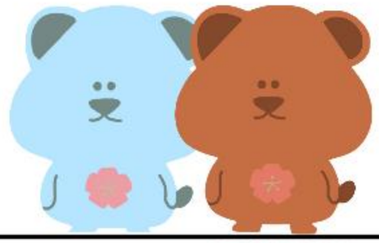
Clustering Kaggle 필사

손소현 박지운 이서영

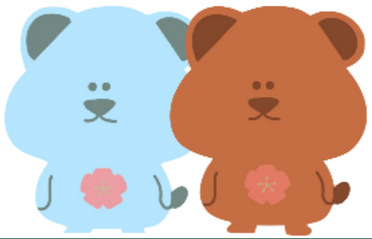
EWHA EURON

목차



EWHA EURON

1. 캐글 Mall Customer Segmentation Data



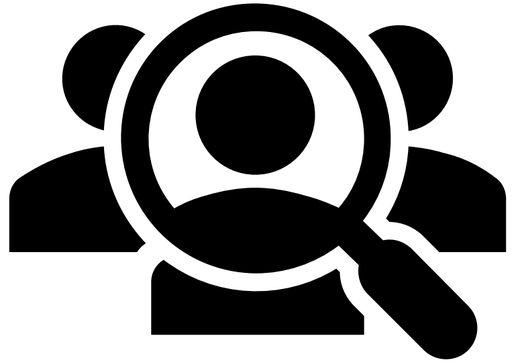
EWHA EURON

1.0 고객 세그먼테이션이란?

- 고객 세그먼테이션(Customer Segmentation) : 다양한 기준으로 고객을 분류하는 기법
- CRM이나 마케팅의 중요기반 요소

- 지역/결혼 여부/성별/소득 과 같은 개인 신상 데이터를 사용하기도 하고
  어떤 상품을 얼마나 많은 비용을 써서 얼마나 자주 사용하는지에 대한 데이터가 더 중요
- 목표? 타깃 마케팅!
  (고객을 여러 특성에 맞게 세분화해서 그 유형에 따라 맞춤형 마케팅이나 서비스를 제공하는 것)

- 기본적인 고객 분석 요소 : **RFM 기법**
  - Recency(R) : 가장 최근 상품 구입일에서 오늘까지의 기간
  - Frequency(F) : 상품구매 횟수
  - Monetary Value(M) : 총 구매금액

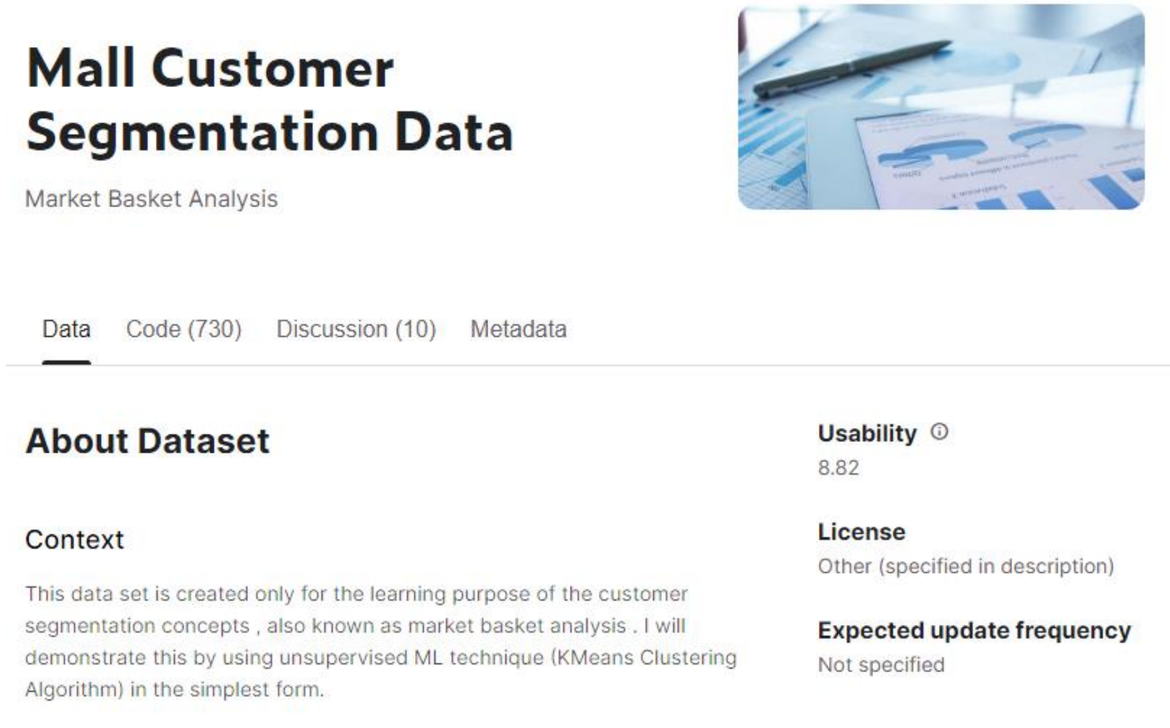


EWHA EURON

1.1 대회 소개

- 4년 전 , 쇼핑몰에서 [타겟 고객]으로 바꿀 수 있는 고객을 이해하기 위하여 마케팅 팀이 그에 따른 전략을 수립하고자 열린 대회

- 비지도학습 중 하나인 군집화 대회이기 때문에 명확한 평가 기준이 없음.



1.2 Data Description

고객이 특정 몰에서 구매한 데이터 200 raw로 구성되어 있음
각 칼럼값은 총 4개이며, 컬럼은 아래과 같음

- CustomerID (고객ID)
- Age (나이)
- Annual Income (k$) (연간 수입)
- Spending Score (1-100) (지출 지수) : 고객 행동이나, 구입 데이터와 같은 파라미터에 기반하여 할당된 점수

1.3 핵심 아이디어/모델링

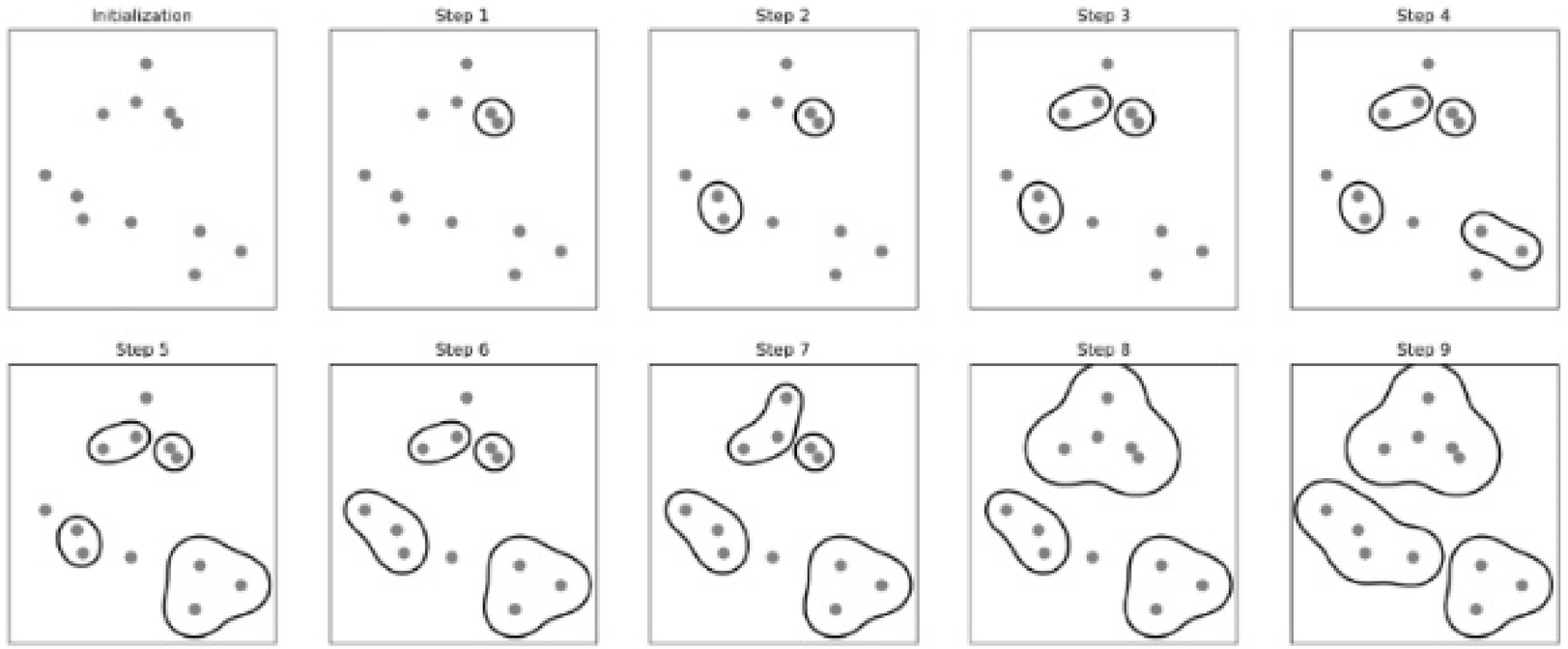
- **K-means** 사용
- 계층적 군집(Hierarchical Clustering),
  그중에서도 **상향식 병합 군집 방식(Agglomerative Clustering)** 사용
  + pca 후 시각화
- **DBSCAN** 사용

1.3 핵심 아이디어/모델링

- 저번에 깊게 다루지 않았던,
  상향식 병합 군집 방식(Agglomerative Clustering)에 대해 살펴봅시다!
- 각각의 데이터 포인트를 하나의 클러스터로 지정하고, 지정된 개수의 클러스터가 남을 때까지 가장 비슷한 두 클러스터를 합쳐 나가는 알고리즘.
- 두 클러스터를 합쳐 나가는 방식에는
- 모든 클러스터 내의 분산을 가장 작게 증가시키는 두 클러스터를 합치는 방식(Ward), 클러스터 포인트 사이의 평균 거리가 가장 짧은 두 클러스터를 합치는 방식(Average), 클러스터 포인트 사이의 최대 거리가 가장 짧은 두 클러스터를 합치는 방식(Complete) 등이 있다. Ward 방식이 대부분의 데이터 세트에 알맞게 동작한다.

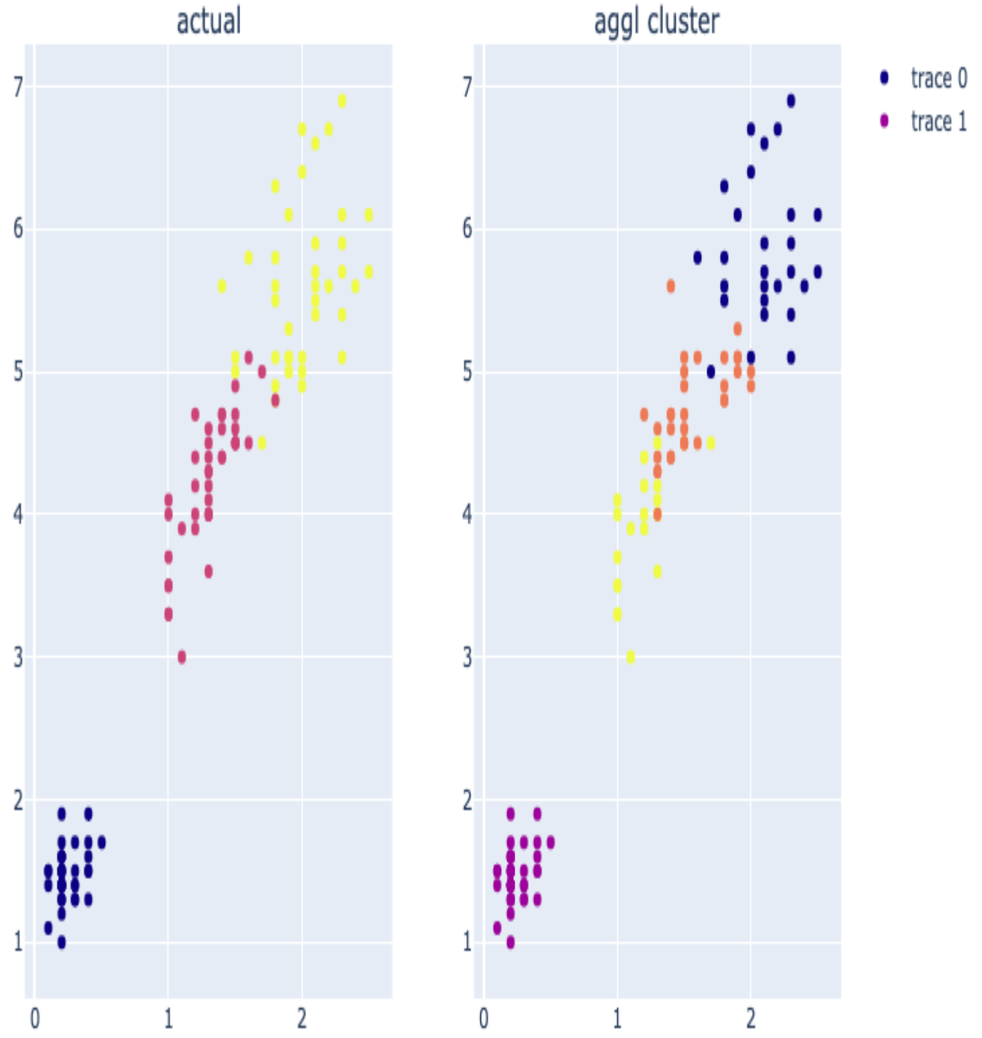
1.3 핵심 아이디어/모델링

- 초기에는 각 포인트가 하나의 클러스터이다. 그 다음 가장 가까운 두 클러스터가 합쳐지기 시작한다.
- 클러스터가 성장하면서 점점 더 멀리 있는 (다른 클러스터보다는 가까운) 데이터를 포함시키고, 지정해준 클러스터 개수가 되면 성장(병합)을 멈춘다.
- 이러한 병합 군집 알고리즘의 작동 특성상 새로운 데이터 포인트에 대해서는 예측을 할 수 없다.



1.3 핵심 아이디어/모델링

특징
- 클러스터의 수를 정하지 않아도 사용가능
- Random Point에서 시작하지 않으므로, 동일한 결과 나옴
- 덴드로그램을 통해 전체 적인 군집 확인 가능
- 대용량 데이터에 비효율적 (계산량이 많음)



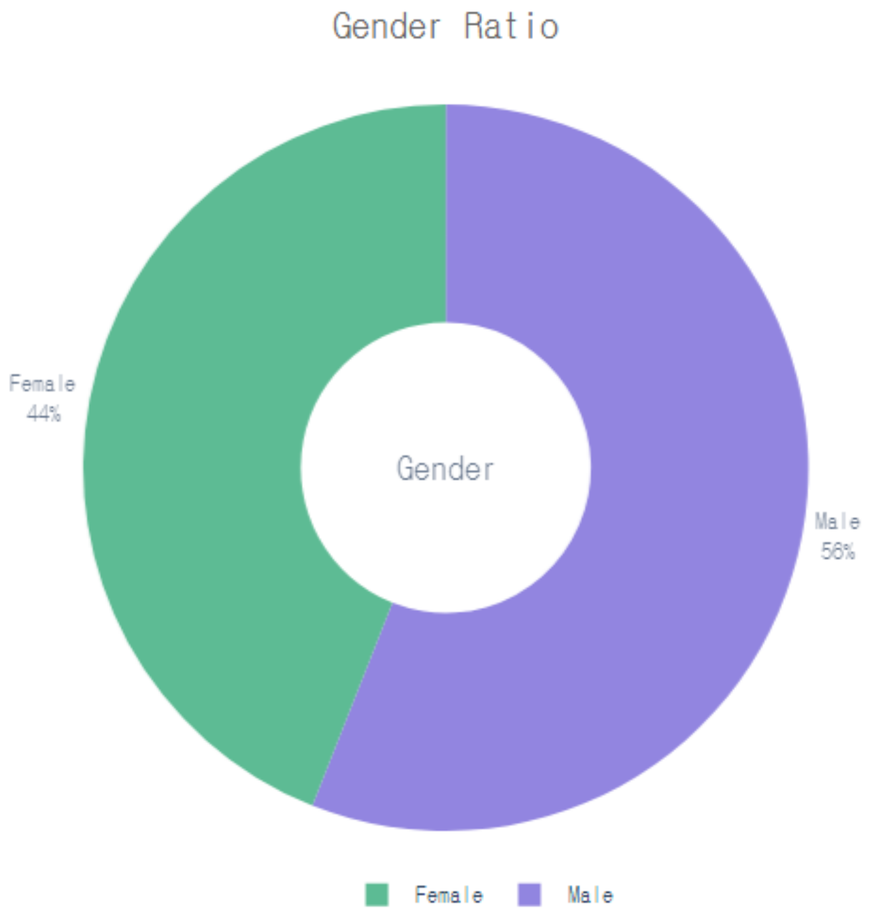
1.4 Data analysis processing에 따른 코드/결과 분석

① EDA
② Data preprocessing
③ K-means
④ 상향식 병합 군집 방식(Agglomerative Clustering)
⑤ DBSCAN

1.4 Data analysis processing에 따른 코드/결과 분석

① EDA – gender

```
d= pd.DataFrame(df['Gender'].value_counts())
fig = px.pie(d,values='Gender',names=['Male','Female'],hole=0.4,opacity=0.7,
             color_discrete_sequence=[colors_mix[7],colors_mix[2]])

fig.add_annotation(text='Gender',
                   x=0.5,y=0.5,showarrow=False,font_size=18,opacity=0.7,font_far

fig.update_layout(
    font_family='monospace',
    title=dict(text='Gender Ratio',x=0.5,y=0.98,
               font=dict(color=colors_dark[2],size=20)),
    legend=dict(x=0.37,y=-0.05,orientation='h',traceorder='reversed'),
    hoverlabel=dict(bgcolor='white'))

fig.update_traces(textposition='outside', textinfo='percent+label')

fig.show()
```



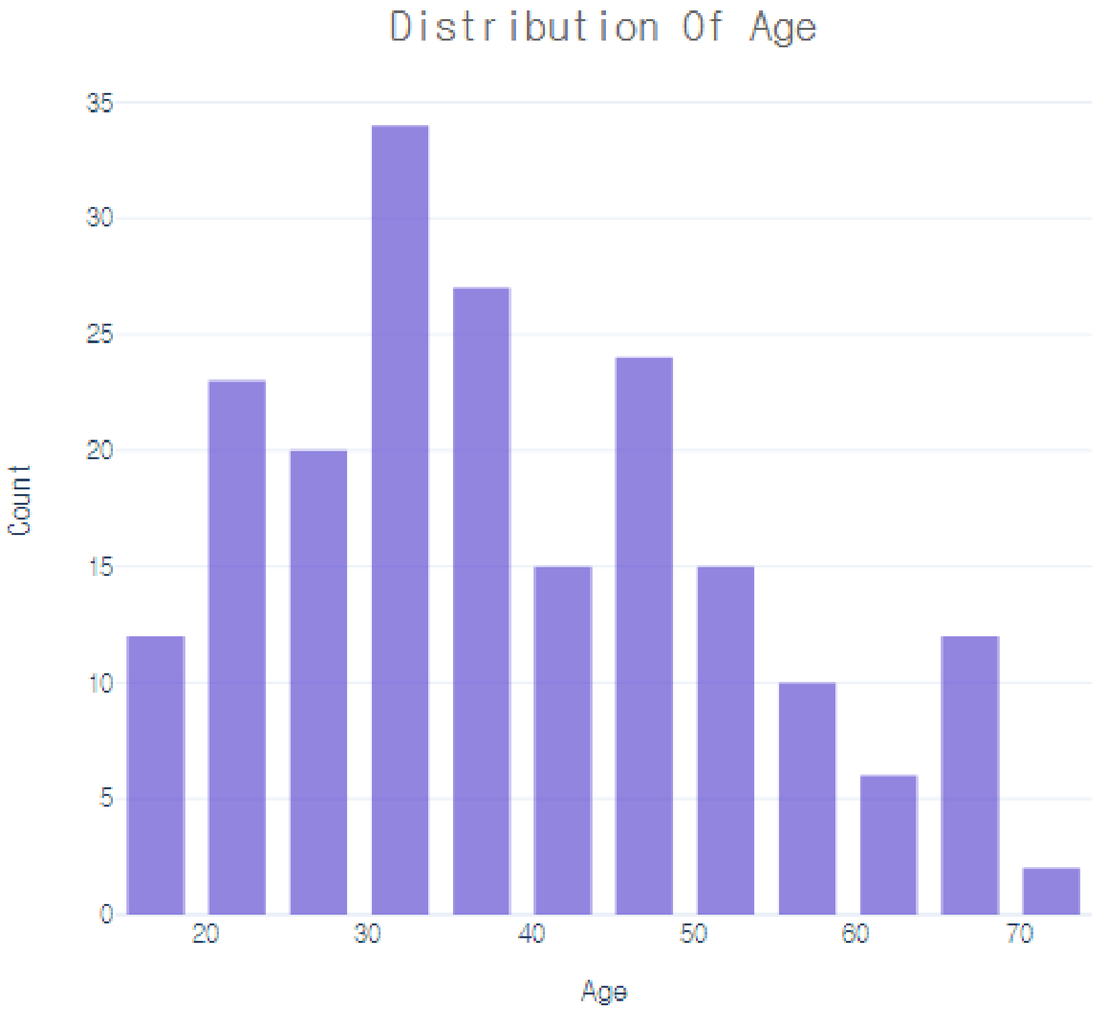
Female이 44%, Male이 56%로 남자가 조금 더 많으나, 반반에 가까움

-> 그래서 인지 많은 노트북에서 gender변수는 사용하지 않음

1.4 Data analysis processing에 따른 코드/결과 분석

① EDA – age

```
fig = px.histogram(df,x='Age',template='plotly_white',opacity=0.7,nbins=25,
                   color_discrete_sequence=[colors_mix[7]])

fig.update_layout(
    font_family='monospace',
    title=dict(text='Distribution Of Age',x=0.5,y=0.95,
               font=dict(color=colors_dark[2],size=20)),
    xaxis_title_text='Age',
    yaxis_title_text='Count',
    legend=dict(x=1,y=0.96,bordercolor=colors_dark[4],borderwidth=0,tracegroupga
    bargap=0.3,
)
fig.show()
```

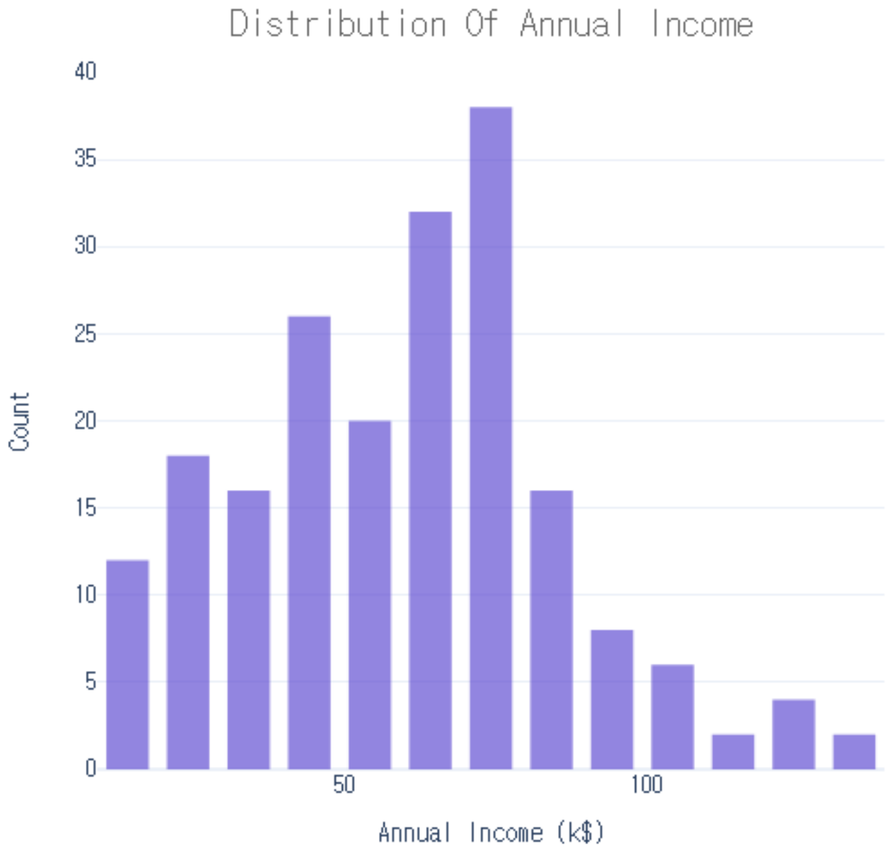


약간 왼쪽으로 치우쳐있음. 203040에 몰려있음.

1.4 Data analysis processing에 따른 코드/결과 분석

① EDA – annual income

```
fig = px.histogram(df,x='Annual Income (k$)',template='plotly_white',opacity=0.7
                   color_discrete_sequence=[colors_mix[7]])

fig.update_layout(
    font_family='monospace',
    title=dict(text='Distribution Of Annual Income',x=0.5,y=0.95,
               font=dict(color=colors_dark[2],size=20)),
    xaxis_title_text='Annual Income (k$)',
    yaxis_title_text='Count',
    legend=dict(x=1,y=0.96,bordercolor=colors_dark[4],borderwidth=0,tracegroupga
    bargap=0.3,
)
fig.show()
```

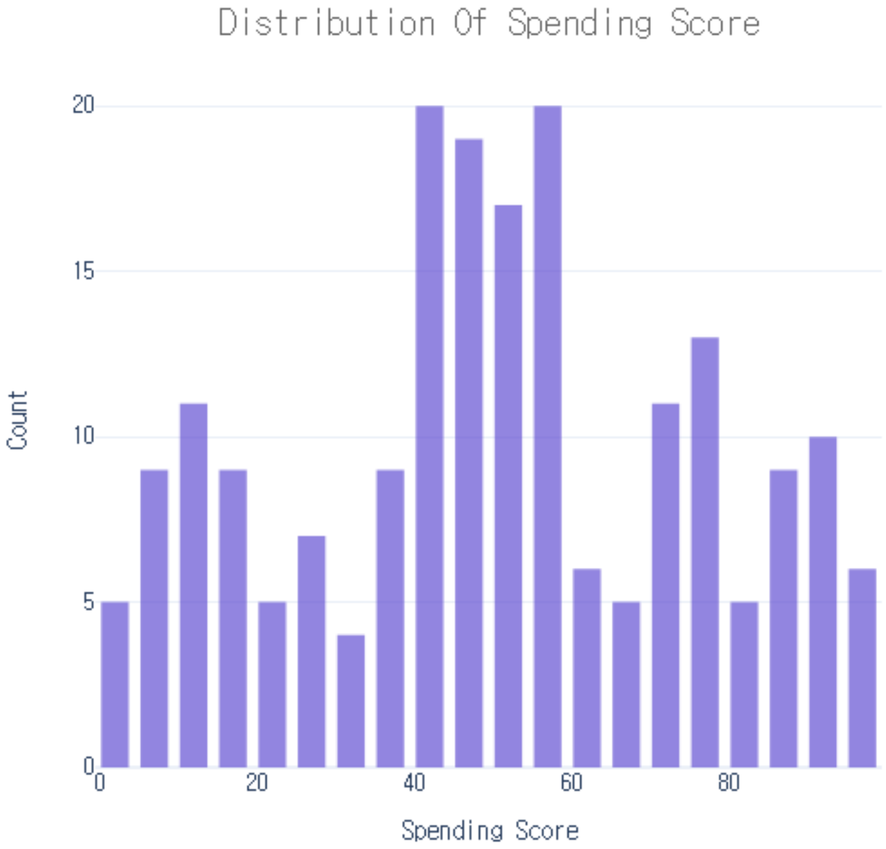


Age보다 더 왼쪽으로 치우쳐있음. 1000이상은 거의 없음.

1.4 Data analysis processing에 따른 코드/결과 분석

① EDA – spending score

```
fig = px.histogram(df,x='Spending Score (1-100)',template='plotly_white',opacity=
                   color_discrete_sequence=[colors_mix[7]])

fig.update_layout(
    font_family='monospace',
    title=dict(text='Distribution Of Spending Score',x=0.5,y=0.95,
               font=dict(color=colors_dark[2],size=20)),
    xaxis_title_text='Spending Score',
    yaxis_title_text='Count',
    legend=dict(x=1,y=0.96,bordercolor=colors_dark[4],borderwidth=0,tracegroupga
    bargap=0.3,
)
fig.show()
```



거의 완벽한 대칭모양을 보임. 40~60이 가장 많음

1.4 Data analysis processing에 따른 코드/결과 분석

② Data preprocessing

```
df.drop('CustomerID',axis=1,inplace=True)
```

```
df.columns
```

```
Index(['Gender', 'Age', 'Annual Income (k$)', 'Spending Score (1-100)'], dtype='obje
ct')
```

```
df['Gender'] = df['Gender'].apply(lambda x: 0 if x=='Male' else 1)
```

```
df['Gender']
```

```
0      0
1      0
2      1
3      1
4      1
      ..
195    1
196    1
197    0
198    0
199    0
Name: Gender, Length: 200, dtype: int64
```

We'll scale the data as algorithms like K-Means Clustering uses Euclidean Distance and other such distance metrics for computation of distances between data points, therefore making it sensitive to outliers.

```
scaler = StandardScaler()
scaler.fit(df)
X = scaler.transform(df)
```

Customer ID 칼럼은 분석에 불필요하므로 삭제

Female, Male로 나와있는 gende칼럼을,
각각 1과0으로 바꿈

K-means가 유클리시안 거리와 같은 거리계산을 하므로
이상치에 민감 -> StandardScale

1.4 Data analysis processing에 따른 코드/결과 분석

③ K-means

```
wcss= []    # within cluster sum of squares
ss = []     # silouette score
for i in range(2,11):
    model = KMeans(n_clusters=i)
    model.fit_transform(X)
    wcss.append(model.inertia_)
    ss.append(silhouette_score(X,labels=model.predict(df)))
```

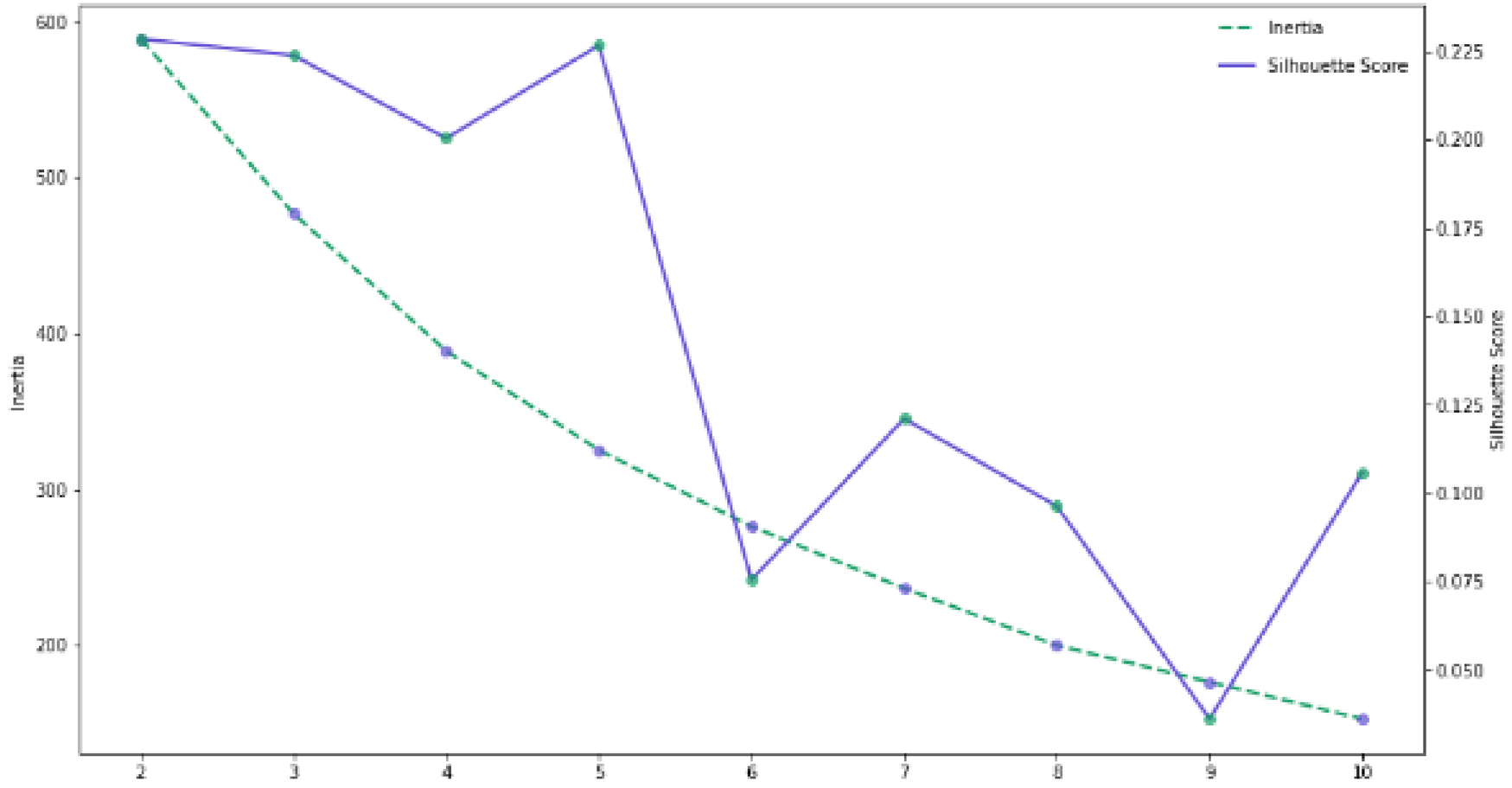
n cluster를 2부터 11까지 반복하여 모델 생성

```
fig,ax1 = plt.subplots(figsize=(14,8))

ax1.plot(range(2, 11), wcss , '--', color=colors_mix[2], linewidth=2)
ax1.legend(['Inertia'],bbox_to_anchor=(0.9365,1),frameon=False)
ax1.plot(range(2, 11), wcss , 'o', color=colors_mix[7],alpha=0.7)
ax1.set_ylabel('Inertia')

ax2 = ax1.twinx()
ax2.plot(range(2, 11), ss, '-', color=colors_mix[7], linewidth=2)
ax2.legend(['Silhouette Score'],bbox_to_anchor=(1,0.95),frameon=False)
ax2.plot(range(2, 11), ss, 'o', color=colors_mix[2], alpha=0.7)
ax2.set_ylabel('Silhouette Score')

plt.xlabel('Number of clusters')
plt.show()
```

생성된 모델 각각에 대한 실루엣계수 그래프 그리기

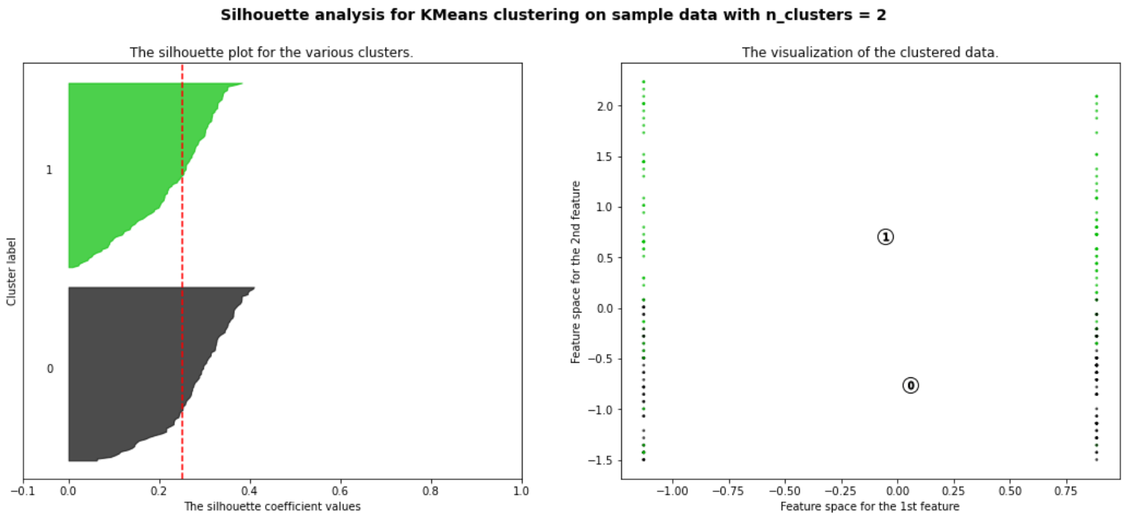
1.4 Data analysis processing에 따른 코드/결과 분석

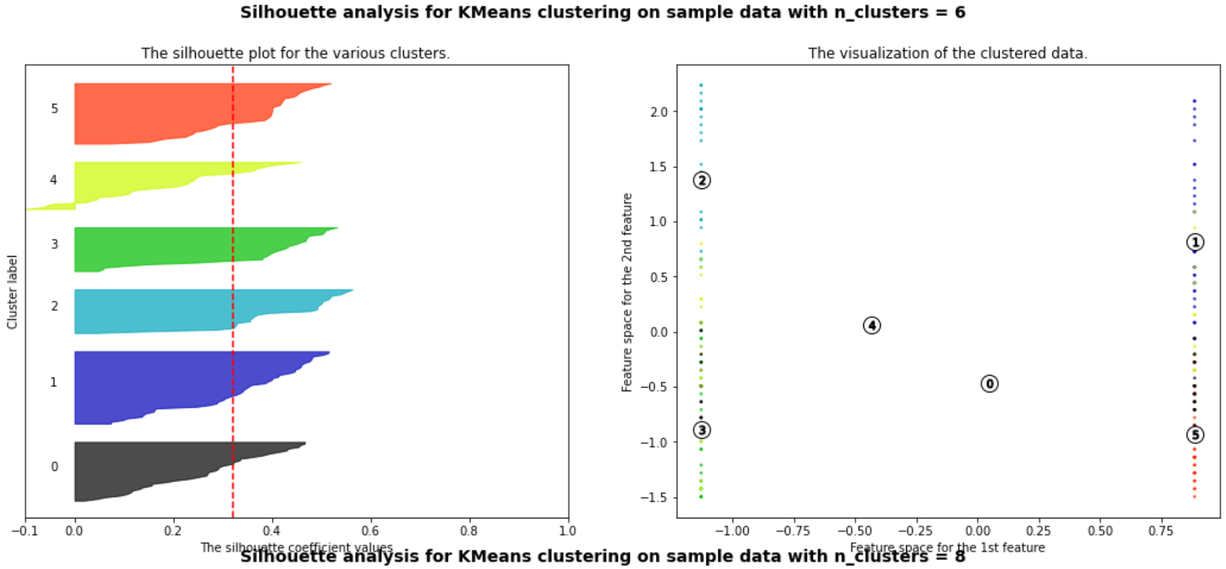
③ K-means

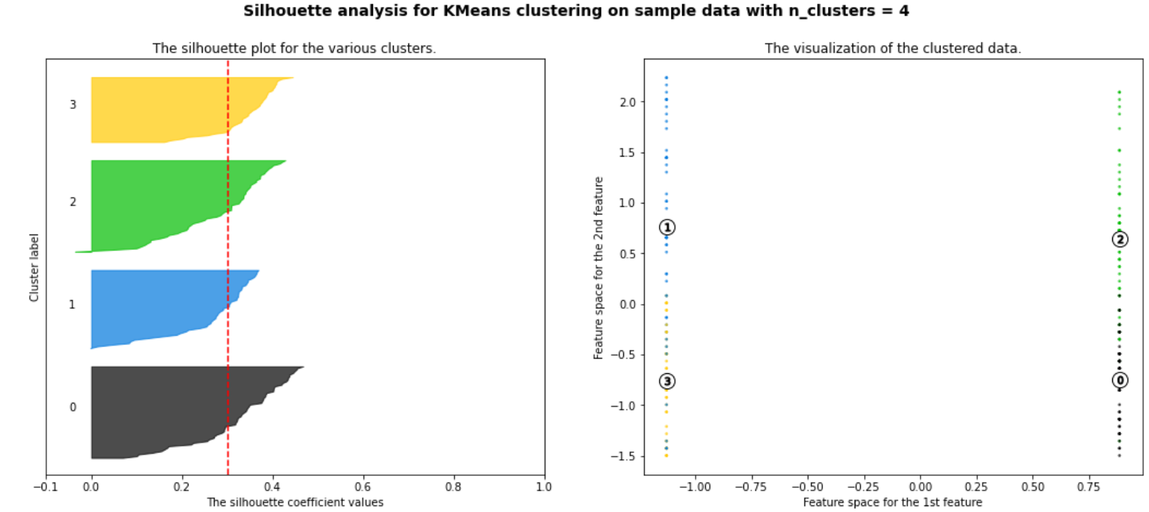


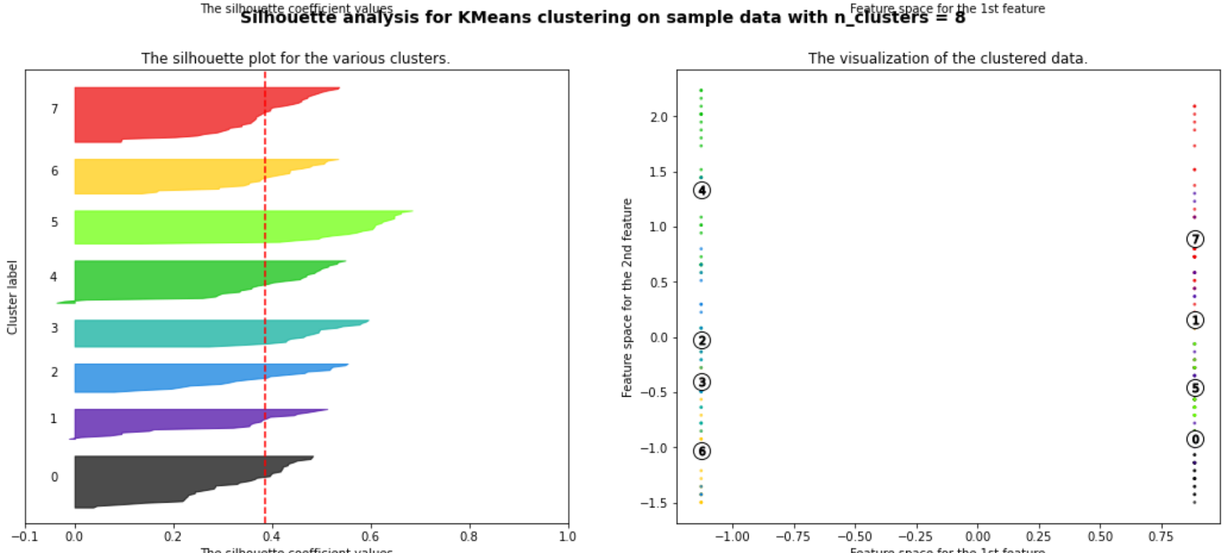
1.4 Data analysis processing에 따른 코드/결과 분석

③ K-means





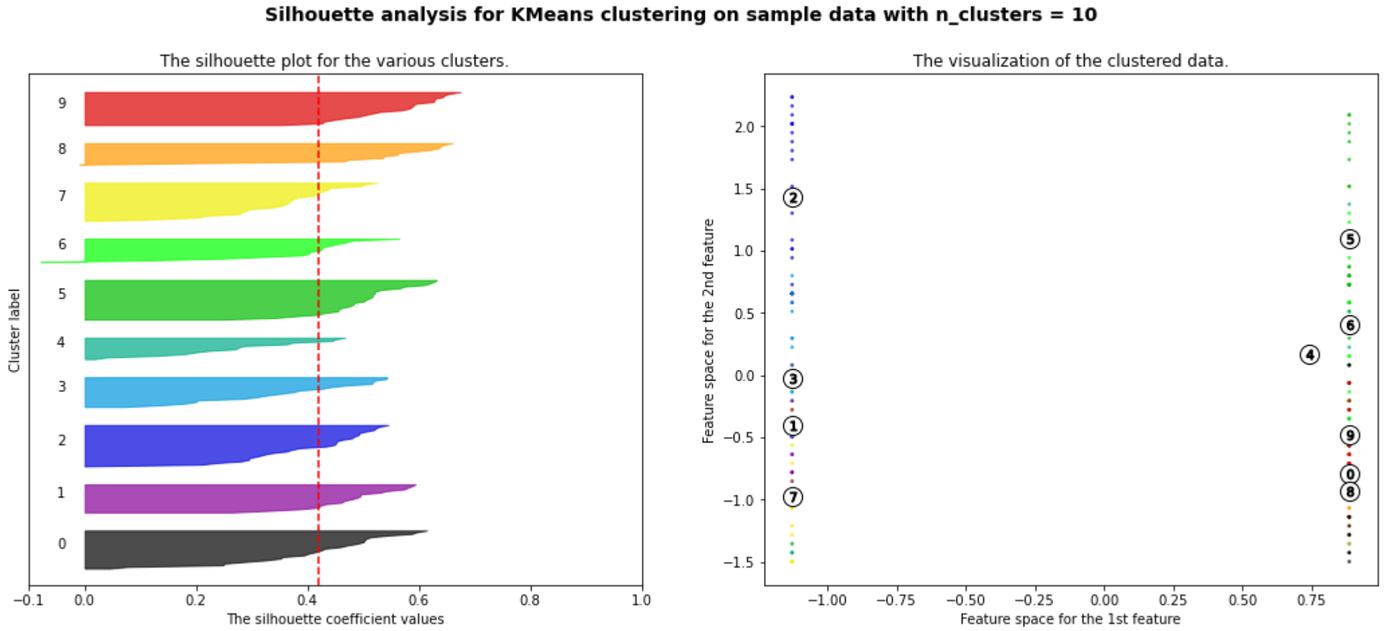




1.4 Data analysis processing에 따른 코드/결과 분석

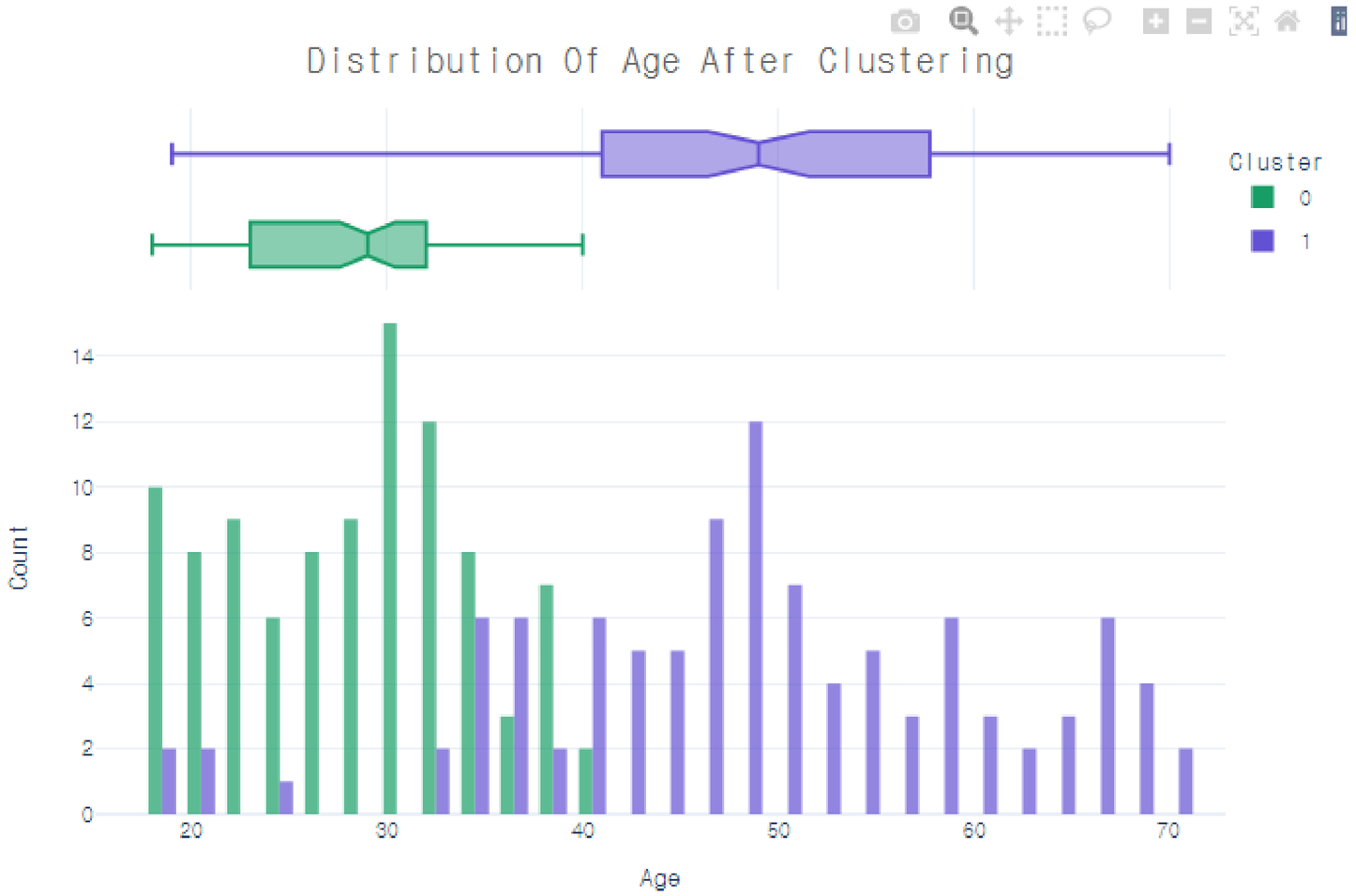
③ K-means

```
For n_clusters = 2 The average silhouette_score is : 0.25181529157884364
For n_clusters = 4 The average silhouette_score is : 0.30123231688013513
For n_clusters = 6 The average silhouette_score is : 0.3199872749106995
For n_clusters = 8 The average silhouette_score is : 0.38738083581583793
For n_clusters = 10 The average silhouette_score is : 0.420111981176221 34
```



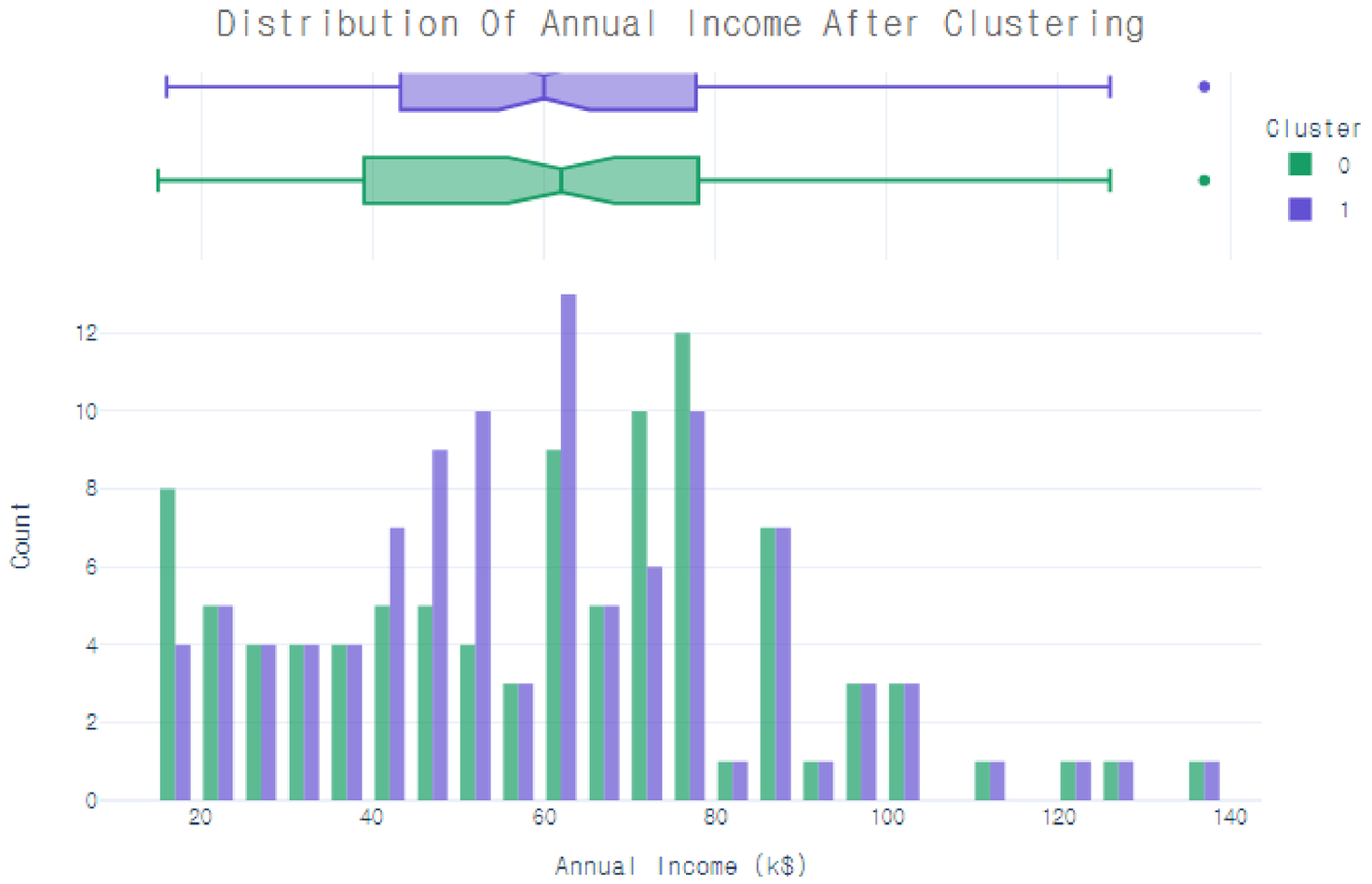
1.4 Data analysis processing에 따른 코드/결과 분석

③ K-means



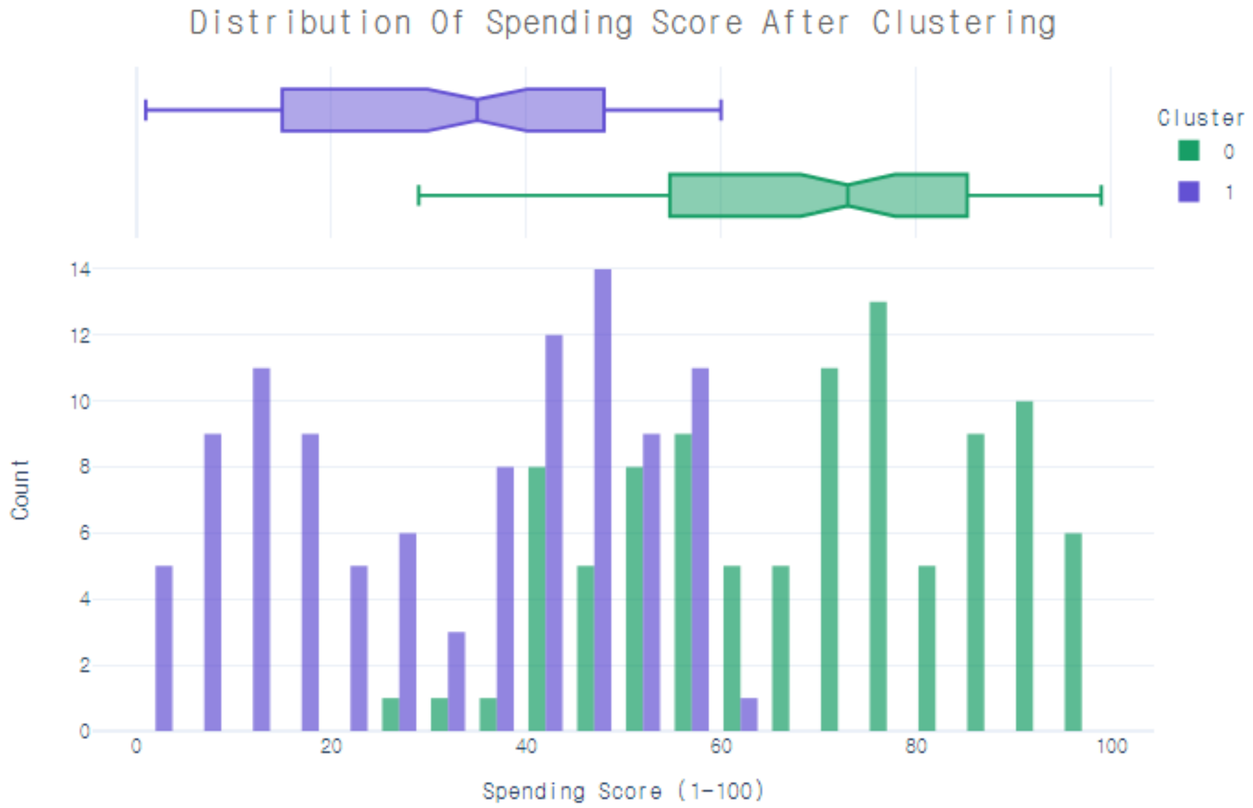
1.4 Data analysis processing에 따른 코드/결과 분석

③ K-means



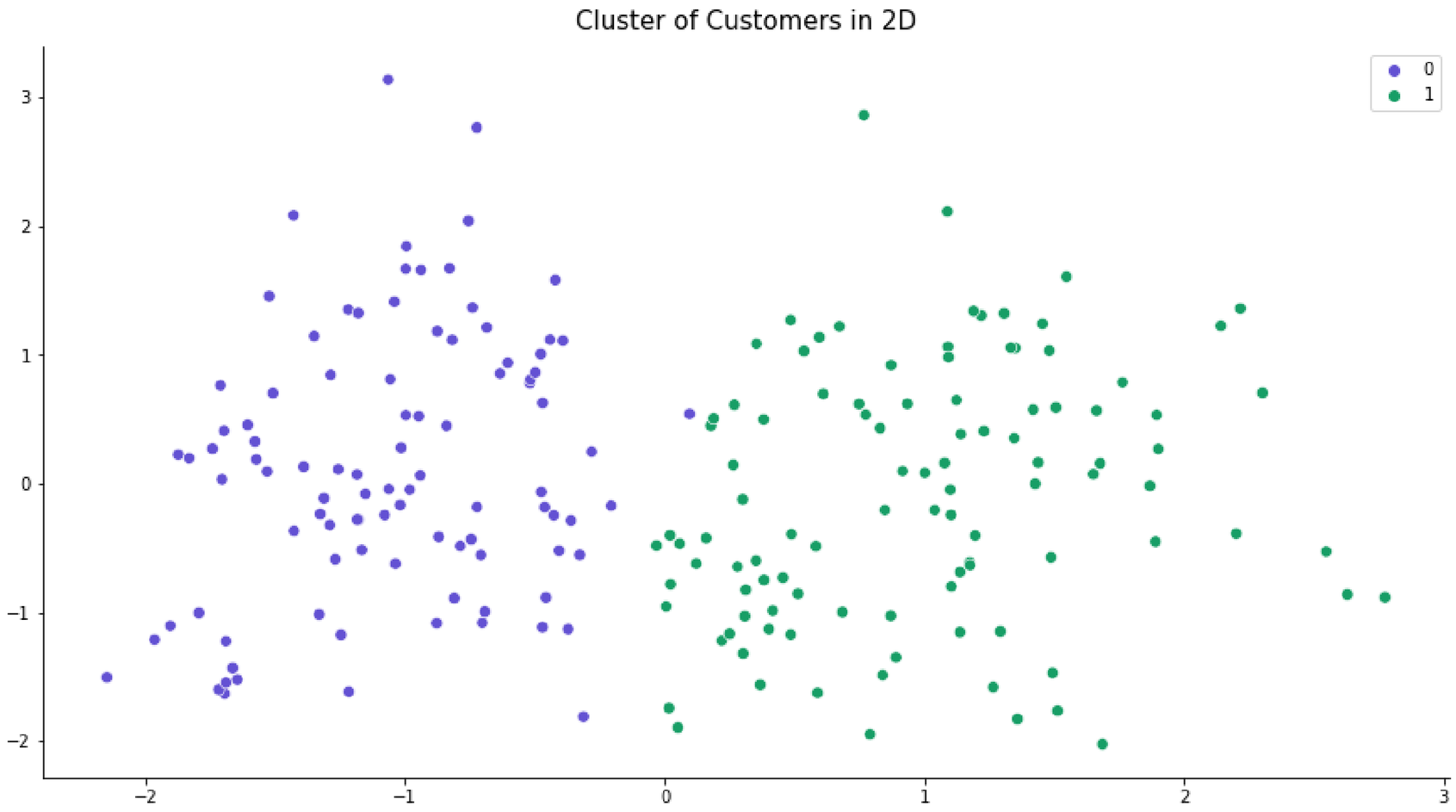
1.4 Data analysis processing에 따른 코드/결과 분석

③ K-means



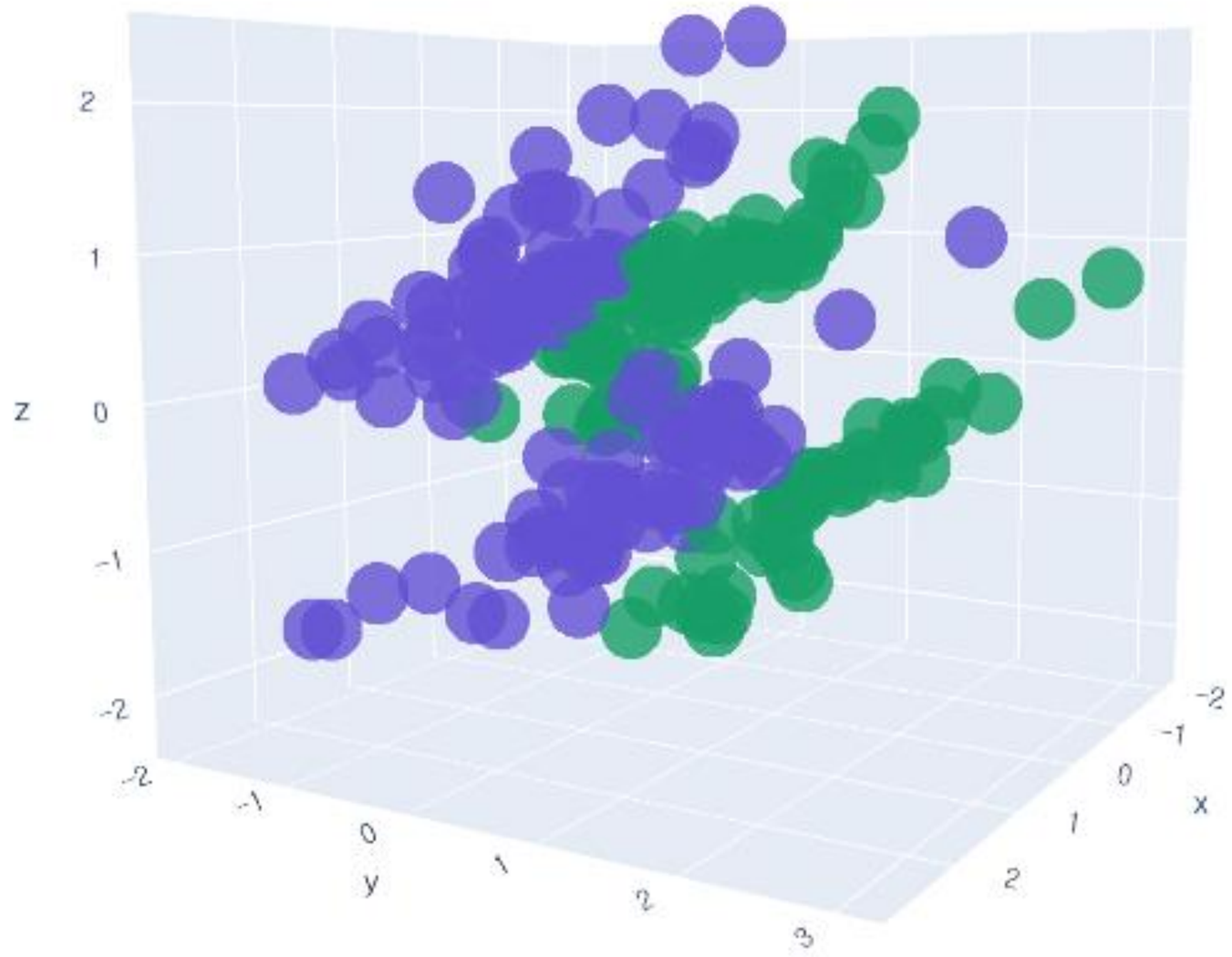
1.4 Data analysis processing에 따른 코드/결과 분석

③ K-means – pca(2)



1.4 Data analysis processing에 따른 코드/결과 분석

③ K-means – pca(3)



1.4 Data analysis processing에 따른 코드/결과 분석

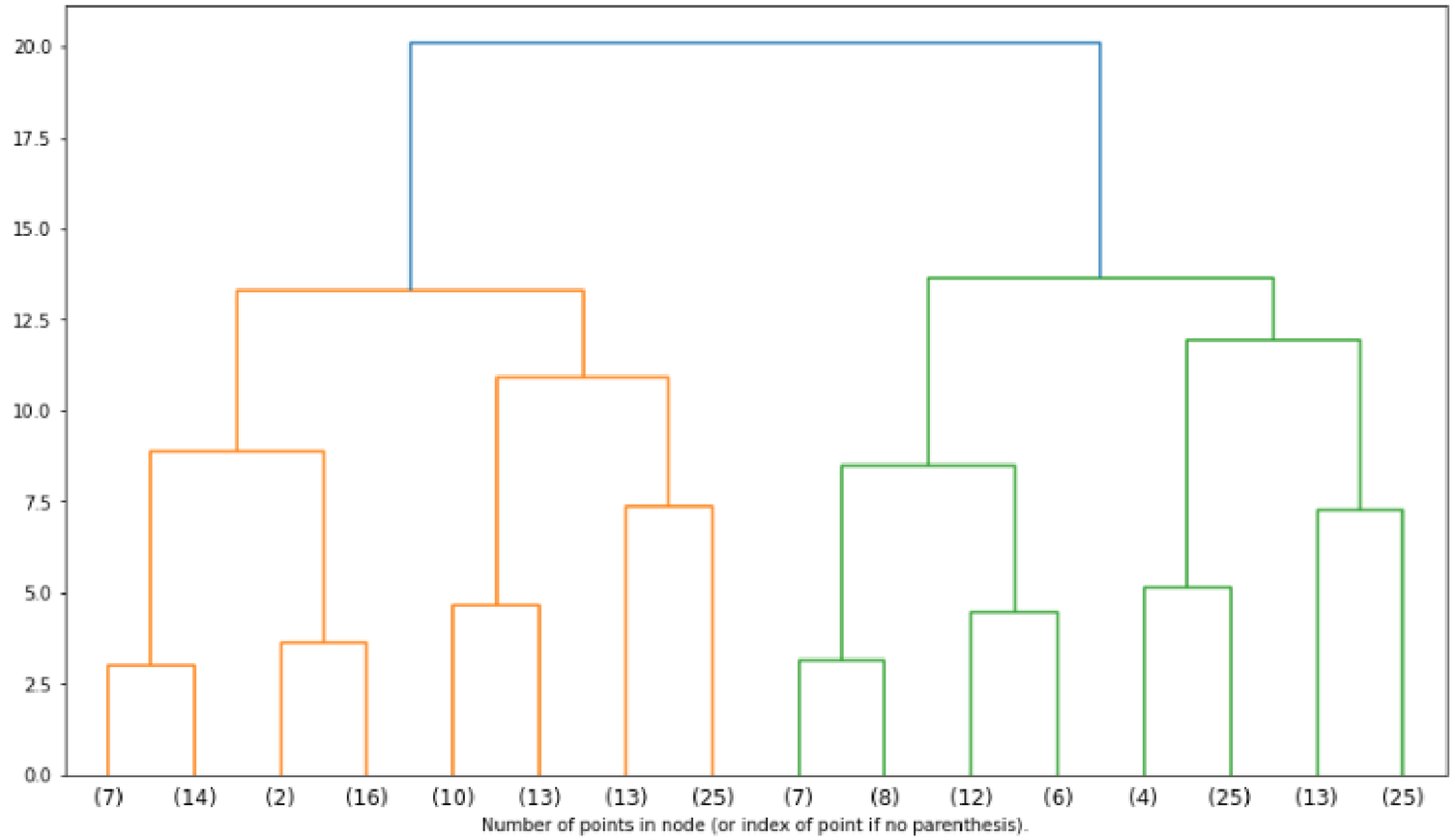
④ 상향식 병합 군집 방식(Agglomerative Clustering)

```
def plot_dendrogram(model, **kwargs):
    # Create linkage matrix and then plot the dendrogram

    # create the counts of samples under each node
    counts = np.zeros(model.children_.shape[0])
    n_samples = len(model.labels_)
    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1  # leaf node
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count

    linkage_matrix = np.column_stack([model.children_, model.distances_,
                                      counts]).astype(float)

    # Plot the corresponding dendrogram
    dendrogram(linkage_matrix, **kwargs)
```

1.4 Data analysis processing에 따른 코드/결과 분석

④ 상향식 병합 군집 방식(Agglomerative Clustering)
-> 2개의 뚜렷한 클러스터를 확인가능

```
plt.figure(figsize=(14,8))
plot_dendrogram(model,truncate_mode = 'level',p=3)
plt.xlabel("Number of points in node (or index of point if no parenthesis).")
plt.show()
```



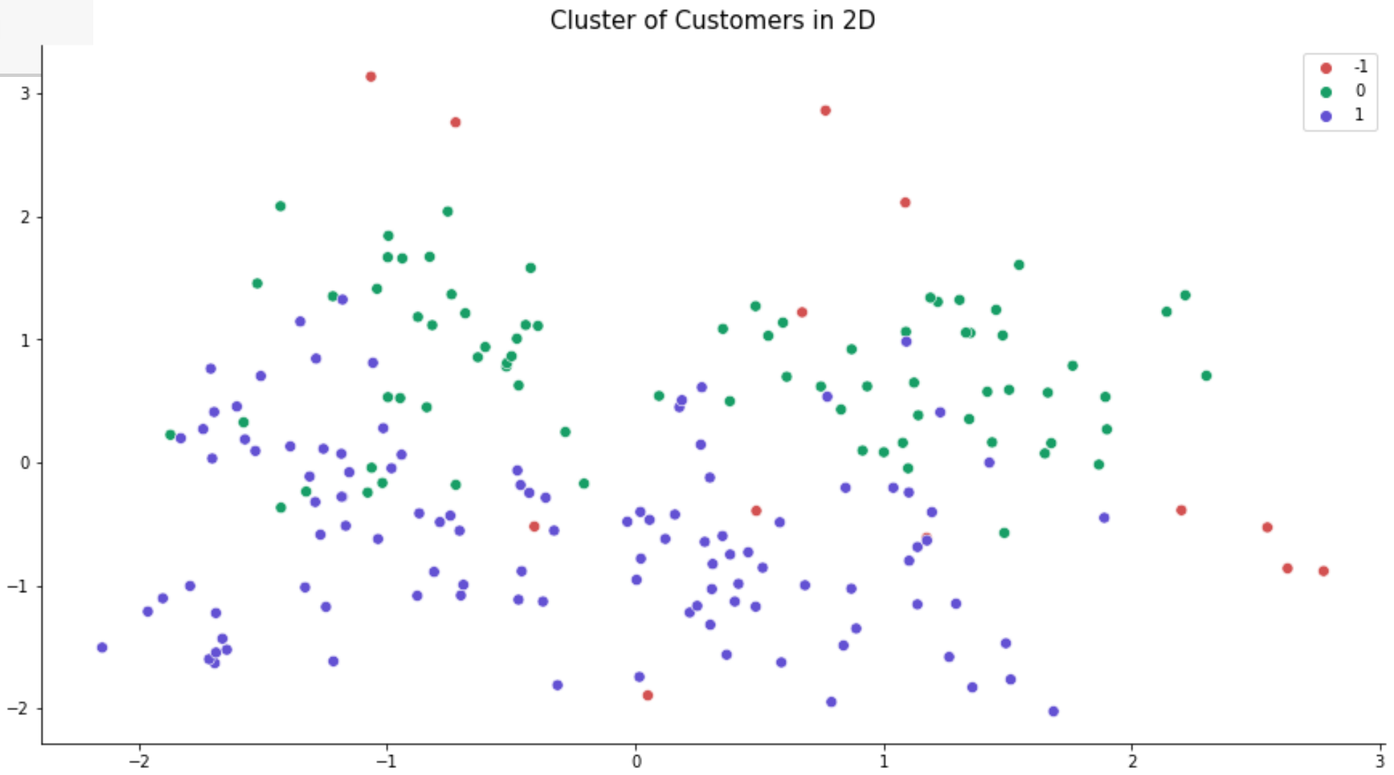
1.4 Data analysis processing에 따른 코드/결과 분석

⑤ DBSCAN – pca(2)

```
model = DBSCAN(eps=1,min_samples=5)
cluster_labels = model.fit_predict(X)
```

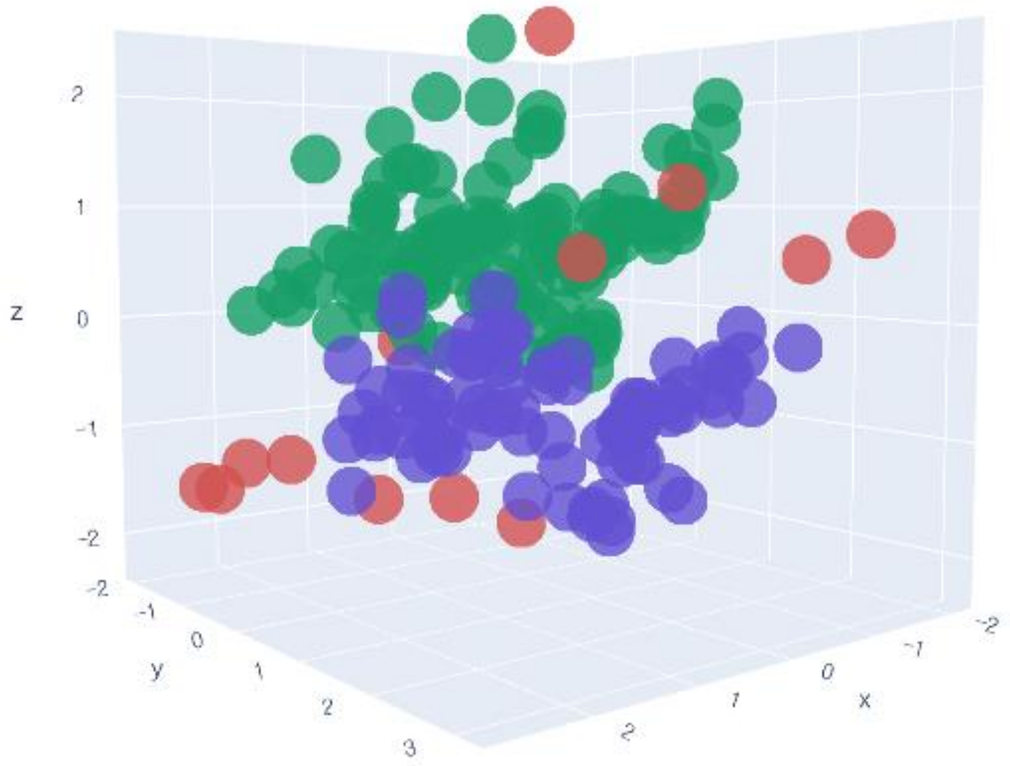
```
silhouette_score(X,cluster_labels)
```

0.25438718242952O2



1.4 Data analysis processing에 따른 코드/결과 분석

⑤ DBSCAN – pca(3)

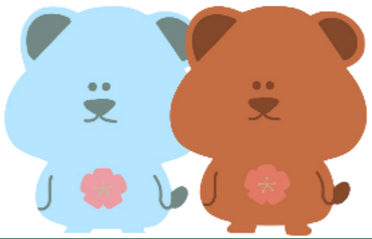


1.5 노트의 결론

- 실루엣 계수를 통해 올바른 값을 찾음
- K-means는 2개의 클러스터가 연령과 지출점수를 기반으로 고객을 매우 뚜렷하게 구분
- 계층적 클러스터링을 사용하면 2개의 클러스터가 형성됨
- DBSCAN을 사용하여 형성된 클러스터는 k-means와 다르며, 이상치가 발견됨

2. H&M recommendation

- GMM clustering



EWHA EURON

2.1 H&M Data EDA

3개의 csv 파일 및 이미지 파일

1. articles.csv

article_id **: A unique identifier of every article.**

product_code, prod_name **: A unique identifier of every product and its name (not the same).**

product_type, product_type_name **: The group of product_code and its name**

graphical_appearance_no, graphical_appearance_name **: The group of graphics and its name**

colour_group_code, colour_group_name **: The group of color and its name**

perceived_colour_value_id, perceived_colour_value_name, perceived_colour_master_id, perceived_colour_master_name **: The added color info**

department_no, department_name: **: A unique identifier of every dep and its name**

index_code, index_name: **: A unique identifier of every index and its name**

index_group_no, index_group_name: **: A group of indeces and its name**

section_no, section_name: **: A unique identifier of every section and its name**

garment_group_no, garment_group_name: **: A unique identifier of every garment and its name**

detail_desc: **: Details**

2. customers.csv

customer_id **: A unique identifier of every customer**

FN **: 1 or missed**

Active **: 1 or missed**

club_member_status **: Status in club**

fashion_news_frequency **: How often H&M may send news to customer**

age **: The current age**

postal_code **: Postal code of customer**

3. transactions.csv

t_dat **: A unique identifier of every customer**

customer_id **: A unique identifier of every customer (in** customers **table)**

article_id **: A unique identifier of every article (in** articles **table)**

price **: Price of purchase**

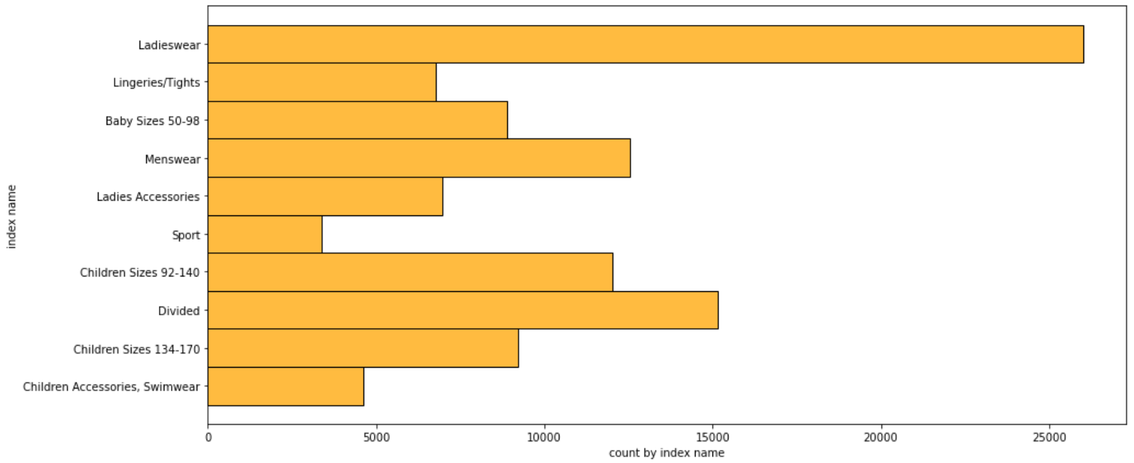
sales_channel_id **: 1 or 2**

출처: H&M EDA FIRST LOOK | Kaggle

2.1 H&M Data EDA – articles.csv

→ contains all h&m articles with details such as a type of product, a color, a product group and other features.

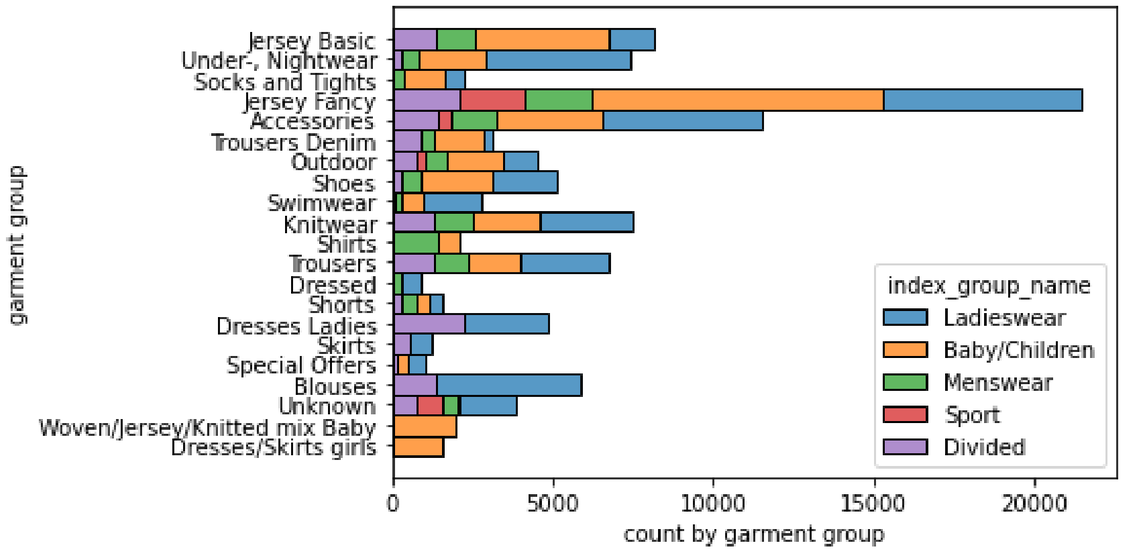
1. index_name별 분포 살펴보기



→ Ladieswear가 가장 많은 비율을 차지함
→ Sportswear가 가장 적은 비율을 차지함

+ Ladieswear과 baby,children은 group 내에 subgroup 또한 존재

2. garment_group, index_name별 분포 살펴보기



→ Jersey Fancy가 가장 많은 비율 차지,
그 중에서 Ladieswear과 baby,children용이 많다.

index_group_name	index_name	
Baby/Children	Baby Sizes 50-98	8875
	Children Accessories, Swimwear	4615
	Children Sizes 134-170	9214
	Children Sizes 92-140	12007
Divided	Divided	15149
Ladieswear	Ladies Accessories	6961
	Ladieswear	26001
	Lingeries/Tights	6775
Menswear	Menswear	12553
Sport	Sport	3392

Name: article_id, dtype: int64

2.1 H&M Data EDA – articles.csv

→ contains all h&m articles with details such as a type of product, a color, a product group and other features.

☑ 소비자에 따라 article분류하기

→ H&M 홈페이지를 보고, index_name별 소비자 구분하기(for men, for women, for mama)

```
df_article = pd.read_csv('C:/Users/USER/Documents/h-and-m-personalized-fashion-recommendat

def set_gender_flg(x):
    x['is_for_male'] = 0
    x['is_for_female'] = 0
    x['is_for_mama'] = 0
    if x['index_group_name'] in ['Ladieswear','Divided']:
        x['is_for_female'] = 1
    if x['index_group_name'] == 'Menswear':
        x['is_for_male'] = 1
    if x['index_group_name'] in ['Baby/Children','Sport']:
        if 'boy' in x['department_name'].lower() or 'men' in x['department_name'].lower():
            x['is_for_male'] = 1
        if 'girl' in x['department_name'].lower() or 'ladies' in x['department_name'].lowe
            x['is_for_female'] = 1
    if x['section_name'] == 'Mama':
        x['is_for_mama'] = 1
    return x

df_article = df_article.apply(set_gender_flg, axis=1)
df_article.head()
```

→

_name	section_no	section_name	garment_group_no	garment_group_name	detail_desc	is_for_male	is_for_female	is_for_mama
	16	Womens Everyday Basics	1002	Jersey Basic	Jersey top with narrow shoulder straps.	0	1	0
	16	Womens Everyday Basics	1002	Jersey Basic	Jersey top with narrow shoulder straps.	0	1	0
	16	Womens Everyday Basics	1002	Jersey Basic	Jersey top with narrow shoulder straps.	0	1	0
	61	Womens Lingerie	1017	Under-, Nightwear	Microfibre T-shirt bra with underwired, moulde...	0	1	0
	61	Womens Lingerie	1017	Under-, Nightwear	Microfibre T-shirt bra with underwired, moulde...	0	1	0

2.1 H&M Data EDA – articles.csv

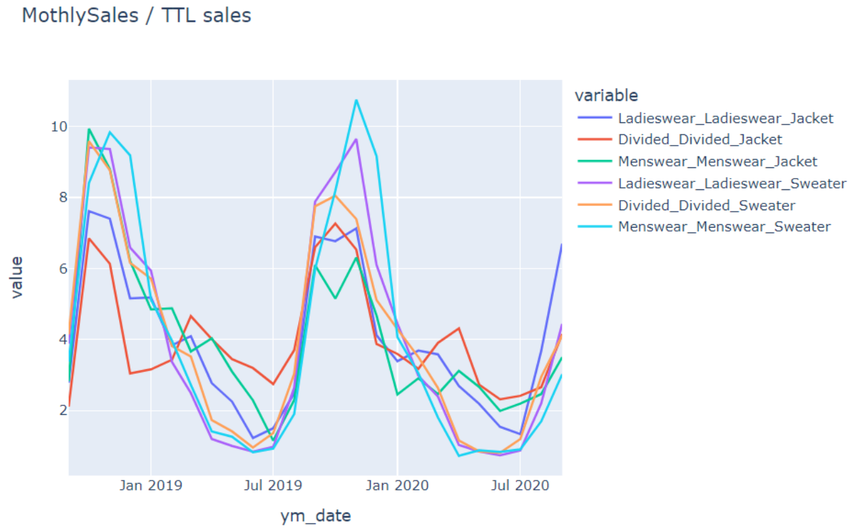
→ contains all h&m articles with details such as a type of product, a color, a product group and other features.

☑ 계절별 물품의 소비량 살펴보기

→ 시기(계절)에 따라 잘 팔리는 물품과 잘 팔리지 않는 물품 有 → **일부 제품의 판매가 계절성이 강하다**
ex) 늦은 9월엔 jacket이 많이 팔림

```
df_article['idxgrp_idx_prdtyp'] = df_article['index_group_name'] + '_' + df_article['index_

df_trans = pd.read_csv('C:/Users/USER/Documents/h-and-m-personalized-fashion-recommendatior
df_trans = df_trans.sample(frac=0.1) # downsampling due to memory limits
df_trans['t_dat'] = pd.to_datetime(df_trans['t_dat'])
df_trans['YYYY_MM'] = df_trans['t_dat'].dt.year.astype(str) + '_' + df_trans['t_dat'].dt.mo
df_trans['year'] = df_trans['t_dat'].dt.year
df_trans['month'] = df_trans['t_dat'].dt.month
df = pd.merge(df_trans, df_article, on='article_id', how='left')
del df_trans, df_article

dfgrp1 = df.groupby(['idxgrp_idx_prdtyp'])[['price']].sum().reset_index()
dfgrp2 = df.groupby(['idxgrp_idx_prdtyp', 'year', 'month'])[['price']].sum().reset_index()
dfgrp2 = pd.merge(dfgrp2, dfgrp1, on='idxgrp_idx_prdtyp', how='left')
dfgrp2['monthsales/ttl-sales'] = dfgrp2['price_x'] / dfgrp2['price_y'] * 100
dfgrp2['ym_date'] = dfgrp2['year'].astype(str) + '-' + dfgrp2['month'].astype(str) + '-1'
dfgrp2['ym_date'] = pd.to_datetime(dfgrp2['ym_date'])
dfgrp2 = pd.pivot_table(dfgrp2, index='ym_date', columns='idxgrp_idx_prdtyp', values='monthl
display(dfgrp2.head())

fig = px.line(dfgrp2, x='ym_date', y=['Ladieswear_Ladieswear_Jacket',
                                      'Divided_Divided_Jacket',
                                      'Menswear_Menswear_Jacket',
                                      'Ladieswear_Ladieswear_Sweater',
                                      'Divided_Divided_Sweater',
                                      'Menswear_Menswear_Sweater'], title="MothlySales / TT

fig.show()
```

→



dfgrp2: 상품을 계절별로 군집화 할 수 있는 피처를 가공한 데이터셋

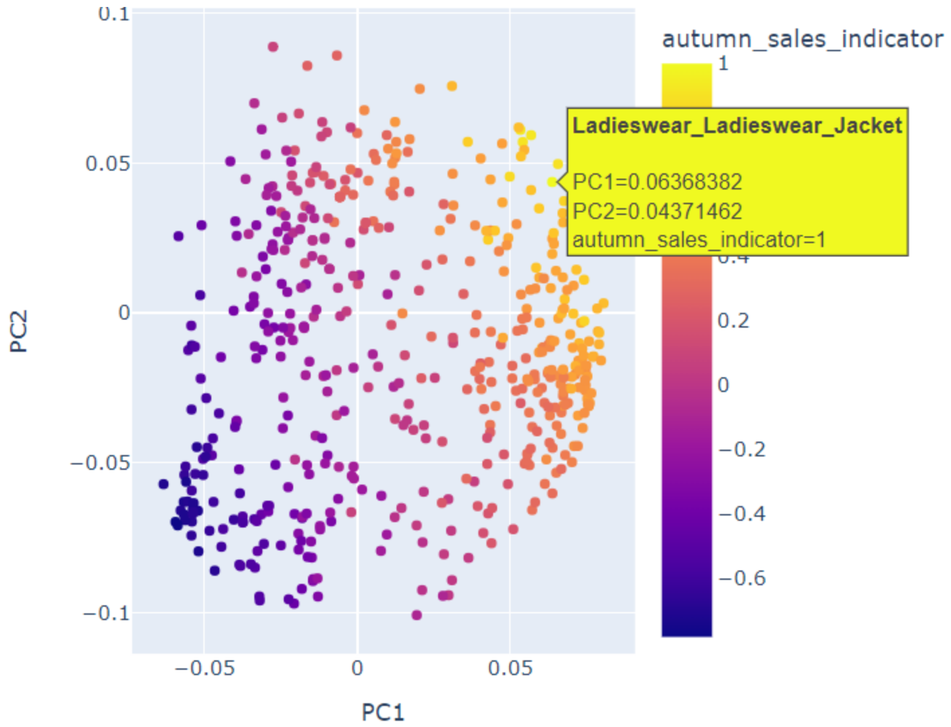
EWHA EURON

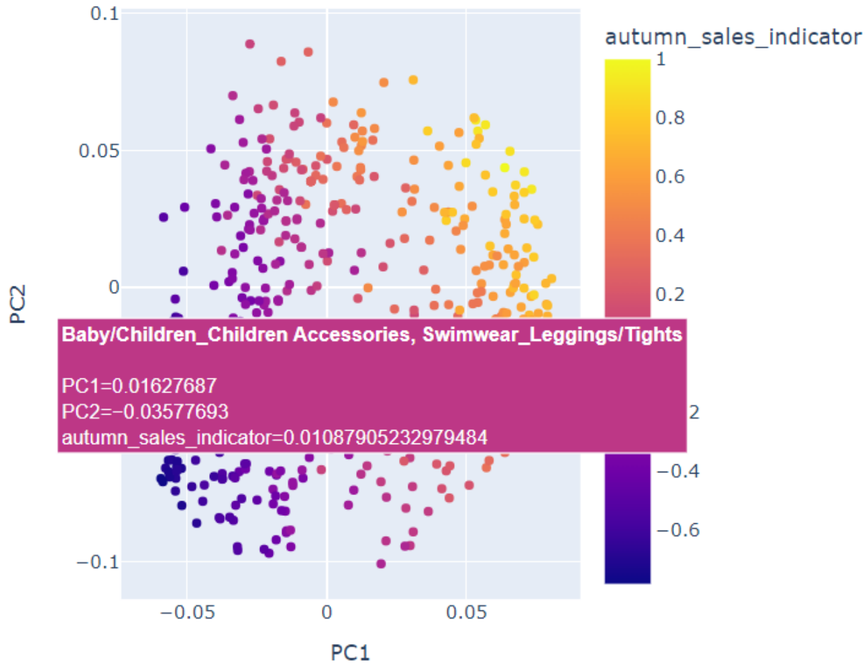
2.2 GMM Clustering – seasonal products 분류

→ 앞서 살펴본 것처럼, 제품 종류별 '판매'에 '계절'의 영향력이 큰 것을 확인

→ 반면, 액세서리와 같은 것들은 계절에 따라 수요가 크게 변하지 않음

⇒ 적절한 clustering을 통해 계절에 따라 제품을 추천해주는 효율적인 모델을 설계할 수 있다!

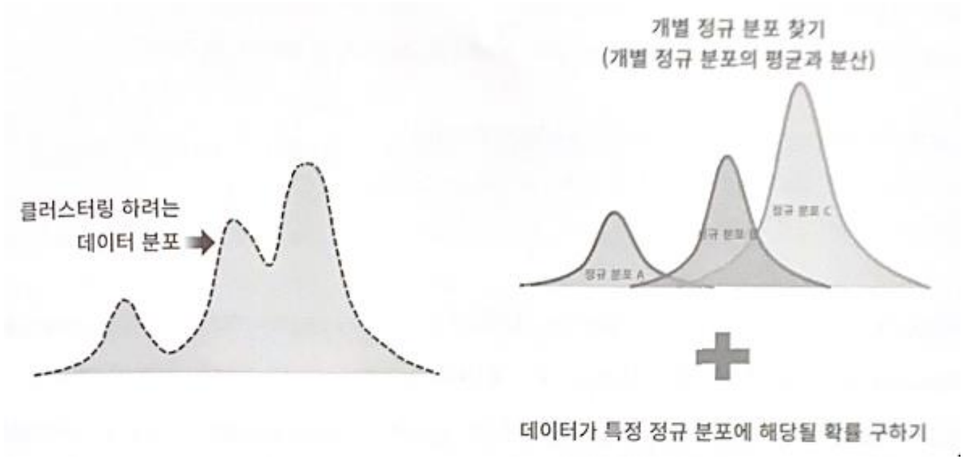




→ Jacket and Sweater는 오른쪽 부분에 위치, Shorts and Swimwear는 좀 더 왼쪽에 위치
(Jacket이 Swimwear보다 autumn_sales_indicator 더 크다)

2.2 GMM Clustering – seasonal products 분류

☑ GMM 복습하기



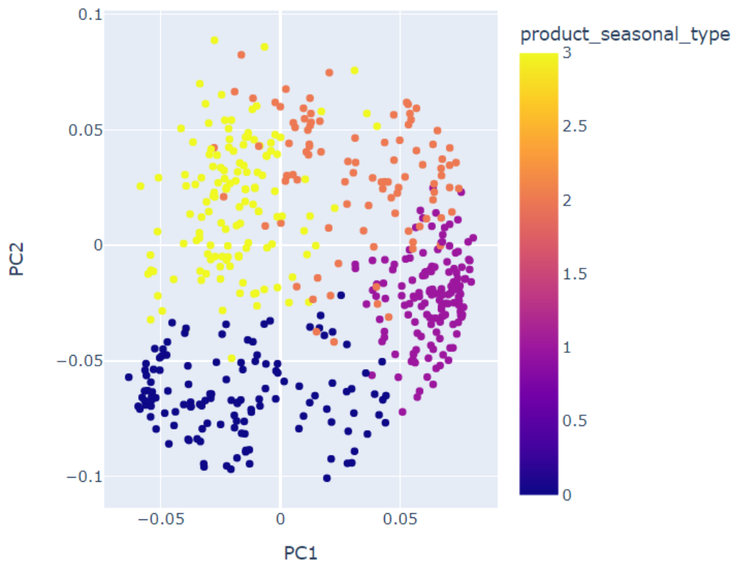
→ 군집화를 적용하고자 하는 데이터가 **여러 개의 가우시안 분포**(GaussianDistribution)를 갖는 데이터 집합들이 섞여 생성된 것이라는 가정하에 군집화를 수행하는 방식

→ GaussianMixture 객체의 가장 중요한 초기화 파라미터 : **n_components**
→ gaussian mixture의 모델의 총 개수(K-Means의 n_clusters와 같이 군집의 개수를 정하는 데 중요한 역할 수행)

2.2 GMM Clustering – seasonal products 분류

```
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=4, covariance_type='full')
gmm.fit(df_eigen[['PC1','PC2','PC3']])
df_eigen['product_seasonal_type'] = gmm.predict(df_eigen[['PC1','PC2','PC3']])
df_eigen['prob_cluster1'] = gmm.predict_proba(df_eigen[['PC1','PC2','PC3']])[:,0]
df_eigen['prob_cluster2'] = gmm.predict_proba(df_eigen[['PC1','PC2','PC3']])[:,1]
df_eigen['prob_cluster3'] = gmm.predict_proba(df_eigen[['PC1','PC2','PC3']])[:,2]
df_eigen['prob_cluster4'] = gmm.predict_proba(df_eigen[['PC1','PC2','PC3']])[:,3]
px.scatter(df_eigen, x='PC1', y='PC2', hover_name='idxgrp_idx_prdtyp', color='product_seasc
```

4계절로 나눌 수 있으므로 n_components=4로 지정



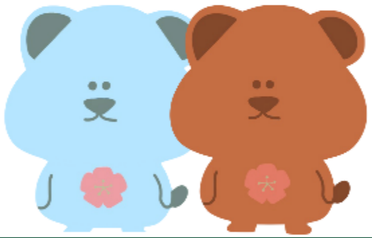
idxgrp_idx_prdtyp	autumn_sales_indicator	product_seasonal_type	prob_cluster1	prob_cluster2	prob_cluster3	prob_cluster4
Ladieswear_Ladieswear_Sweater	0.923106	2	2.962933e-12	1.235655e-02	0.987637	0.000006
Divided_Divided_Top	0.560570	2	4.561374e-14	2.325528e-11	0.996412	0.003588
Ladieswear_Ladieswear_Dress	-0.622045	0	9.654789e-01	3.527333e-24	0.000006	0.034515
Ladieswear_Ladieswear_Trousers	0.075160	0	9.884297e-01	3.496458e-09	0.000441	0.011129
Ladieswear_Ladieswear_Sweater	0.923106	2	2.962933e-12	1.235655e-02	0.987637	0.000006

Jackets과 Sweaters는 class 1 and 2로, Shorts and others는 class 3으로 분류됨.

⇒ Jackets과 같은 경우는 다른 것들보다 특히 계절성을 많이 띈다.
authumn_sales_indicator라는 지표를 통해 가을에 잘 팔리는 제품과 그렇지 않은 제품을 분류할 수 있다.

2. H&M recommendation

- K-means based on **monthly sales** of each article



EWHA EURON

2.3.1 핵심 아이디어/모델링

- K-means 사용
- ‘월별 매출’을 기준으로 제품 군집화

- 각 항목의 가장 빠른 구매 날짜(2018년 9월 20일 이후)
- 각 항목의 최신 구매 날짜(2020년 9월 22일 이전)
- 가장 이른 구매 날짜와 가장 늦은 구매 날짜 사이의 일수로 계산된 판매 기간

2.3.2 Data analysis processing에 따른 코드/결과 분석

① Sales period

```
def plot_sales(article_id, imshow=False):
    plt.figure(figsize=(24, 1.5))
    plot_df = articles.query(f"article_id == '{article_id}'")
    sns.barplot(x=plot_df.columns[29:], y=list(*plot_df.values)[29:], palette=sns.husl_palette(1
2))
    plt.title(" ".join(["Monthly Sales of ID :", article_id, "    earliest :", str(plot_df.iloc
[0, 27])[:10], "    latest :", str(plot_df.iloc[0, 26])[:10]]))
    if imshow:
        show_images(articles.article_id[loc])
```

Hide code

```
temp_df = articles.sort_values([202009, "period"], ascending=False).head(100)
longsellers = temp_df.query("300 < period")
newitems = temp_df.query("period <= 30")
temp_df[["article_id", "product_type_name", "colour_group_name", "period"]].head(30)
```

- Longtime sellers
- New items
- Summer clothing
- Autumn clothing 으로 분류

2.3.2 Data analysis processing에 따른 코드/결과 분석

② K-means

N_cluster=9로 하여 k-means 클러스터링

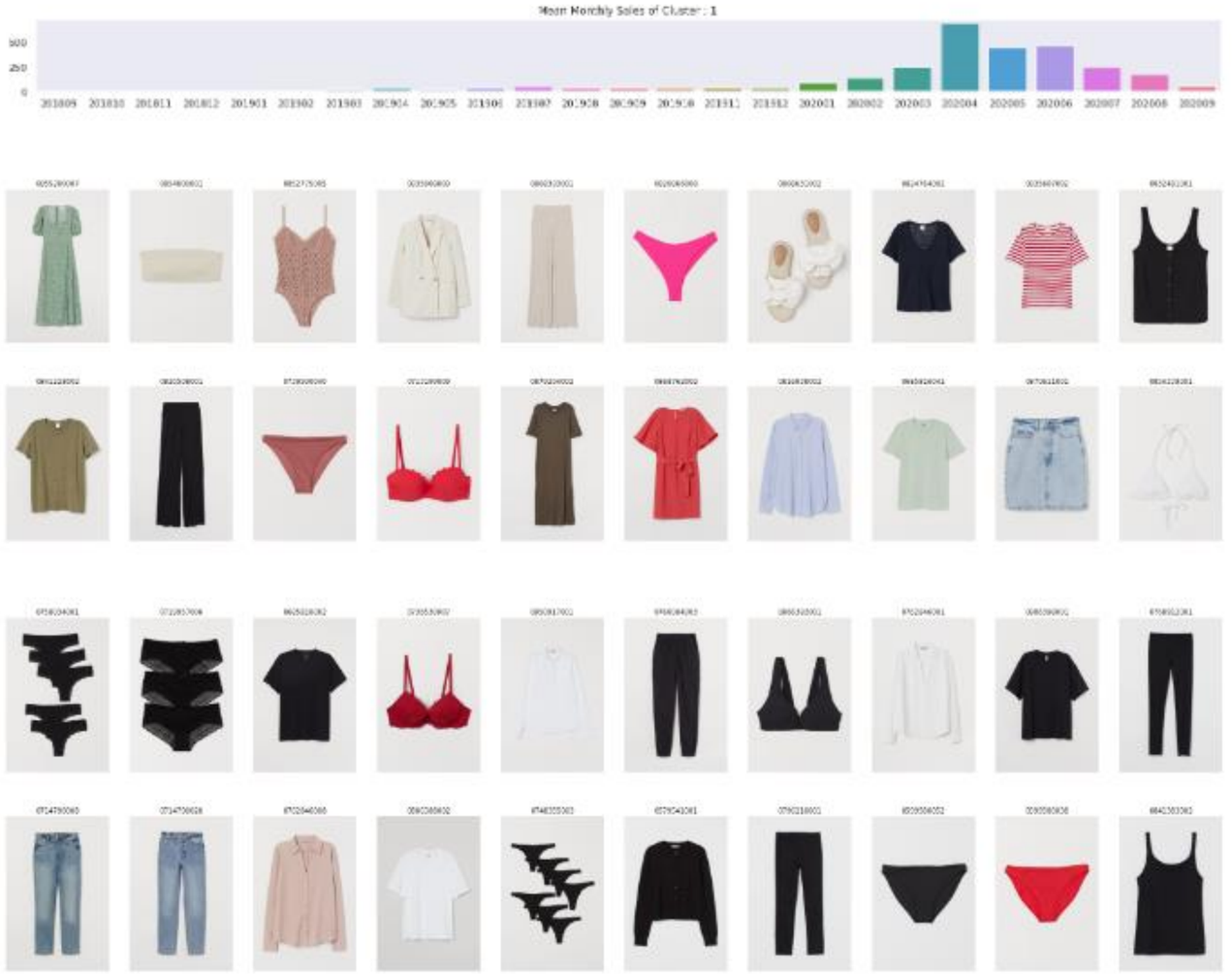
```
articles = articles.sort_values(by="sales_count")

from sklearn import cluster
n_clusters = 9
model = cluster.KMeans(n_clusters=n_clusters)
model.fit(articles.iloc[:,29:]) # consider the period

def plot_cluster(k, n=10):
    temp_df = articles[model.labels_==k]
    plt.figure(figsize=(24, 1.5))
    plot_df = temp_df.iloc[:,29:].describe().loc[["mean"]]
    sns.barplot(x=plot_df.columns, y=list(*plot_df.values), palette=sns.husl_palette(12))
    plt.title(" ".join(["Mean Monthly Sales of Cluster :", str(k)]))
    show_images(list(temp_df.article_id.values[:n]), 10)
    show_images(list(temp_df.article_id.values[-n:]), 10)
    return temp_df.iloc[:,[0] + [i for i in range(29, 54)]]
```

시각화

2.3.2 Data analysis processing에 따른 코드/결과 분석

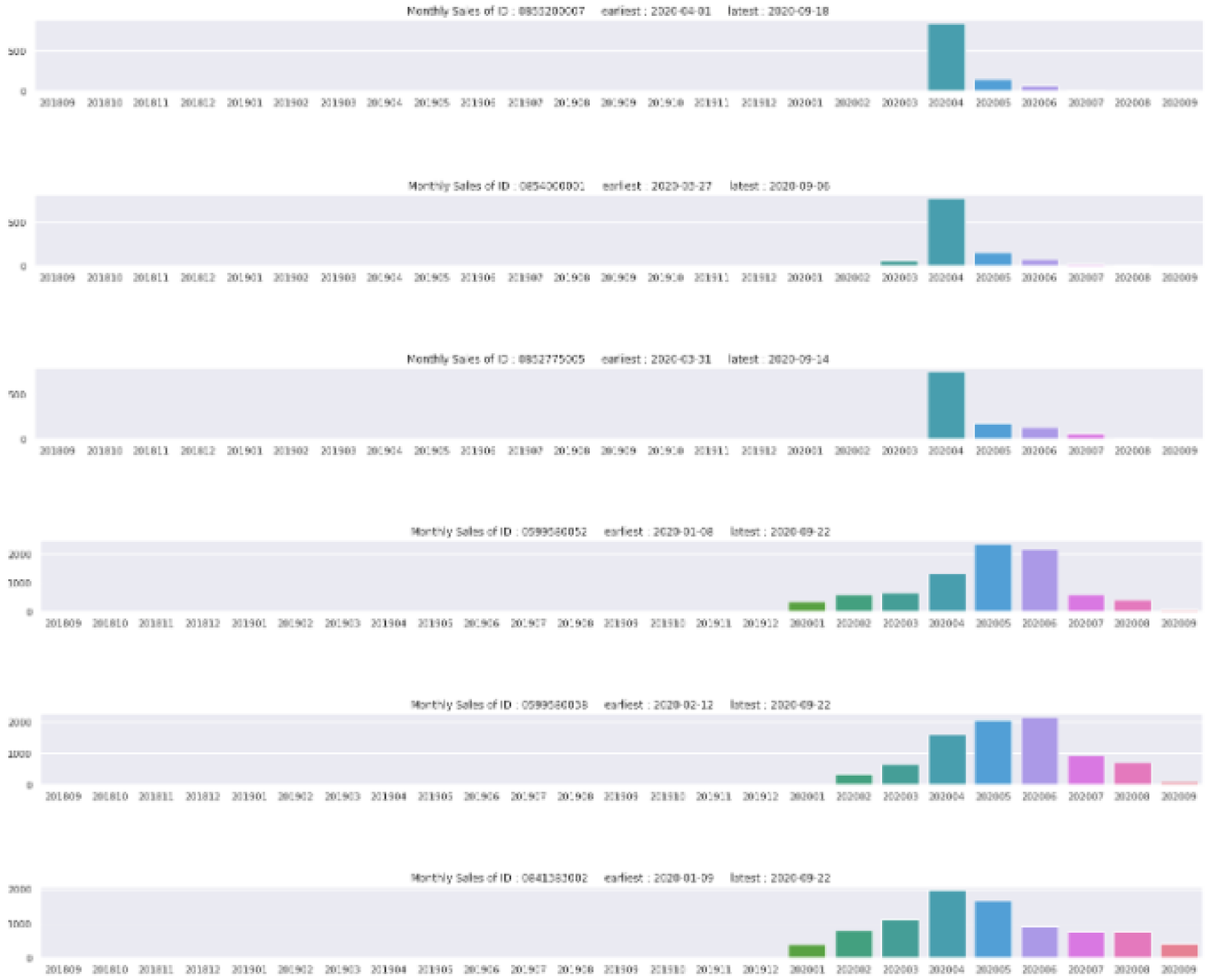
② K-means

```
temp_df = plot_cluster(1, 20)
for ID in list(temp_df.head(3).article_id): plot_sales(ID)
for ID in list(temp_df.tail(3).article_id): plot_sales(ID)
```



2.3.2 Data analysis processing에 따른 코드/결과 분석

② K-means



#3 유방암 예측

EWHA EURON

3.1 대회 소개 및 Data Description

유방 조직의 이미지로 이루어진 위스콘신 유방암 데이터를 이용하여 양성인지 음성인지 분류해내는 것이 목표

- ID number
- Diagnosis (M = malignant, B = benign) → 양성 음성 여부

세포핵으로부터 계산된 열 가지 피처 → 평균, 표준오차, 피처의 worst 값이 각 계산돼 총 30개 피처
- radius : 반경
- texture : gray-scale values 표준편차
- Perimeter : 둘레
- area : 영역
- smoothness : 반경 길이의 국부적 변화
- compactness : perimeter^2 / area - 1.0
- concavity : 윤곽의 오목함
- concave points : 오목한 부분의 개수
- symmetry : 대칭성
- fractal dimension : coastline approximation - 1)

https://www.kaggle.com/code/bhuvanchennoju/women-and-cancer-analysis-and-detection/notebook

3.2 Loading Libraries and Utilities

1. null값 찾고 없애고 타겟값 숫자로 바꾸기

```
print(df.isnull().sum())
df.drop(columns = ['Unnamed: 32'], inplace = True)
df['diagnosis'] = df['diagnosis'].map({'M':1, 'B':0})
```

	0	1
id	842302.000000	842517.000000
diagnosis	1.000000	1.000000
radius_mean	17.990000	20.570000
texture_mean	10.380000	17.770000
perimeter_mean	122.800000	132.900000
area_mean	1001.000000	1326.000000
smoothness_mean	0.118400	0.084740
compactness_mean	0.277600	0.078640
concavity_mean	0.300100	0.086900
concave points_mean	0.147100	0.070170
symmetry_mean	0.241900	0.181200
fractal_dimension_mean	0.078710	0.056670
radius_se	1.095000	0.543500
texture_se	0.905300	0.733900
perimeter_se	8.589000	3.398000
area_se	153.400000	74.080000
smoothness_se	0.006399	0.005225
compactness_se	0.049040	0.013080
concavity_se	0.053730	0.018600
concave points_se	0.015870	0.013400
symmetry_se	0.030030	0.013890
fractal_dimension_se	0.006193	0.003532
radius_worst	25.380000	
texture_worst	17.330000	
perimeter_worst	184.600000	
area_worst	2019.000000	
smoothness_worst	0.162200	
compactness_worst	0.665600	
concavity_worst	0.711900	
concave points_worst	0.265400	
symmetry_worst	0.460100	
fractal_dimension_worst	0.118900	

2. 피처 통계량 확인하기

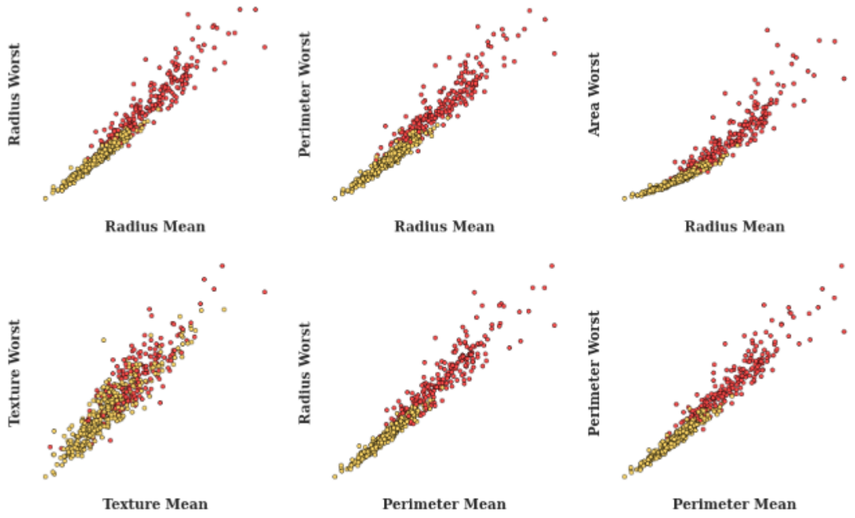
	Features	count	mean	std	min	25%	50%	75%	max
0	DIAGNOSIS	569	0.373	0.484	0.000	0.000	0.000	1.000	1.000
1	RADIUS_MEAN	569	14.127	3.524	6.981	11.700	13.370	15.780	28.110
2	TEXTURE_MEAN	569	19.290	4.301	9.710	16.170	18.840	21.800	39.280
3	PERIMETER_MEAN	569	91.969	24.299	43.790	75.170	86.240	104.100	188.500
4	AREA_MEAN	569	654.889	351.914	143.500	420.300	551.100	782.700	2,501.000
5	SMOOTHNESS_MEAN	569	0.096	0.014	0.053	0.086	0.096	0.105	0.163
6	COMPACTNESS_MEAN	569	0.104	0.053	0.019	0.065	0.093	0.130	0.345
7	CONCAVITY_MEAN	569	0.089	0.080	0.000	0.030	0.062	0.131	0.427
8	CONCAVE POINTS_MEAN	569	0.049	0.039	0.000	0.020	0.034	0.074	0.201
9	SYMMETRY_MEAN	569	0.181	0.027	0.106	0.162	0.179	0.196	0.304
10	FRACTAL_DIMENSION_MEAN	569	0.063	0.007	0.050	0.058	0.062	0.066	0.097
11	RADIUS_SE	569	0.405	0.277	0.112	0.232	0.324	0.479	2.873
12	TEXTURE_SE	569	1.217	0.552	0.360	0.834	1.108	1.474	4.885
13	PERIMETER_SE	569	2.866	2.022	0.757	1.606	2.287	3.357	21.980
14	AREA_SE	569	40.337	45.491	6.802	17.850	24.530	45.190	542.200
15	SMOOTHNESS_SE	569	0.007	0.003	0.002	0.005	0.006	0.008	0.031
16	COMPACTNESS_SE	569	0.025	0.018	0.002	0.013	0.020	0.032	0.135
17	CONCAVITY_SE	569	0.032	0.030	0.000	0.015	0.026	0.042	0.396
18	CONCAVE POINTS_SE	569	0.012	0.006	0.000	0.008	0.011	0.015	0.053
19	SYMMETRY_SE	569	0.021	0.008	0.008	0.015	0.019	0.023	0.079
20	FRACTAL_DIMENSION_SE	569	0.004	0.003	0.001	0.002	0.003	0.005	0.030
21	RADIUS_WORST	569	16.269	4.833	7.930	13.010	14.970	18.790	36.040
22	TEXTURE_WORST	569	25.677	6.146	12.020	21.080	25.410	29.720	49.540
23	PERIMETER_WORST	569	107.261	33.603	50.410	84.110	97.660	125.400	251.200
24	AREA_WORST	569	880.583	569.357	185.200	515.300	686.500	1,084.000	4,254.000
25	SMOOTHNESS_WORST	569	0.132	0.023	0.071	0.117	0.131	0.146	0.223
26	COMPACTNESS_WORST	569	0.254	0.157	0.027	0.147	0.212	0.339	1.058
27	CONCAVITY_WORST	569	0.272	0.209	0.000	0.115	0.227	0.383	1.252
28	CONCAVE POINTS_WORST	569	0.115	0.066	0.000	0.065	0.100	0.161	0.291
29	SYMMETRY_WORST	569	0.290	0.062	0.157	0.250	0.282	0.318	0.664
30	FRACTAL_DIMENSION_WORST	569	0.084	0.018	0.055	0.071	0.080	0.092	0.207

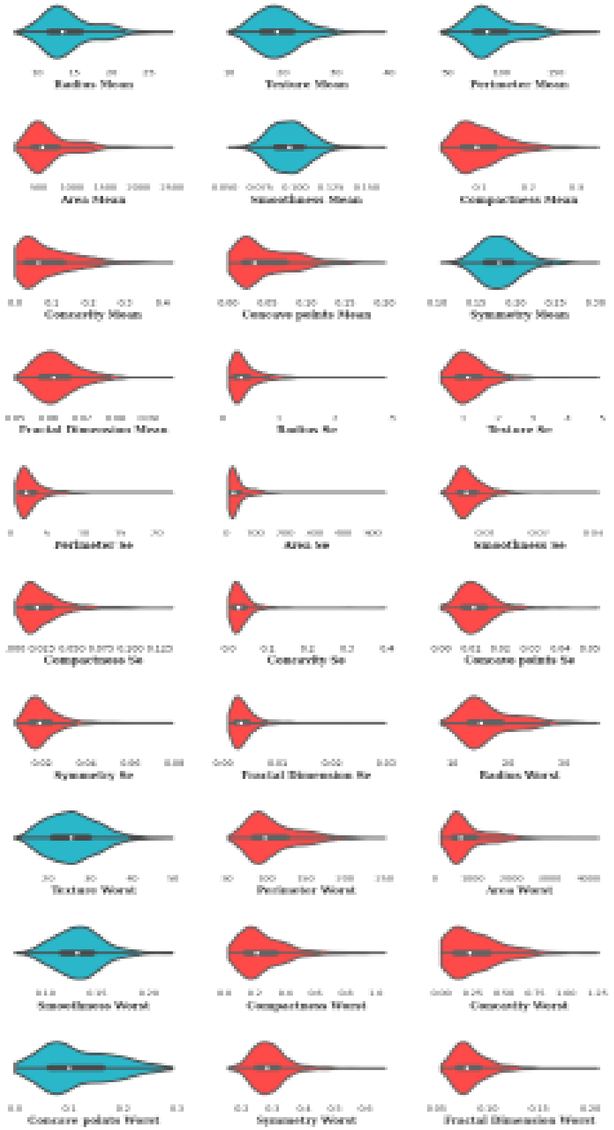
3.3 Data Exploration and Explinatory Analysis

여러 방법을 통해 데이터의 분포 및 특징 확인

1. Target Distribution
  : 약 40%의 여성이 암에 취약한 상태
2. Univariate Analysis / Binary Feature Analysis/ Multivariate Analysis
  : 모델링을 위해 “ 왜도>1 ” 조건에 따라 피처 분리한 결과

- 대부분의 피처가 치우쳐진 분포를 보임 -> 스케일링 필요
- 대부분의 피처는 비슷한 분포를 보임 (일부는 정규분포와 비슷함)
- 많은 피처들끼리 강한 상관관계를 보임 -> 다중공선성 조심



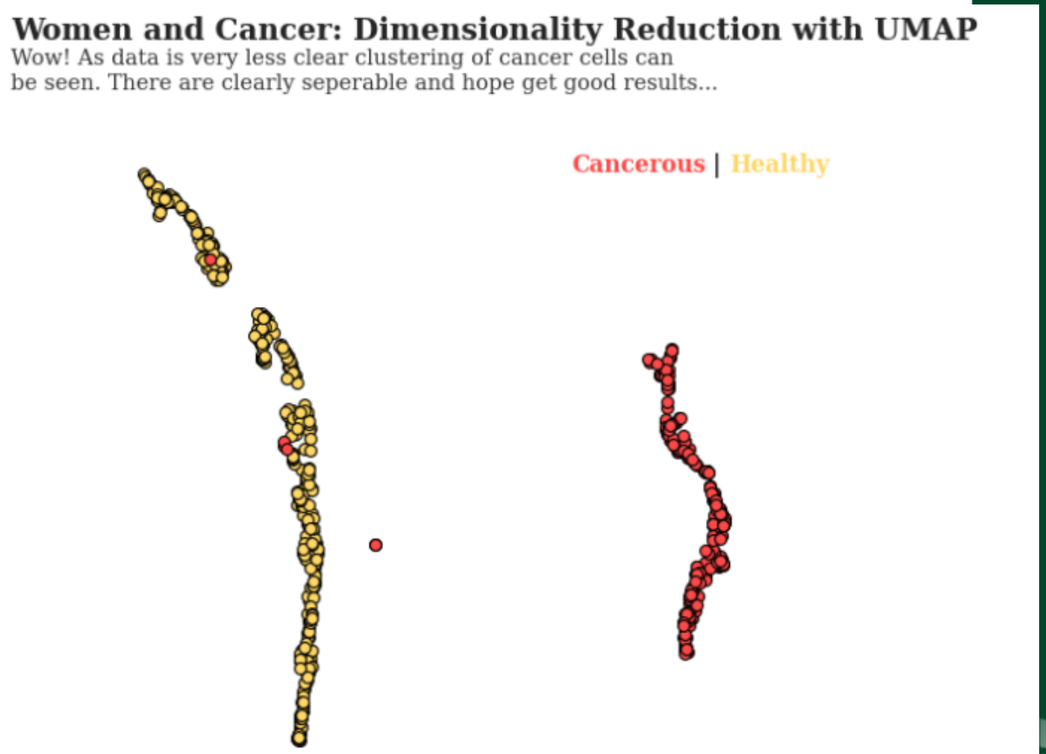


3.3 Data Exploration and Explanatory Analysis

3. Class Segregation with Dimensionality Reduction

```
temp = df.copy()

X_temp = temp.drop(columns = ['id','diagnosis'])
y_temp = temp['diagnosis']

# fitting on umap
umap = UMAP(random_state=2021)
model_umap = umap.fit_transform(X_temp, y_temp)

fig,ax = plt.subplots(figsize=(7,7),dpi =80)

# plots
ax.scatter(model_umap[temp['diagnosis'] == 0][:,0], model_umap[temp['diagnosis'] == 0][:,1], c= col
ax.scatter(model_umap[temp['diagnosis'] == 1][:,0], model_umap[temp['diagnosis'] == 1][:,1], c= col


## titles and text

ax.set_xticklabels('')
ax.set_yticklabels('')

fig.text(0,1.01,'Women and Cancer: Dimensionality Reduction with UMAP', {'font':'serif','size':18,
fig.text(0,0.95,'''Wow! As data is very less clear clustering of cancer cells can
be seen. There are clearly seperable and hope get good results...''',{'font':'serif','size':13, 'wei

fig.text(0.68,0.85, "Cancerous",{'font':'serif','size':14, 'weight':'bold', 'color':colors[0]})
fig.text(0.85,0.85, '|',{'font':'serif','size':14, 'weight':'bold'})
fig.text(0.87,0.85, "Healthy",{'font':'serif','size':14, 'weight':'bold','color':colors[2]})

fig.text(0.65,0.05,'© Made by bhuvanchennoju/Kaggle',{'font':'serif', 'size':10,'weight':'bold'}, al


fig.show()
```

차원축소 방법인 UMAP을 이용하면
암세포의 두 상태가 명확하게 분리됨



3.4 Data Cleaning Techniques And Feature Engineering

다변수 피처 이상치 처리 방법

1. Elliptic Envelope 2. DBSCAN 3. Local Outlier Factor 4. Isolation Forest

DBSCAN(Density Based Spatial Clustering of Applications with Noise)

: 데이터의 분포가 기하학적으로 복잡한 데이터 세트에서도 효과적인 결과를 보이는 밀도 기반 군집화 방법

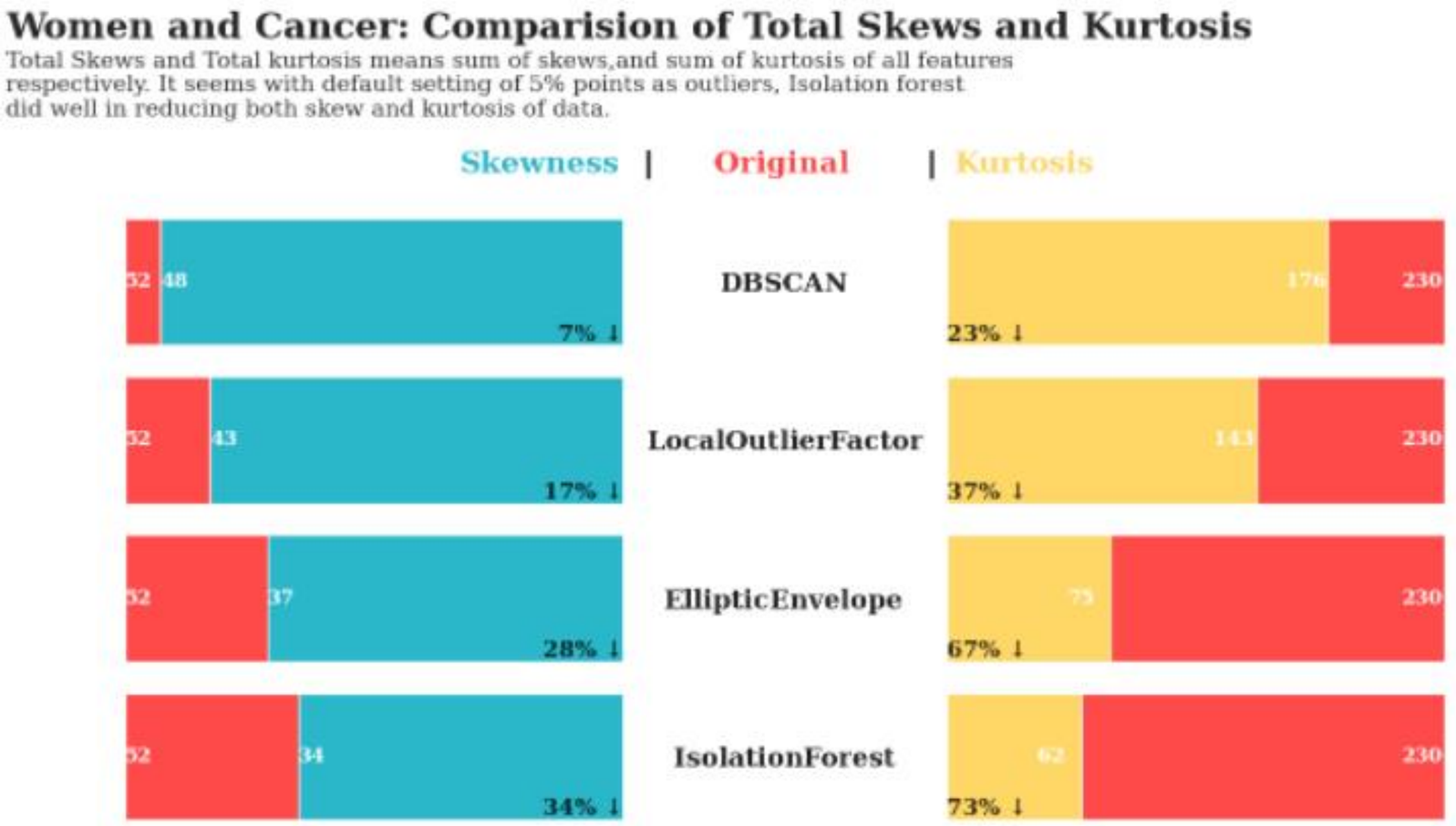
=> DBSCAN에서 계산한 noise point을 이상치로 간주하여 제거하는 방식 제안

```python
def outlier_detect(algo,data):
    cols = data.drop(columns = ['id']).columns
    # creating feature and target numpy arrays
    feat, tar = data[cols].drop(columns = 'diagnosis').values, data['diagnosis'].values
    # fitting the features to algo
    yhat = algo.fit_predict(feat)
    # masking the features that are not outliers
    mask = yhat != -1
    X,y = feat[mask,:], tar[mask]
    data_inarray = np.append(y.reshape(-1,1),X,axis = 1)
    return pd.DataFrame(data = data_inarray, columns = cols)

def skew_sum(data):
    return skew(data).sum()

def kurtosis_sum(data):
    return kurtosis(data).sum()

def shape(data):
    return data.shape

outlier_algos = [IsolationForest(contamination = 0.05),#
                 EllipticEnvelope(contamination = 0.05),#
                 LocalOutlierFactor(contamination = 0.05), #
                 DBSCAN(eps = 70, min_samples = 10)]
```

```python
algorithms = ['Original','IsolationForest', 'EllipticEnvelope', 'LocalOutlierFactor', 'DBSCAN']
outliers_info = pd.DataFrame({'algorithms':algorithms,'df_list':df_list,'shapes':shapes, 'skews':skews, 'kurts': kurts})

outliers_info['skews_sum'] = outliers_info['skews'].apply(lambda x: round(x.sum(),2))
outliers_info['kurts_sum'] = outliers_info['kurts'].apply(lambda x: round(x.sum(),2))
outliers_info.sort_values(by = 'shapes').reset_index(drop = True, inplace = True)


for idx, df_ in enumerate(outliers_info['df_list']):
    from sklearn.metrics import f1_score

    lr = LinearRegression()
    X = df_.drop(columns = ['diagnosis'])
    y = df_['diagnosis']
    xtrain, xtest,ytrain,ytest = train_test_split(X,y,test_size = 0.2)

    # linear regression
    preds = LinearRegression().fit(xtrain.values,ytrain.values).predict(xtest.values)

    r2 = round(r2_score(ytest,preds),3)
    #ypred_class = preds > 0.85
    #acc = round(accuracy_score(ytest, ypred_class),3)
    #roc_auc = round(roc_auc_score(ytest,ypred_class),3)

    metric_list = r2

    outliers_info.loc[idx, 'r2_score'] = metric_list
```

3.4 Data Cleaning Techniques And Feature Engineering

가장 결과가 좋은 IsolationForest 선택
그 결과 (569,31) -> (540,31)로 outlier 제거됨



** IsolationForest
: 데이터셋을 의사결정나무 형태로 표현하여 이상치는 나무 상단에서 분리하고 정상값은 나무를 깊게 타고 내려가야 분리할 수 있는 특성을 이용해 내려간 횟수를 기준으로 분리하는 밀도기반 탐지법

3.4 Data Cleaning Techniques And Feature Engineering

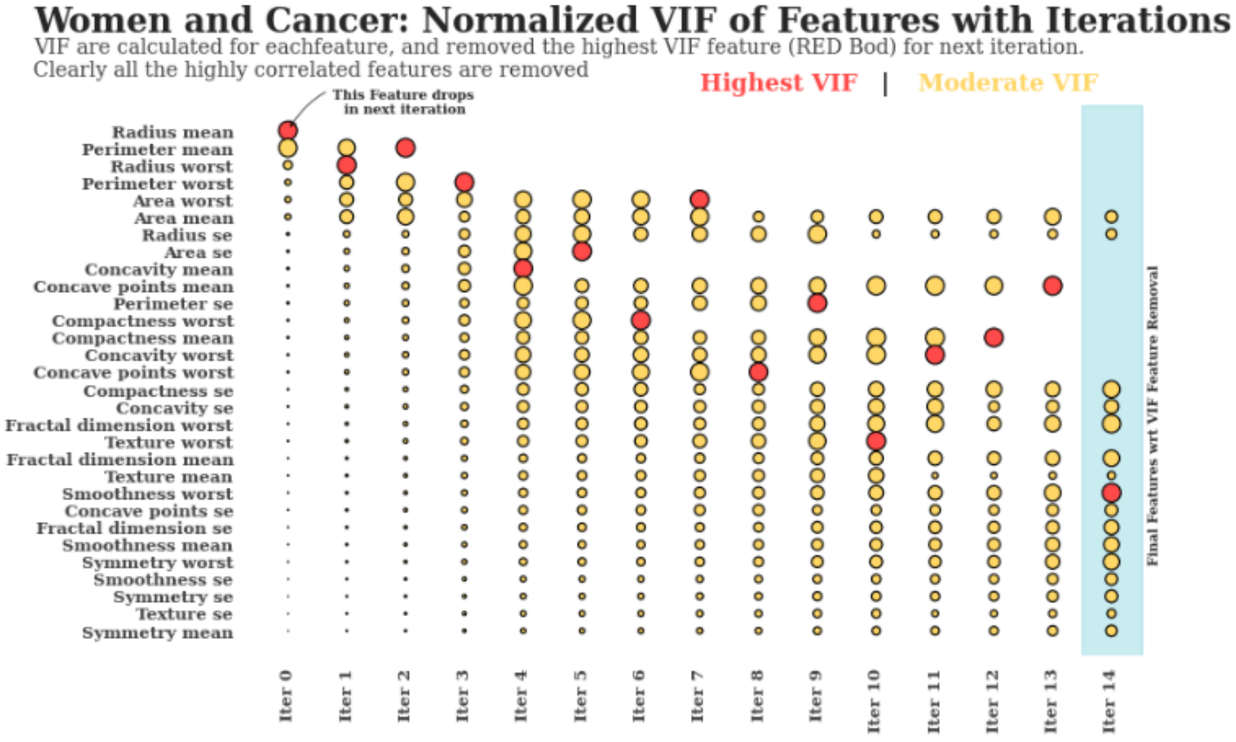
다중공선성 해결 방법

1. VIF 2. 순열 피처 중요도/ 피처 선택 3. 라쏘나 릿지

```
def VIF(data):
    vif_list = list()
    for col in data.columns:
        X = data.drop(columns = [col])
        y = data[col]
        model = LinearRegression().fit(X.values,y.values)
        ypreds = model.predict(X.values)
        r2 = r2_score(y.values,ypreds)
        VIF = 1 /(1-r2)
        vif_list.append(VIF)
    return vif_list
```

```
Variance Inflation factor implementation for feature selection....


Calcuations are finished! feature and corresponding VIF are stored

Final Features In Data and Final VIFs...
```

	Features	VIF
0	smoothness_worst	9.321104
1	fractal_dimension_worst	8.658489
2	compactness_se	7.597517
3	fractal dimension mean	7.068115
4		
5		
6	fr	
7		
8		
9		
10		
11		
12		
13		
14		
15		



3.4 Data Cleaning Techniques And Feature Engineering

다중공선성 해결 방법

1. VIF 2. 순열 피처 중요도/피처 선택 3. 라쏘나 릿지

```
X_temp = df.drop(columns = ['diagnosis'])
y_temp = df['diagnosis']
temp_X_train,temp_X_val,temp_y_train,temp_y_val = train_test_split(X_tem
p,y_temp, test_size = 0.2, random_state = 2021)

temp_model= XGBClassifier(eval_metric='logloss').fit(temp_X_train,temp_y.
train)

perm = PermutationImportance(temp_model, scoring = 'r2').fit(temp_X_val,
emp_y_val)
eli5_feature_importance1 = (pd.DataFrame({'Features':temp_X_train.column
s.tolist(),'Importance':perm.feature_importances_})
                            .sort_values(by = 'Importance'))
perm_imp_feats1 = (eli5_feature_importance1.sort_values(by = 'Importanc
e', ascending = False)
                          .reset_index(drop = True))['Features'][0:15]
```

Weight	Feature
0.1271 ± 0.0594	area_worst
0.1032 ± 0.0389	concave points_worst
0.0874 ± 0.0778	texture_worst
0.0794 ± 0.0502	concavity_worst
0.0635 ± 0.0810	area_se
0.0635 ± 0.0635	compactness_se
0.0397 ± 0.0502	texture_mean
0.0397 ± 0.0000	smoothness_worst
0.0238 ± 0.0389	symmetry_mean
0.0159 ± 0.0389	symmetry_worst
0.0159 ± 0.0389	smoothness_mean
0.0079 ± 0.0594	concave points_mean
0 ± 0.0000	radius_mean
0 ± 0.0000	perimeter_mean
0 ± 0.0000	compactness_mean
0 ± 0.0000	concavity_mean
0 ± 0.0000	fractal_dimension_mean
0 ± 0.0000	radius_se
0 ± 0.0000	area_mean
0 ± 0.0000	fractal_dimension_worst
… *10 more* …	

Permutation Feature Importance(PFI)

: 기본 feature importance 의 부정적 영향을 주는 피처를 알 수 없다는 단점을 보완한 방법

3.4 Data Cleaning Techniques And Feature Engineering

다중공선성 해결 방법

1. VIF 2. 피처 중요도/피처 선택 3. 라쏘나 릿지

: PFI, VIF, 원래 피처를 이용한 선형 회귀 비교

```
Accuracy score, r2 score, and roc_auc score:

Orginal Features: [0.824, 0.721, 0.973]

Permutation Feature Selection: [0.815, 0.726, 0.992]

Variance Infaltion Factor based Feature Selection: [0.796, 0.726, 0.988]
```

```
def test_linear_features():
        X_orig = df.drop(columns = ['diagnosis'])
        y_orig = df['diagnosis']
        X_train,X_val,y_train,y_val = train_test_split(X_orig,y_orig, test_size = 0.2, random_state = 2021)

        # linear regression
        orig_preds = LinearRegression().fit(X_train.values,y_train.values).predict(X_val.values)
        vif_preds = LinearRegression().fit(X_train[vif_features].values,y_train.values).predict(X_val[vif_features].values)
                                                                                                t(X_train[perm_imp_feats1].values,y_train.values).predict(X_val[perm_imp_feats1].va
                                                                                                ig_preds),3)
                                                                                                _preds),3)
                                                                                                rm_preds),3)
                                                                                                re
        orig_ypred_class = orig_preds > 0.85
        vif_ypred_class = vif_preds > 0.85
        perm_ypred_class = perm_preds > 0.85

        orig_auc = round(accuracy_score(y_val, orig_ypred_class),3)
        vif_auc = round(accuracy_score(y_val, vif_ypred_class),3)
        perm_auc = round(accuracy_score(y_val, perm_ypred_class),3)

        orig_roc_auc = round(roc_auc_score(y_val, orig_preds),3)
        vif_roc_auc = round(roc_auc_score(y_val, vif_preds),3)
        perm_roc_auc = round(roc_auc_score(y_val, perm_preds),3)


        orig_list = [orig_auc,orig_r2,orig_roc_auc]
        vif_list = [vif_auc, vif_r2,vif_roc_auc]
        perm_list = [perm_auc,perm_r2,perm_roc_auc]
        return orig_list,vif_list,perm_list
```

3.4 Data Cleaning Techniques And Feature Engineering

다중공선성 해결 방법

1. VIF 2. 피처 중요도/피처 선택 3. 라쏘나 릿지

: 결과를 PCA를 통해 차원 축소 후 시각화

```
class pca_viz():

    def __init__(self,feat,tar,ax):
        self.feat = feat
        self.tar = tar
        self.ax = ax

    def visualize_data(self):

        temp_y = pd.DataFrame({'y':self.tar})

        pca = PCA(n_components= 2).fit_transform(self.feat)
        # plotting
        self.ax.scatter(pca[temp_y['y'] == 0][:,0], pca[temp_y['y'] == 0]
[:,1], c = colors[2], s = 50, linewidth =1, ec = 'black')
        self.ax.scatter(pca[temp_y['y'] == 1][:,0], pca[temp_y['y'] == 1]
[:,1], c = colors[0], s = 50, linewidth =1,ec ='black')
```

3.4 Data Cleaning Techniques And Feature Engineering

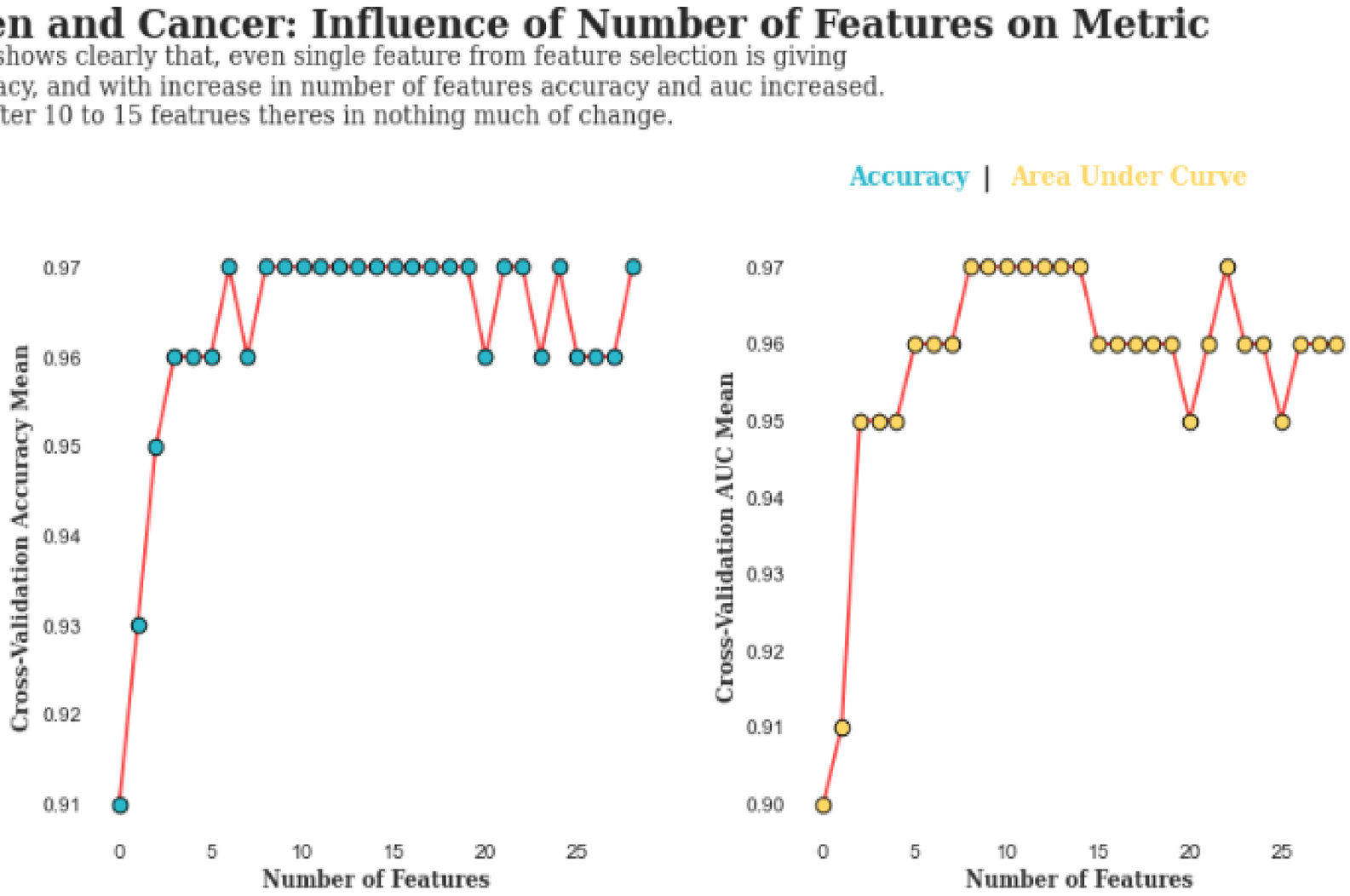
roc_auc metric을 이용한 feature importance

```
X_temp = df.drop(columns = ['diagnosis'])
y_temp = df['diagnosis']
temp_X_train,temp_X_val,temp_y_train,temp_y_val = train_test_split(X_temp,y_temp, test_size = 0.2, random_state = 26

temp_model= XGBClassifier(eval_metric='logloss').fit(temp_X_train,temp_y_train)

perm = PermutationImportance(temp_model, scoring = 'roc_auc').fit(temp_X_val,temp_y_val)
eli5_feature_importance2 = (pd.DataFrame({'Features':temp_X_train.columns.tolist(),'Importance':perm.feature_importa
                            .sort_values(by = 'Importance'))
perm_imp_feats_auc = (eli5_feature_importance2.sort_values(by = 'Importance', ascending = False)
                        .reset_index(drop = True))['Features']

print(color_class.BOLD_COLOR + 'Feature Importance with roc_auc metric....')
eli5.show_weights(perm, feature_names = temp_X_val.columns.tolist())
```

Weight	Feature
0.0135 ± 0.0091	texture_worst
0.0103 ± 0.0164	area_se
0.0082 ± 0.0065	concave points_worst
0.0081 ± 0.0093	area_worst
0.0023 ± 0.0067	concavity_worst
0.0021 ± 0.0009	texture_mean
0.0018 ± 0.0015	compactness_se
0.0014 ± 0.0025	concave points_mean
0.0010 ± 0.0023	symmetry_worst
0.0007 ± 0.0013	symmetry_mean
0.0006 ± 0.0004	symmetry_se
0.0004 ± 0.0009	concavity_se
0 ± 0.0000	fractal_dimension_mean
0 ± 0.0000	perimeter_mean
0 ± 0.0000	fractal_dimension_worst
0 ± 0.0000	perimeter_se
0 ± 0.0000	radius_mean
0 ± 0.0000	smoothness_se
0 ± 0.0000	area_mean
-0.0001 ± 0.0007	compactness_mean
… *10 more* …	

3.4 Data Cleaning Techniques And Feature Engineering

효과적인 importance feature 개수 파악
: 10~15개 정도가 적당



3.5 Modeling

Scaling

1. StandardScaler 2. MinMaxScaler 3. RobustScaler

```
def plot_feat(axes_idx = None, data_ = None, scaler_method = None, color = None):

    col_names = data_.columns

    scaled_array = scaler_method.fit_transform( data_)

    # scaled dataframe
    scaled_df = pd.DataFrame(scaled_array, columns=col_names)

    skew_scaler = []
    for idx, col in zip(axes_idx, col_names):
        col_skew = skew(scaled_df[col])
        if col_skew > 1.5:
            temp = np.log1p(scaled_df[col] + 0.5)
        else:
            temp = scaled_df[col]

        ## plot
        sns.kdeplot(x = temp, ax = axes[idx],
                    color = color,fill = True, alpha = 1,
                    linewidth = 3,ec = 'black')


        skew_scaler.append(col_skew)

        xlabel = ' '.join([value.capitalize() for value in str(col).split('_') ])
        #ax.set_facecolor(colors[-1])
        axes[idx].axes.get_yaxis().set_visible(False)
        axes[idx].axes.set_xlabel(xlabel,{'font':'serif','size':14, 'weight':'bold'}, alpha = 1)

    return skew_scaler
```

```
scaler_skews = []
for axes_idx_list,scaler, color in zip(axes_np_list,scaler_list, colors_list):

    skewness = plot_feat(axes_idx = axes_idx_list, data_ = data, scaler_method = scaler,color=color)
    scaler_skews.append(skewness)
```

이미 outlier를 제거해 세 방법 모두 비슷한 정도이나
정규분포와 비슷한 결과들을 낸 StandardScaler 선택

3.5 Modeling

총 9개의 분류 모델에 대해 cross-validation score 측정 후 top 3개 bottom 2개 선택

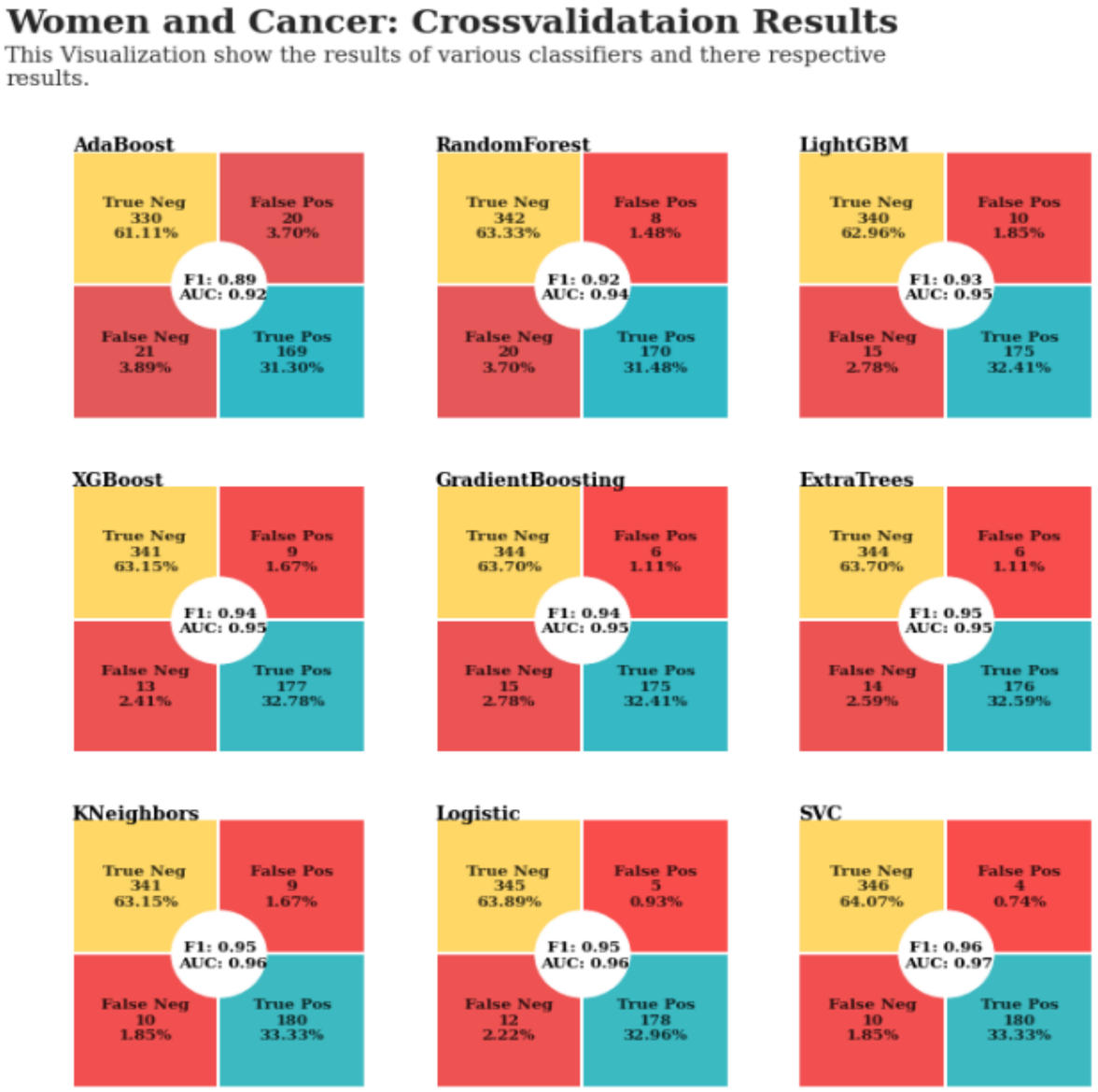
```
stratified = StratifiedKFold(n_splits = 5, shuffle = True, random_state  = 20)
Algorithms = ["Logistic","SVC","KNeighbors","AdaBoost",
              "RandomForest","GradientBoosting",
              "ExtraTrees","XGBoost", "LightGBM"]
```

```
for classifier,algo in zip(classifiers,Algorithms):

    fold_accuracy = []
    fold_f1 = []
    fold_roc_auc = []
    fold_preds = []
    fold_class_states = []
    fold_valid_truths = []
    fold_valid_features = []
    fold_cm = []

    n = 0

    print(color_class.BOLD + '*'*17+  color_class.END + color_class.BOLD_COLOR + str(algo) + color_class.END + co
    for train_idx,valid_idx in stratified.split(xdata,ydata):
        xtrain,xvalid = xdata.iloc[train_idx],xdata.iloc[valid_idx]
        ytrain,yvalid = ydata.iloc[train_idx],ydata.iloc[valid_idx]

        ## scaling
        ss = StandardScaler()
        xtrain = ss.fit_transform(xtrain)
        xvalid = ss.transform(xvalid)
```

```
        # model
        model = classifier
        model.fit(xtrain,ytrain)
        preds = model.predict(xvalid)

        ## scores
        #### fold results, feaures,preds, states
        accuracy = accuracy_score(yvalid, preds)
        f1 = f1_score(yvalid,preds)
        roc_auc = roc_auc_score(yvalid,preds)
        cm = confusion_matrix(yvalid,preds)

        fold_accuracy.append(accuracy)
        fold_f1.append(f1)
        fold_roc_auc.append(roc_auc)
        fold_preds.append(preds)
        fold_class_states.append(model)
        fold_valid_truths.append(np.array(yvalid).astype(int));
        fold_valid_features.append(xvalid)
        fold_cm.append(cm)
```

EWHA EURON

3.5 Modeling



F1 score, AUC, 혼동행렬 등의 지표들을 통해 선택된 분류 모델

- Logistic
- LightGBM
- XGBoost

- RandomForest
- AdaBoost

3.5 Modeling

하이퍼파라미터 튜닝

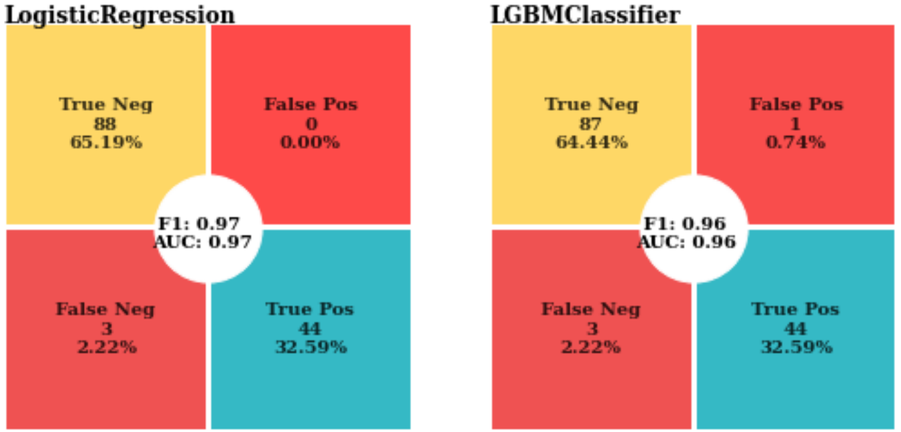
```
classifiers_params = {

    LogisticRegression(): {'C':[0.001,0.01,0.05,0.1,0.5,1,10,100,200,1000]
                'penalty': ['l1','l2']} ,

    LGBMClassifier():     {
                'class_weight': [{1:6,0:4},{1:7,0:3},{1:8,0:4}],
                'n_estimators': np.arange(100,3000,250),
                'num_leaves': np.arange(10,50,10),
                'learning_rate': [0.01,0.05,0.1,0.5]},

    RandomForestClassifier() :     {
                 'class_weight': [{1:6,0:4}],
                 'max_depth': [2,4,6,8,10],
                 'max_leaf_nodes': [5,10,15],
                 'n_estimators': np.arange(100,2000,500)} ,
    AdaBoostClassifier() :    {
                 'base_estimator': [DecisionTreeClassifier()],
                 'learning_rate': [0.01,0.05,0.1],
                 'n_estimators': np.arange(100,1000,500)} ,
}
```

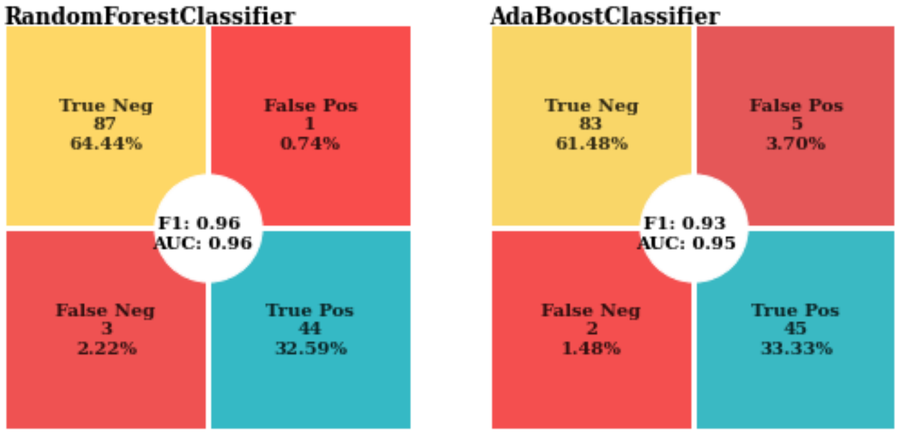
```
ss = StandardScaler()

xtrain = ss.fit_transform(xtrain)
xtest = ss.transform(xtest)

best_est = []
best_pms = []

for clf,params in classifiers_params.items():
    print(color_class.BOLD_COLOR + '*'*20 + color_class.END +'₩n')

    gs = GridSearchCV(estimator= clf, param_grid = params,cv = stratified, verbose= 2,scoring = 'roc_auc',n_j
    #gs = clf
    gs.fit(xtrain,ytrain)
    best_estimator = gs.best_estimator_
    best_params = gs.best_params_
    best_est.append(best_estimator)
    best_pms.append(best_params)
    preds = best_estimator.predict(xtest)

    print(color_class.BOLD)
    print(best_estimator)

    print('Accuracy: {}'.format(accuracy_score(ytest,preds)))
    print('Roc_Auc: {}'.format(round(roc_auc_score(ytest,preds),8)))
```

3.5 Modeling

Women and Cancer: GridSearch Results

This Visualization show the results of various classifiers and there respective results.





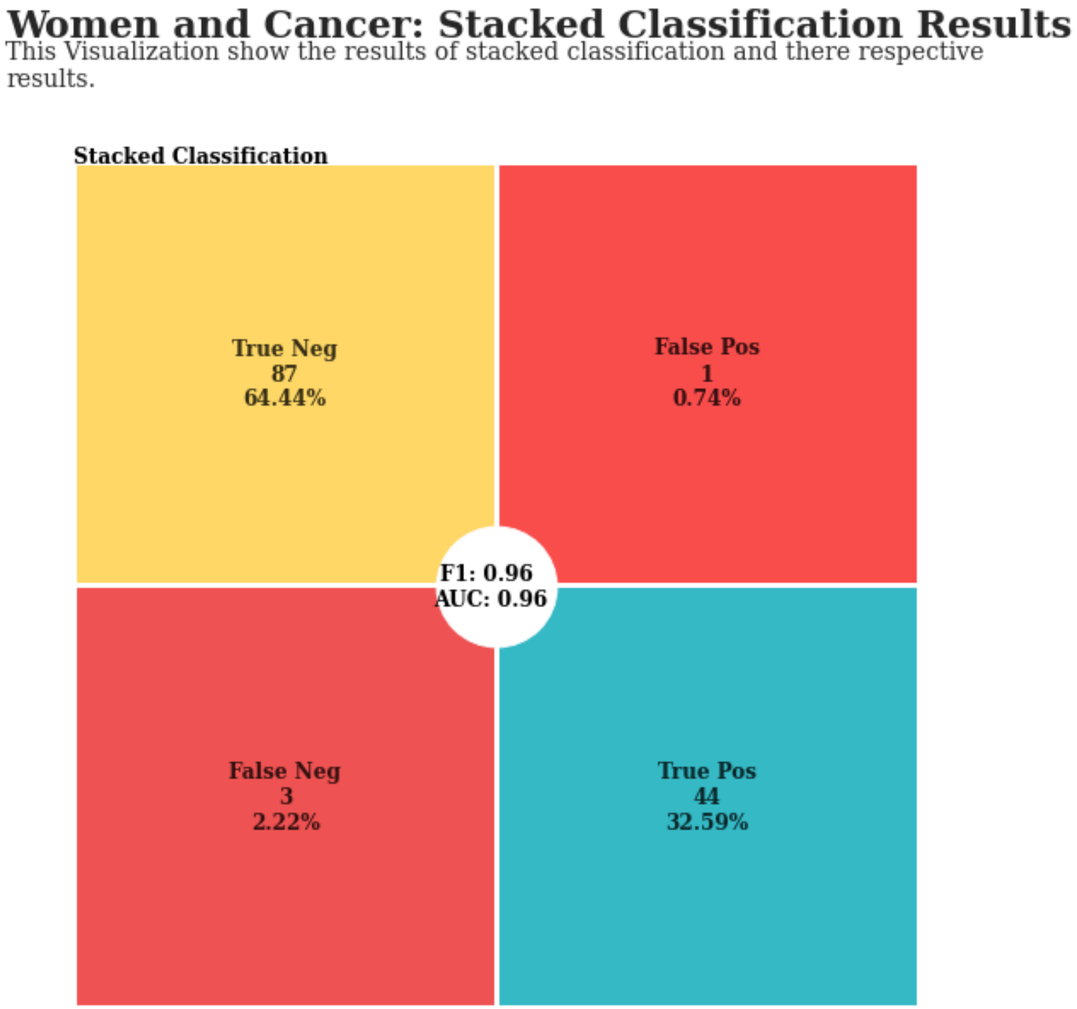

```
Gridsearch CV implementation with predefined grid params

**********

Fitting 5 folds for each of 20 candidates, totalling 100 fits

LogisticRegression(C=10)
Accuracy: 0.9777777777777777
Roc_Auc: 0.96808511
**********

Fitting 5 folds for each of 576 candidates, totalling 2880 fits

LGBMClassifier(class_weight={0: 4, 1: 6}, learning_rate=0.01, n_estimators=2600,
               num_leaves=20)
Accuracy: 0.9703703703703703
Roc_Auc: 0.96240329
**********

Fitting 5 folds for each of 60 candidates, totalling 300 fits

RandomForestClassifier(class_weight={0: 4, 1: 6}, max_depth=6,
                       max_leaf_nodes=15)
Accuracy: 0.9703703703703703
Roc_Auc: 0.96240329
**********

Fitting 5 folds for each of 6 candidates, totalling 30 fits

AdaBoostClassifier(base_estimator=DecisionTreeClassifier(), learning_rate=0.1,
                   n_estimators=600)
Accuracy: 0.9481481481481482
Roc_Auc: 0.95031431
```

3.5 Modeling

Stacked Classification

```
models = best_est

stack_train,stack_test = stacking(models = best_est,
                                  X_train = xtrain,
                                  y_train = ytrain,
                                  X_test = xtest,
                                  regression = False,
                                  metric = 'roc_auc',
                                  n_folds = 5,shuffle = True,
                                  stratified = True)

fin_model = XGBClassifier(eval_metric='logloss')

fin_model.fit(stack_train, ytrain)
fin_preds = fin_model.predict(stack_test)
print(color_class.BOLD_COLOR+'Stacked Classification...'+'₩n'+ color_class.END)
print(color_class.BOLD)
print('accuracy: {}'.format(round(accuracy_score(ytest,fin_preds),3)))
print('roc_auc: {}'.format(round(roc_auc_score(ytest,fin_preds),3)))
print('f1:{}'.format(round(f1_score(ytest,fin_preds),3)))
```

```
Stacked Classification...


accuracy: 0.97
roc_auc: 0.962
f1:0.957
```

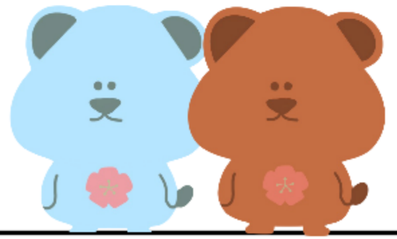


3.6 Summary

노트북에서 다뤄본 기법들

1. 일변량/이변량/다변량 분석
2. 다변수 피처 이상치 처리 방법 -> DBSCAN이나 IsolationForest
3. 다중공선성 해결 방법
4. 교차 검증과 gridsearch CV, stacked classification 을 이용한 모델링

THANK YOU



EWHA EURON