silhouette Score: -0.1695682589970686

eps 0.3
\min samples 8
clusters present: [-1  0  1]
clusters sizes: [130  29   8]
Silhouette Score: -0.15138644771886964

eps 0.4
\min samples 7
clusters present: [-1  0  1]
clusters sizes: [ 53 101  13]
Silhouette Score: 0.2581064152198133

eps 0.4
\min samples 8
clusters present: [-1  0  1  2]
clusters sizes: [70 78 11  8]
Silhouette Score: 0.11859148004718649
```

Eps = 0.4, min_samples = 7 일 때

Silhouette score가 가장 높음

EWHA EURON

#03 DBSCAN clustering

군집화

eps = 0.4, min_samples = 7 로 군집화 진행 & 산점도 확인

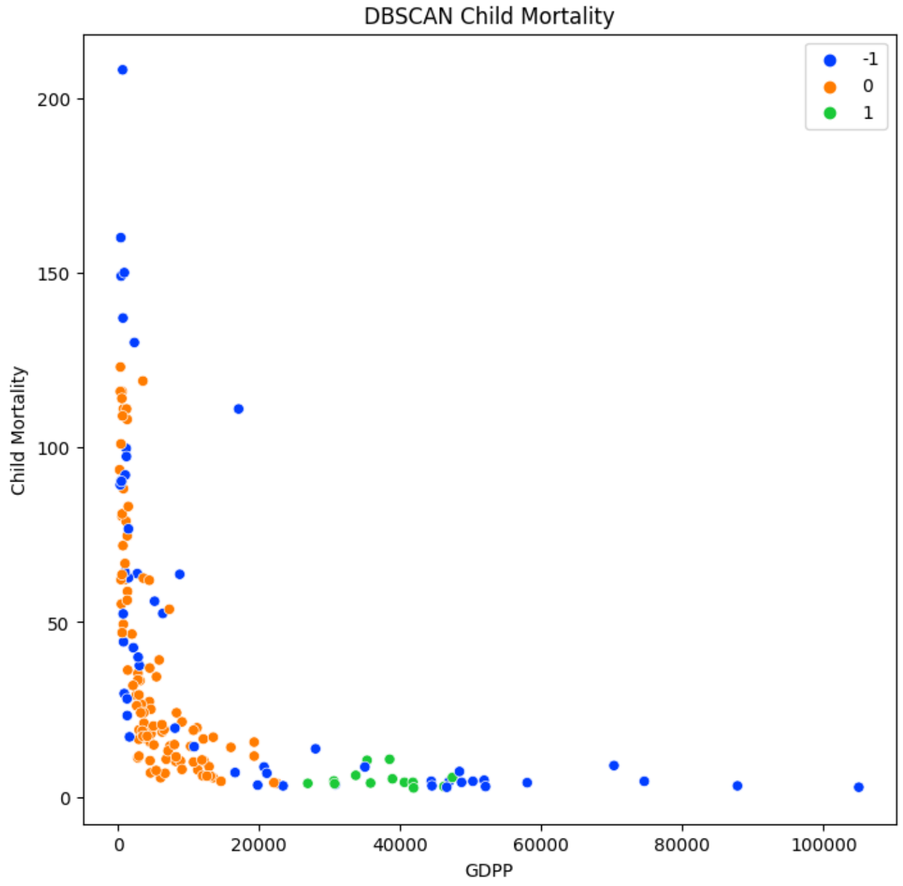
```
dbscan = DBSCAN(eps = 0.4, min_samples = 7)
labels = dbscan.fit_predict(country_clust_scaled)
country_clust['clustdbscan'] = labels
```

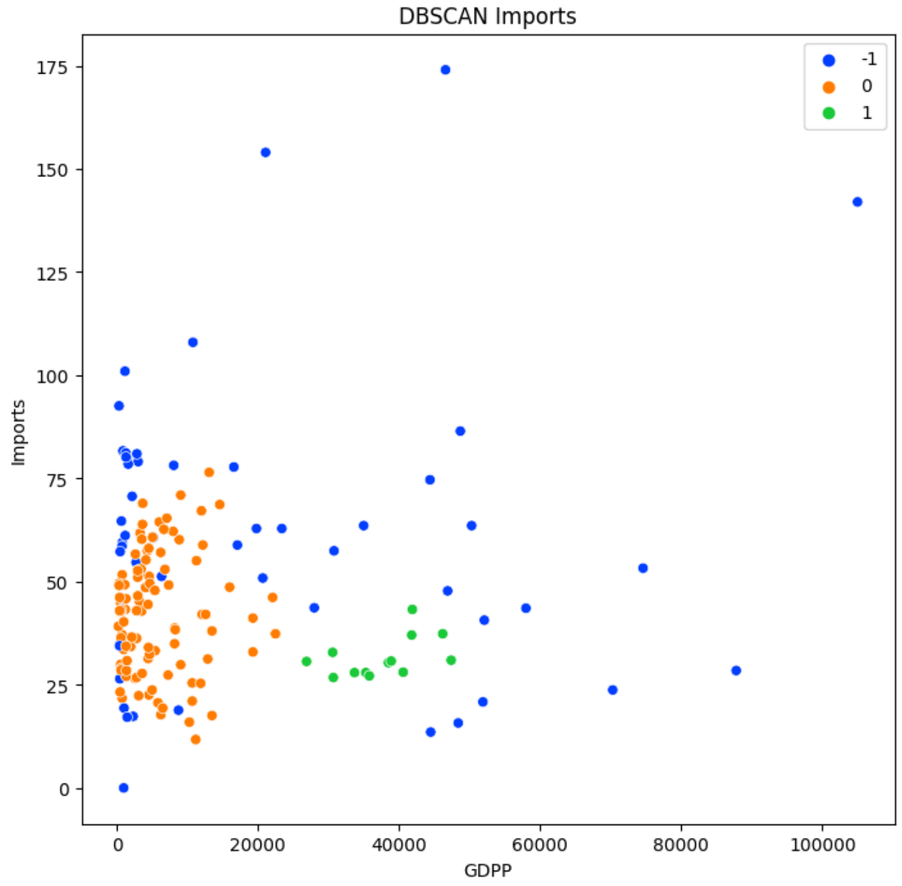
```
plt.figure(figsize = (18,8))
plt.subplot(1, 2, 1)
sns.scatterplot(x = 'gdpp', y = 'child_mort', data = country_clust, hue = 'clustdbscan', palette = 'bright')
plt.legend(loc = 1)
plt.title('DBSCAN Child Mortality')
plt.xlabel('GDPP')
plt.ylabel('Child Mortality')
plt.subplot(1, 2, 2)
sns.scatterplot(x = 'gdpp', y = 'imports', data = country_clust, hue = 'clustdbscan', palette = 'bright')
plt.legend(loc = 1)
plt.title('DBSCAN Imports')
plt.xlabel('GDPP')
plt.ylabel('Imports')
plt.show()
```

#03 DBSCAN clustering

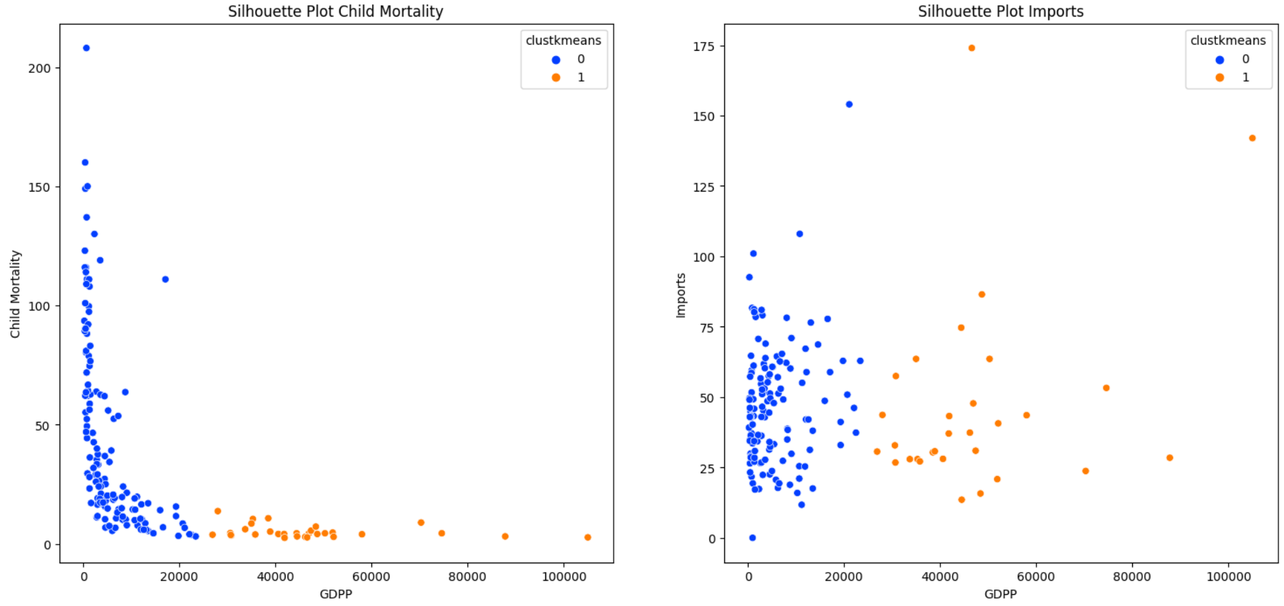
군집화

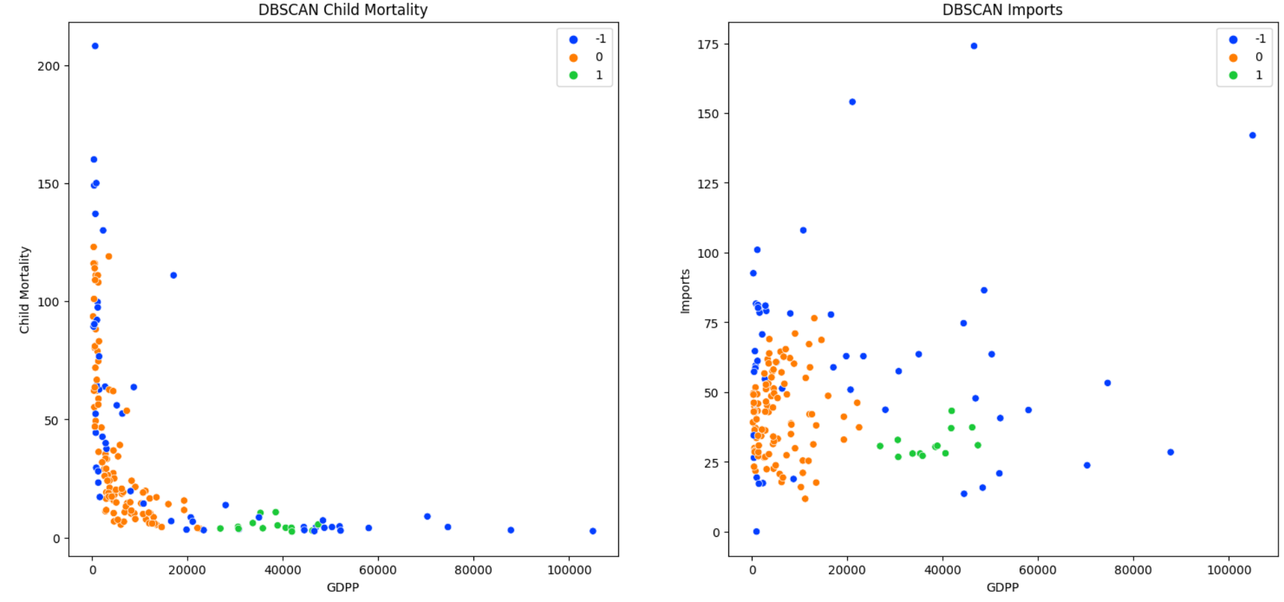
파란 색 = outlier





#04 Kmeans vs DBSCAN clustering





KMeans

초기에 클러스터 개수 설정

centroid 영향 많이 받음

이상치에 취약함

➡️ outlier 포함

DBSCAN

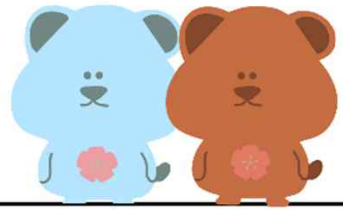
클러스터 개수 미리 지정 X

밀도 기반, 밀도가 낮은 부분

outlier로 걸러낼 수 있음

➡️ 유연한 클러스터링

THANK YOU



EWHA EURON