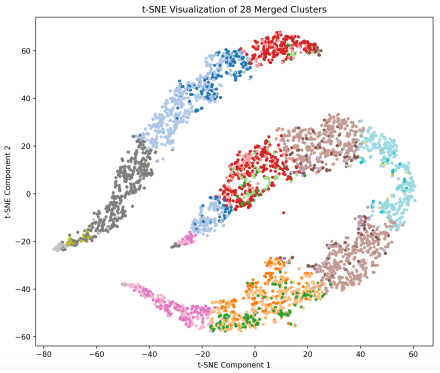
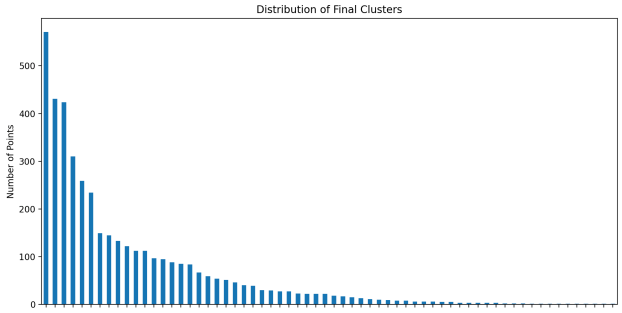
[A4] Clustering and Feature Engineering 2021313075 백경인

1. 전복 데이터로 # Data point: 4177 / # feature: 8
2. 우선 전처리로는 feature scaling 방식을 택했습니다. Minmax 와 standard scaling 을 해봤지만 standard scaling 이 더 성능을 보였습니다. Feature Engineering 으로는 **Nested Binning (Discretization)(있는 용어인지 모르겠지만) 과 feature selection, PCA 를 활용한 feature extraction 을 사용했습니다. 그중 제가 생각해낸 아이디어 Nested Binning 에 차원이 8 개라 그렇게 크다고 판단하진 않아서 feature selection 보단 extraction 기법인 PCA 를 설정했습니다. Nested Binning** 이 k-means 할 때 binning 으로 구간을 discretize 를 이중으로 했습니다. y label 이 1 부터 29 까지 28 을 제외한 28 개의 class 가 존재했습니다. 이때 처음 k-means 의 k 를 7 개로 설정해 7 개의 bin 을 만들었습니다. 그 이후 세부적으로 나누고 싶어 7 개로 나누어진 cluster 를 각각 4 개씩 나누었습니다. 이렇게 돌린 이유는 처음부터 28 개의 cluster 를 찾는 것보다 나이를 이야기할 때, 저는 10 대입니다 20 대입니다 30 대입니다. 혹은 20 대 중반입니다. 이런식으로 나아가는 것을 생각했습니다.
   PCA 를 사용하게 된 계기는 차원이 그렇게 크다고 판단되진 않아 feature selection 도 사용해보고 했더니 extraction 이 더 좋은 결과를 도출했습니다. 신기하게도 같이 사용했을 때보다 주로 한 가지 방법을 사용했더니 더 좋았습니다. 생각을 하게된 계기는 성별에 따른 나이가 나와있는데 남성과 여성이든 큰 차이가 없으며 infant 또한 큰 차이가 없다고 판단해서 feature 를 해석하는데 어려움이 있다 판단했습니다.
3. 클러스터 개수에 따른 평가지표 추이 확인
   클러스터 개수를 1-28 까지 진행한 결과 낮게 설정했던 6 개와 14 개가 좋은 ARI 값(0. ARI for Merged Clustering (Final Clusters vs RingBin): 0.21146)을 주었습니다. 그래서 두 가지 코드를 만들었습니다, 최적의 ari 값을 주는 코드와 결국 우리는 나이를 정확하게 맞추는 모델을 만들려면 결과적으로 28 개의 클러스터로 진행해야한다고 판단했습니다.
4. 최적의 클러스터링 결과 2 차원 시각화 (t-Stochastic Neighbor Embedding 사용) ->



5. t-SNE 는 시각화를 위한 도구이지만 manifold 개념에 관심을 갖게 되었습니다. 만약 manifold 개념을 가져와 이를 feature engineering 의 도구로 쓰면 어떨까도 생각했습니다. 그리고 클러스터링 성능 평가를 위해 정확한 28 개의 클러스터링으로 정확한 나이를 맞추는 ari 값보다는 범주화를 하는게 좋다고 생각했습니다. 클러스터 내 데이터 개수가 비대칭적이기에 어느정도 1 살부터 몇 살 이런식으로 범주화하는 것이 도움이 될 것 같습니다.



1. cluster 28 개로 나눈(정확히 나이맞추기 코드)

```python
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder
from sklearn.metrics import adjusted_rand_score
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
```

Python

Flags to toggle PCA and Feature Selection

```python
use_pca = True
use_feature_selection = False
```

Python

Load the Abalone dataset

```python
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.data"
column_names = ["Sex", "Length", "Diameter", "Height", "WholeWeight", "ShuckedWeight", "VisceraWeight", "ShellWeight", "Rings"]
data = pd.read_csv(url, header=None, names=column_names)
```

Python

Encode 'Sex' as numeric

```python
data['Sex'] = data['Sex'].map({'M': 0, 'F': 1, 'I': 2})
```

Python

Case 1: 4-7

Bin Rings (Binning from 1-4, 5-9, 10-14, etc.)

```python
bins = [0, 4, 8, 12, 16, 20, 24, 29]
labels = [0, 1, 2, 3, 4, 5, 6]
data['RingBin'] = pd.cut(data['Rings'], bins=bins, labels=labels, include_lowest=True)
```

Python

Separate features and labels

```python
X = data.drop(columns=['Rings', 'RingBin'])
y = data['RingBin']
y_real = data["Rings"]
```

Python

Standardize features
scaler = MinMaxScaler --> Standard is better!

```python
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Python

Feature Selection

```python
if use_feature_selection:
    selector = SelectKBest(score_func=f_classif, k=6)
    X_processed = selector.fit_transform(X_scaled, y)
    print(f"Feature selection reduced the number of features to {X_processed.shape[1]}.")
else:
    X_processed = X_scaled
```

Python

PCA for Dimensionality Reduction

```python
if use_pca:
    pca = PCA(n_components=0.95, random_state=42)
    X_processed = pca.fit_transform(X_processed)
    print(f"PCA reduced the number of features to {X_processed.shape[1]}.")
else:
    print("PCA skipped.")
```

Python

Step 1: Perform K-Means clustering with 7 clusters (global clustering)

```python
kmeans_7 = KMeans(n_clusters=7, n_init=100, random_state=42)
data['Cluster_label'] = kmeans_7.fit_predict(X_processed)
```

Python

Step 2: Perform K-Means clustering within each bin (local clustering)

```python
for bin_label in labels:
    bin_data = data[data['Cluster_label'] == bin_label]
    if not bin_data.empty:
        kmeans_bin = KMeans(n_clusters=4, random_state=42)
        bin_clusters = kmeans_bin.fit_predict(X_processed[bin_data.index])
        cluster_label = [f"{bin_label}_{cluster}" for cluster in bin_clusters]
        data.loc[bin_data.index,"Cluster_label" ] = cluster_label
```

Python

Map 'Cluster_label' to numeric values for t-SNE and ARI computation

```python
data['Cluster_numeric'] = data['Cluster_label'].astype('category').cat.codes
```

Python

+ Code + Markdown

Extract features (all columns except 'Cluster_label' and 'Cluster_numeric')

```python
features = data.drop(columns=['Cluster_label', 'Cluster_numeric']).values
cluster_labels = data['Cluster_numeric'].values
```

Python

Apply t-SNE for visualization

```python
tsne = TSNE(n_components=2, random_state=42)
tsne_results = tsne.fit_transform(features)
```

Python

Plot t-SNE results

```python
plt.figure(figsize=(10, 7))
scatter = plt.scatter(tsne_results[:, 0], tsne_results[:, 1], c=cluster_labels, cmap='tab20', s=30, alpha=0.7)
plt.colorbar(scatter, label='Cluster Label')
plt.title('t-SNE Visualization of Clusters', fontsize=14)
plt.xlabel('t-SNE Dimension 1', fontsize=12)
plt.ylabel('t-SNE Dimension 2', fontsize=12)
plt.show()
```

Python

Compute ARI score

```python
ari_score = adjusted_rand_score(y_real, cluster_labels)
print(f"Adjusted Rand Index (ARI): {ari_score:.4f}")
```

Python

2. 최적의 결과값(ARI: 0.21146)

```python
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder
from sklearn.metrics import adjusted_rand_score
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
```

Python

Flags to toggle PCA and Feature Selection

```python
use_pca = True
use_feature_selection = False
```

Python

Load the Abalone dataset

```python
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.data"
column_names = ["Sex", "Length", "Diameter", "Height", "WholeWeight", "ShuckedWeight", "VisceraWeight", "ShellWeight", "Rings"]
data = pd.read_csv(url, header=None, names=column_names)
```

Python

Encode 'Sex' as numeric

```python
data['Sex'] = data['Sex'].map({'M': 0, 'F': 1, 'I': 2})
```

Python

Bin Rings (Binning from 1-4, 5-9, 10-14, etc.)

```python
bins = [0, 4, 8, 12, 16, 20, 24, 29]
labels = [0, 1, 2, 3, 4, 5, 6]
data['RingBin'] = pd.cut(data['Rings'], bins=bins, labels=labels, include_lowest=True)
```

Python

Separate features and labels

```python
X = data.drop(columns=['Rings', 'RingBin'])
y = data['RingBin']
```

Python

Standardize features

```python
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Python

Feature Selection

```python
if use_feature_selection:
    selector = SelectKBest(score_func=f_classif, k=6)
    X_processed = selector.fit_transform(X_scaled, y)
    print(f"Feature selection reduced the number of features to {X_processed.shape[1]}.")
else:
    X_processed = X_scaled
```

Python

PCA for Dimensionality Reduction

```python
if use_pca:
    pca = PCA(n_components=0.95, random_state=42)
    X_processed = pca.fit_transform(X_processed)
    print(f"PCA reduced the number of features to {X_processed.shape[1]}.")
else:
    print("PCA skipped.")
```

Python

Step 1: Perform K-Means clustering with 6 clusters (global clustering)

```python
kmeans_7 = KMeans(n_clusters=7, n_init=100, random_state=42)
data['Cluster_7'] = kmeans_7.fit_predict(X_processed)
```

Step 2: Perform K-Means clustering within each bin (local clustering)

```python
for bin_label in labels:
    bin_data = data[data['RingBin'] == bin_label]
    if not bin_data.empty:
        kmeans_bin = KMeans(n_clusters=4, random_state=42)
        bin_clusters = kmeans_bin.fit_predict(X_processed[bin_data.index])
        data.loc[bin_data.index, f'Cluster_Bin_{bin_label}'] = bin_clusters
```

Combine all clusters into a final label

```python
data['Final_Cluster'] = data['Cluster_7'].astype(str)
for bin_label in labels:
    if f'Cluster_Bin_{bin_label}' in data.columns:
        data['Final_Cluster'] += '_' + data[f'Cluster_Bin_{bin_label}'].fillna(-1).astype(int).astype(str)
```

Encode Final_Cluster to numeric labels for visualization and ARI computation

```python
label_encoder = LabelEncoder()
merged_clusters = label_encoder.fit_transform(data['Final_Cluster'])
```

Step 3: Apply t-SNE on the processed features

```python
tsne = TSNE(n_components=2, random_state=42, perplexity=30, n_iter=1000)
X_tsne = tsne.fit_transform(X_processed)
```

Plot the t-SNE results for 28 clusters

```python
plt.figure(figsize=(12, 8))
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=merged_clusters, cmap='tab20', s=10)
plt.title("t-SNE Visualization of 28 Merged Clusters")
plt.colorbar(label="Merged Cluster")
plt.xlabel("t-SNE Component 1")
plt.ylabel("t-SNE Component 2")
plt.show()
```

Step 4: Compute ARI for Merged Clusters (Final Clusters vs RingBin)

```python
ari_merged = adjusted_rand_score(data['RingBin'].astype(float), merged_clusters)
print(f"ARI for Merged Clustering (Final Clusters vs RingBin): {ari_merged:.4f}")
```

Step 5: Optional - Plot distribution of final clusters

```python
final_cluster_counts = pd.Series(merged_clusters).value_counts()
final_cluster_counts.plot(kind='bar', figsize=(12, 6), title="Distribution of Final Clusters")
plt.xlabel("Cluster Labels")
plt.ylabel("Number of Points")
plt.show()
```