군집화 실습 - 고객 세그먼테이션

고객 세그먼테이션(Customer Segmentation)
: 다양한 기준으로 고객을 분류하는 기법(어떤 상품을 얼마나 많은 비용을 써서 얼마나 자주 사용하는가)

주요 목표: 타겟 마케팅→ 고객을 여러 특성에 맞게 세분화해서 그 유형에 따라 맞춤형 마케팅이나 서비스를 제공하는 것

⇒ 고객의 어떤 요소를 기반으로 군집화할 것인가를 결정하는 것이 중요!

(RFM 기법→ Recency(가장 최근 상품 구입 일에서 오늘까지의 기간), Frequency(상품 구매 횟수), Monetary value(총 구매 금액))

데이터 세트 로딩과 데이터 클렌징

```
import pandas as pd
import datetime
import math
import numpy as np
import matplotlib.pyplot as plt

retail_df = pd.read_excel('Online Retail.xlsx')
retail_df.head(3)
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom

→ 제품 주문 데이터 세트

→ Invoice(주문번호) + StockCode(제품코드)를 기반으로 주문량, 주문 일자, 제품 단가, 주문고객번호, 주문 고객 국가의 칼럼으로 구성

→ CustomerID의 NULL값 다수 존재

- 데이터 정제

1. Null 데이터 제거: CustomerID에 Null인 데이터가 많으므로, 고객 세그먼테이션을 수행하는데 있어 고객 식별번호가 없는 데이터는 필요없으므로 삭제
2. 오류 데이터 삭제: Quantity, UnitPrice가 0보다 작은 경우 → 데이터 삭제

```
retail_df = retail_df[retail_df['Quantity']>0]
retail_df = retail_df[retail_df['UnitPrice']>0]
retail_df = retail_df[retail_df['CustomerID'].notnull()]
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
```

➕retail_df['Country'].value_counts()를 했을 때 주요 주문 고객이 영국이므로, 다른 국가 데이터를 제외

```
retail_df = retail_df[retail_df['Country']=='United Kingdom']
print(retail_df.shape)
```

```
(354321, 8)
```

RFM 기반 데이터 가공

- Unitprice와 Quantity를 곱하여 주문 금액 데이터 생성 및 CustomerID를 int형으로 바꾸기

```
retail_df['sale_amount'] = retail_df['UnitPrice']*retail_df['Quantity']
retail_df['CustomerID'] = retail_df['CustomerID'].astype(int)
```

RFM 기반의 고객 세그먼테이션은 고객 레벨로 주문 기간, 주문 횟수, 주문 금액 데이터를 기반으로 해 세그먼테이션 수행

→ 주문번호 + 상품코드 기준의 데이터를 **개별 고객 기준의 데이터로 Group by!**

```
# DataFrame의 groupby() 의 multiple 연산을 위해 agg() 이용
# Recency는 InvoiceDate 컬럼의 max() 에서 데이터 가공
# Frequency는 InvoiceNo 컬럼의 count() , Monetary value는 sale_amount 컬럼의 sum()
aggregations = {
    'InvoiceDate': 'max',
    'InvoiceNo': 'count',
    'sale_amount':'sum'
}
cust_df = retail_df.groupby('CustomerID').agg(aggregations)
# groupby된 결과 컬럼값을 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(3)
```

	CustomerID	Recency	Frequency	Monetary
0	12346	2011-01-18 10:01:00	1	77183.60
1	12747	2011-12-07 14:34:00	103	4196.01
2	12748	2011-12-09 12:20:00	4595	33719.73

➕Recency는 오늘 날짜를 기준으로 가장 최근 주문 일자를 뺀 날짜

(데이터가 2010.12.1 ~ 2011.12.9까지의 데이터이므로 오늘 날짜를 2011.12.10로 간주해 Recency 수정)

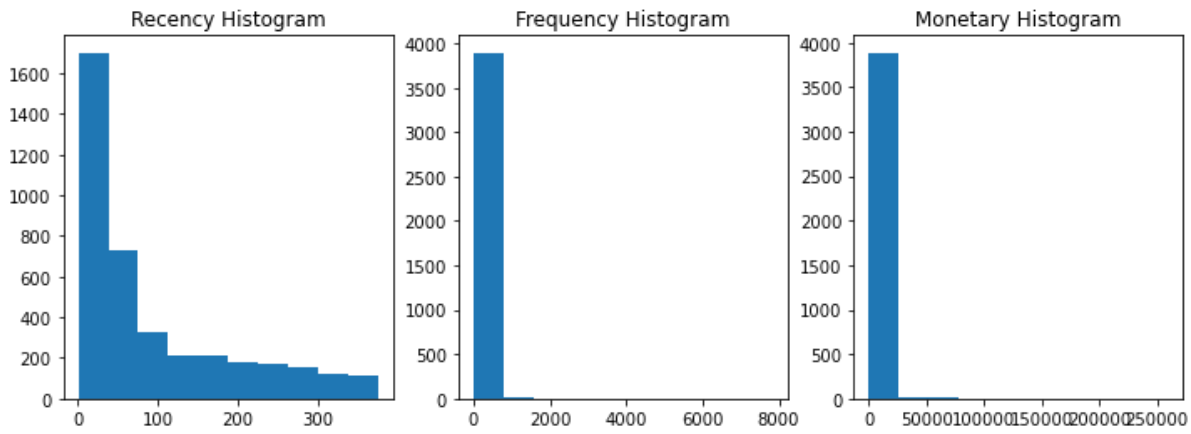
```
import datetime as dt

cust_df['Recency'] = dt.datetime(2011,12,10) - cust_df['Recency']
cust_df['Recency'] = cust_df['Recency'].apply(lambda x: x.days+1)
cust_df.head(3)
```

	CustomerID	Recency	Frequency	Monetary
0	12346	326	1	77183.60
1	12747	3	103	4196.01
2	12748	1	4595	33719.73

RFM 기반 고객 세그먼테이션

- 칼럼별 분포 히스토그램

```
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'])
```



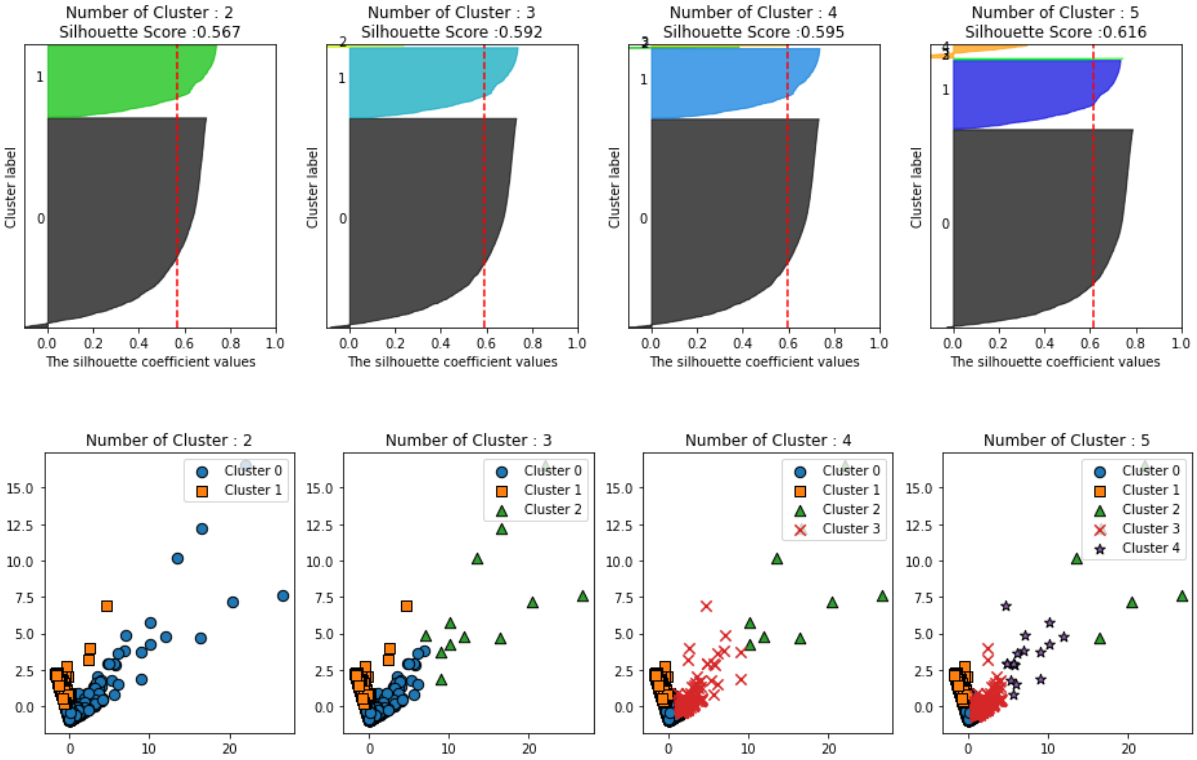
→ Recency, Frequency, Monetary 모두 왜곡된 데이터 값 분포도를 가짐(소매업체의 대규모 주문을 포함하고 이어 개인 고객 주문과 주문 횟수 및 주문 금액에서 매우 큰 차이)

❓왜곡 정도를 해소하지 않고 군집화를 실행한다면

```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
```

```
from sklearn.metrics import silhouette_score, silhouette_samples

X_features = cust_df[['Recency','Frequency','Monetary']].values
X_features_scaled = StandardScaler().fit_transform(X_features)

kmeans = KMeans(n_clusters=3, random_state=0)
labels = kmeans.fit_predict(X_features_scaled)
cust_df['cluster_label'] = labels

print('실루엣 스코어는 : {0:.3f}'.format(silhouette_score(X_features_scaled,labels)))
```

```
실루엣 스코어는 : 0.592
```



→ 데이터 값이 거리 기반으로 광범위하게 퍼져 이어 군집 수를 계속 늘려봐도 의미 없는 군집화 결과로 이어진다.

⇒ 지나치게 왜곡된 데이터 세트는 K-Means와 같은 거리 기반 군집화 알고리즘에서 지나치게 일반적인 군집화 결과를 도출

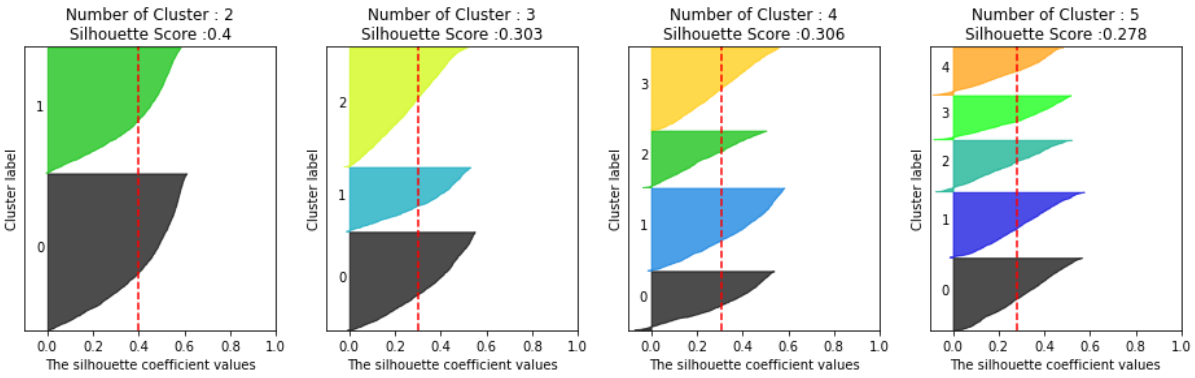
⇒ 데이터 세트의 왜곡 정도를 낮추기 위해 로그 변환을 한 뒤 K-Means 알고리즘을 적용하자!

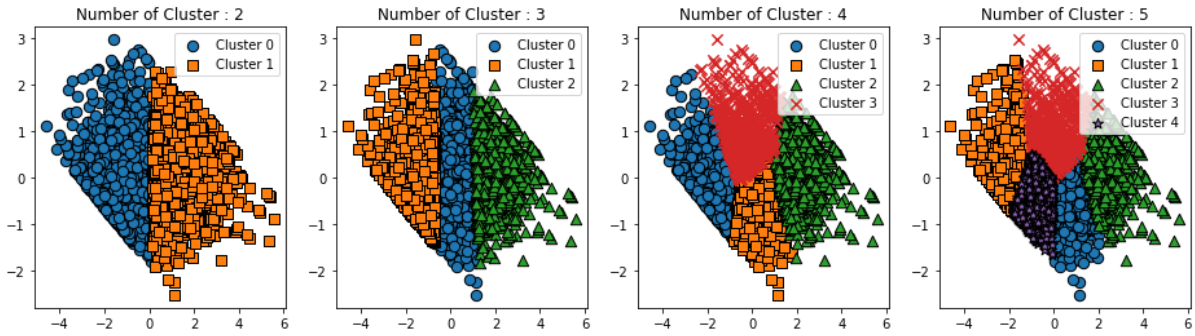
```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, silhouette_samples
```

```
#Recency, Frequency, Mpnetary 칼럼에 np.log1p()로 로그 변환
cust_df['Recency_log'] = np.log1p(cust_df['Recency'])
cust_df['Frequency_log'] = np.log1p(cust_df['Frequency'])
cust_df['Monetary_log'] = np.log1p(cust_df['Monetary'])

#로그 변환 데이터에 StandardScaler 적용
X_features = cust_df[['Recency_log', 'Frequency_log', 'Monetary_log']].values
X_features_scaled = StandardScaler().fit_transform(X_features)

kmeans = KMeans(n_clusters=3, random_state=0)
labels = kmeans.fit_predict(X_features_scaled)
cust_df['cluster_label'] = labels
print('실루엣 스코어는:', silhouette_score(X_features_scaled, labels))
```

```
실루엣 스코어는: 0.3033967048804945
```

→ 로그 변환 전보다 실루엣 스코어는 더 떨어지지만 실루엣 스코어의 절대치가 중요한 것이 아니고, **어떻게 개별 군집이 더 균일하게 나뉠 수 있는지가 더 중요함**





⇒ 로그 변환 하기 전보다 더 균일하게 군집화가 구성됨