04장. 고객의 행동을 예측하는 테크닉

> 💡
> **REMEMBER**

- 클러스터링을 적용할 때 , **특징적인 변수를 포함**시키면 보다 복잡한 클러스터링도 가능.
- 클러스터링 결과를 시각화하고 싶을 때는, PCA 등의 **차원 축소를 사용해서 2차원으로** 나타냄.
  그 후 2개의 축이 **어떤 변수로 구성되는지**를 분석해서 파악하는 것이 일반적
- 클러스터링을 통한 그룹화를 실시함으로써, 고객의 특징을 파악할 수 있음.
  분석의 목적에 따라 다양한 방식의 분석을 진행하는 것이 좋음.
- 머신러닝을 진행할 때는 결측치를 처리해야 하므로 **결측치 처리 후 index를 초기화**하는 것이 좋음
- 기간에 따라 예측을 진행할때, 오래된 데이터는 가입 시기 데이터가 존재하지 않거나 이미 안정적인 가능성이 있기 때문에 **비교적 신규 데이터로 예측**.
- 정확도가 높은 모델을 구축하더라도 설명이 불가능하면 실제로 사용하기 힘듬.
  정확도가 높은 블랙박스 모델보다 **정확도가 낮아도 설명이 가능한 모델**이 사용되는 경우가 많음

> 📚
> **PLUS TOPIC**

🔶표준화
🔶잔차

🔶표준화

- Reference

  https://mizykk.tistory.com/101

  https://mkjjo.github.io/python/2019/01/10/scaler.html

- 데이터 표준화(Standardization)
  - 데이터의 모든 특성의 범위를 같게 만들어주는 방법
  - Train-Test로 분리한 경우 훈련 데이터에 대해서만 fit()을 적용해야 함
  - 스케일링을 통해 다차원의 값들을 비교 분석하기 쉽게 만듬
  - 최적화 과정에서의 안정성 및 수렴 속도 향상
    - 오버플로우나 언더플로우를 방지
    - 독립 변수의 공분산 행렬의 조건수를 감소
    - 최적화 과정에서의 안정성 및 수렴 속도 향상

- **StandardScaler**(=표준화)
  - 평균을 0으로, 표준편차를 1로 변환
  - 평균을 제거하고 데이터를 단위 분산으로 조정
  - 이상치가 있다면 평균과 표준편차에 영향을 미쳐 변환된 데이터의 확산이 달라짐

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler = scaler.fit_transfrom(data)
```

```
scaler.fix(x_train)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)
```

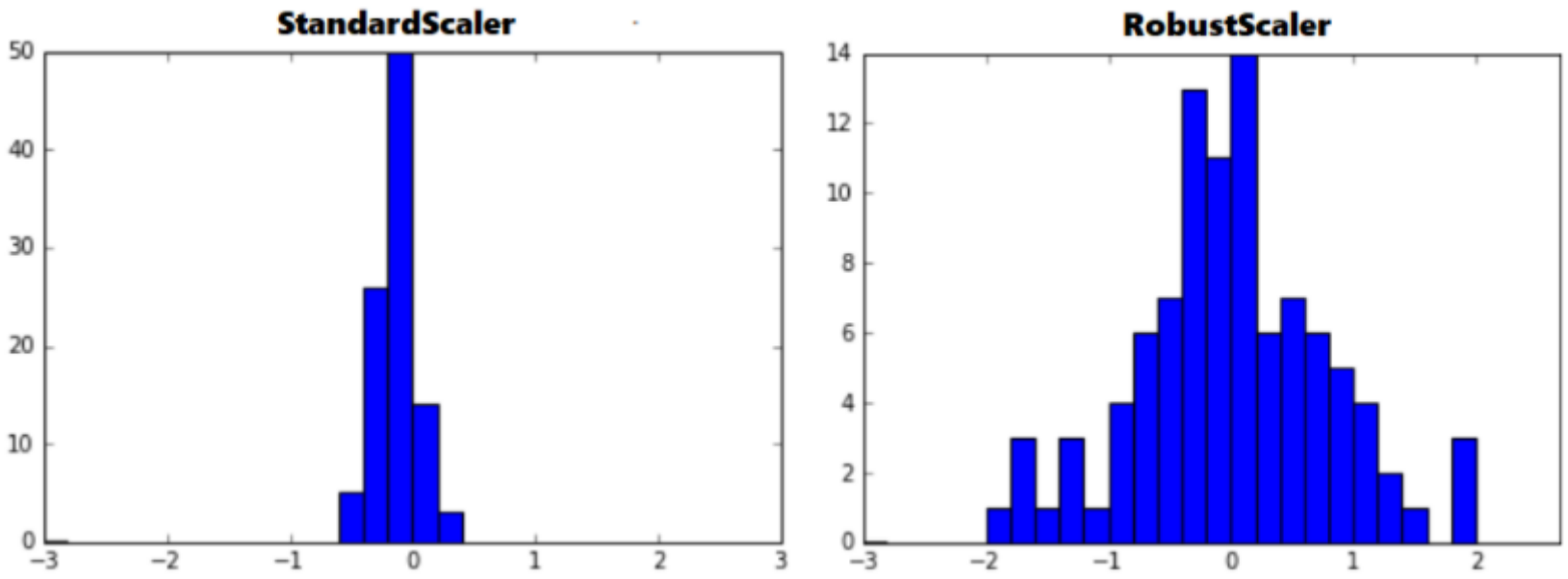
- **MinMaxScaler**(=최소-최대 정규화)
  - 최대값을 1로, 최솟값을 0으로 변환
  - 즉, 모든 feature 값이 0~1 사이에 있도록 데이터를 재조정
  - 이상치가 있는 경우 변환된 값이 매우 좁은 범위로 압축 → 이상치에 민감

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler = scaler.fit_transform(data)

scaler.fit(x_trian)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)
```

- **RobustScaler**
  - 중앙값을 0으로, IQR을 1로 변환
  - 이상치 영향 최소화

▼ StandardScaler와 비교하면 표준화 후 동일한 값을 넓게 분포




```
from sklearn.preprocessing import RobustScaler
scaler = RobusterScaler()
scaler = scaler.fit_transform(data)

scaler.fit(x_train)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)
```

- **MaxAbsScaler**
  - 0을 기준으로 절대값이 가장 큰 수가 1 또는 -1이 되도록 변환
  - 즉, 절댓값이 0~1 사이에 매핑 = 데이터를 -1~1 사이로 재조정
  - 양수 데이터로만 구성된 데이터셋에서는 MinMaxScaler와 유사하게 동작
  - MinMaxScaler와 유사하나 음수와 양수값에 따른 대칭 분포를 유지
  - 큰 이상치에 민감

```
from sklearn.preprocessing import MaxAbsScaler
scaler = MaxAbsScaler()
scaler = scaler.fit_transform(data)

scaler.fit(x_train)
```

```
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)
```

- 스케일러 처리 전에는 아웃라이어를 처리하는 것이 좋음

▼ 데이터의 분포 특징에 따라 적절한 스케일러 적용

원본 / minMaxScale / maxAbsScale / standardScale / robustScale 순

- 스케일링시 feature 별로 크기를 유사하게 만드는 것은 중요하지만, 그렇다고 모든 feature의 분포를 동일하게 만들 필요는 없음 (특성에 따라 어떤 항목은 원본데이터의 분포를 유지하는 것이 유의할 수도 있음)

🔶잔차

- Reference

  https://jangpiano-science.tistory.com/116

  https://m.blog.naver.com/samsjang/221003939973

  https://mindscale.kr/course/basic-stat-python/14/

- 표본으로 추정한 회귀식과 실제 관측값의 차이

- 분석에 사용하는 데이터는 보통 표본집단의 데이터이기 때문에 잔차를 기준으로 회귀식을 추정
- 회귀 모델의 적합도는 예측값에 대한 잔차가 0에 가까울수록 좋음
- 잔차 0 기준선으로부터 멀리 떨어진 데이터는 이상치로 분류
- 오차가 정규분포를 따른다면, 회귀모형 역시 정규분포를 따르고
  잔차는 회귀모형에 선형결합으로 표현될 수 있기에 잔차 역시 정규분포를 따름

잔차 분석

- 잔차의 정규성
  - 잔차가 정규분포를 따른다는 가정
  - Q-Q 플롯으로 확인 : 점들이 점선을 따라 배치되어 있어야 정규분포

```
import scipy.stats
import seaborn as sns

sr = scipy.stats.zscore(residual)
(x,y),_ = scipy.stats.probplot(sr)
sns.scatterplot(x,y)
plt.plot([-3,3],[-3,3],'--',color='grey')
scipy.stats.shapiro(residual) # 잔차의 정규성 검정 - 결과의 두번째 값이 p값
```

- 잔차의 등분산성
  - 예측된 모든 값에 대하여 잔차의 분산이 동일하다는 가정
  - 예측값에 따라 잔차가 어떻게 달라지는지 시각화

```
import numpy as np
import seaborn as sns

sns.regplot(fitted,np.sqrt(np.abs(sr)),lowess=True,line_kws={'color':'red'})
```

- 극단값
  - Cook's distance이 극단값을 나타내는 지표

```
from statsmodels.stats.outliers_influence import OLSInfluence

cd,_ = OLSInfluence(res).cooks_distance
cd.sort_values(ascending=False).head()
```

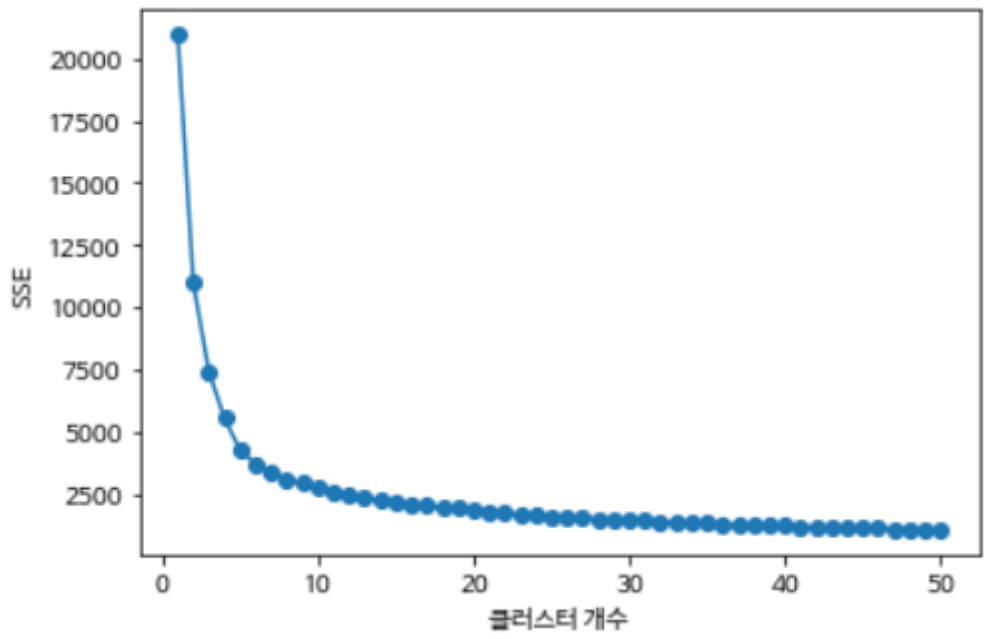
- 잔차의 독립성
  - 자료 수집 과정에서 random sampling을 했다면 잔차의 독립성은 만족한다고 판단
  - 시계열 자료, 종단연구 자료 등 설계 자체가 독립성을 담보할 수 없는 경우네는 더빈-왓슨 검정 등을 실시

- 잔차 분석 결과에 따라 다양한 대응이 가능
  - 극단값을 제거
  - 독립변수를 추가
  - 종속변수를 수학적으로 변환

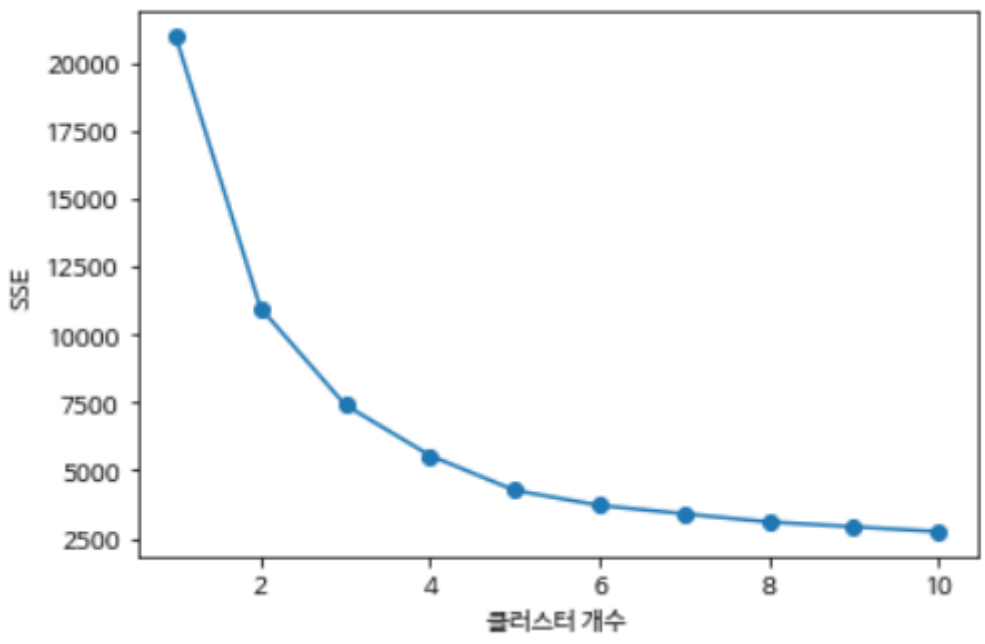
> 📌
> **ADVANCED CODING**

1. **클러스터링 적용 시 최적의 K 값 찾아보기**
   (https://m.blog.naver.com/samsjang/221017639342 **- elbow**
   https://ariz1623.tistory.com/224 **- silhoutte**)
   1. 설계
      - 동일한 변수 , 표준화 사용
      - 4192개의 데이터
      - Elbow Method , Silhouette 사용하여 최적 값 찾기
      - 1~50개로 변화
   2. 결과

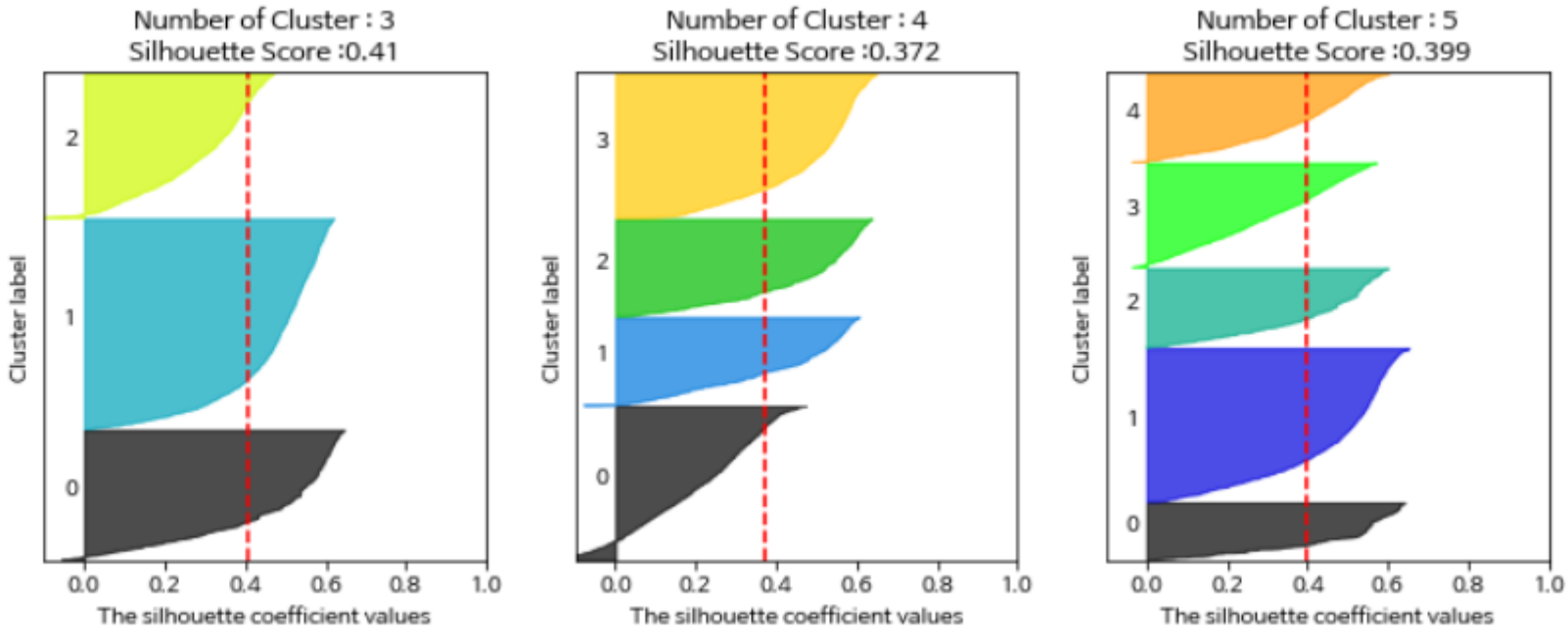
      ▼ 1~50 개는 불필요하게 많은 계산



▼ 중요한 포인트가 되어 보이는 1~10개로 다시 계산



▼ k값 설정이 가능해보이는 3,4,5만을 Silhouette 계산



- k=3을 최적으로 결정

2. **최적으로 판단한 k값으로 클러스터링 진행**

▼ 결과

cluster	월평균값	월중앙값	월최댓값	월최솟값	회원기간
0	1125	1125	1125	1125	1125
1	1823	1823	1823	1823	1823
2	1244	1244	1244	1244	1244

cluster	월평균값	월중앙값	월최댓값	월최솟값	회원기간
0	7.676584	7.640000	9.903111	5.550222	8.673778
1	5.009976	5.007954	7.650576	2.394953	32.374657
2	3.687399	3.445338	6.197749	1.719453	8.185691

- Cluster 0 : 회원기간은 길지 않지만 , 이용횟수가 높은 그룹
- Cluster 1 : 회원기간이 길지만, 이용빈도는 낮은 그룹
- Cluster 2 : 회원기간도 길지 않고, 이용빈도도 낮은 그룹

1. **클러스터링 적용 전 진행하는 표준화 방법 바꿔보기**

   StandardScaler와 비교하면(쉬운 비교를 위해 회원 기간만 비교) ,

   - RobustScaler와 MaxAbsSclaer의 경우 기존 회원기간의 평균이 36정도이던 그룹이 32로 변화
   - 그 외에는 그룹의 순서가 바뀌었다고 판단 가능하므로 해당 데이터에서는 스케일러의 종류에 크게 영향을 받지 않는다고 판단

2. **PCA 구성 변수 파악하기**
   (https://m.blog.naver.com/tjdrud1323/221720259834 **- pca_variance_ratio**
   https://studychfhd.tistory.com/228 **- pca_components**)

   ▼ 결과

```
pca.explained_variance_ratio_
```

```
array([0.69042666, 0.18937526])
```

```
sum(pca.explained_variance_ratio_)
```

```
0.8798019226140575
```

	PC1	PC2
월평균값	0.532650	-0.109719
월중앙값	0.513845	-0.149198
월최댓값	0.441831	-0.236745
월최솟값	0.470014	0.127829
회원기간	-0.190050	-0.945153

- PC2까지의 설명력은 약 88%
- PC1에 미치는 영향은 회원 기간이 제일 적고, 나머지 변수는 비슷
- PC2에 미치는 영향은 회원 기간이 대부분

3. **클러스터링 변수 바꿔서 진행해보기**
   (https://www.delftstack.com/ko/howto/python/extract-substring-from-a-string-in-python/ **- slice**
   https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=youji4ever&logNo=221698612004 **- applymap**
   https://hogni.tistory.com/70 **- astype**)
   1. 설계
      - class , gender , campaign_id , mean , max , min , routine_flg , membership_period 변수 선택
      - class , campain_id는 숫자만 남기기
      - gender의 경우 원-핫 인코딩 진행
   2. 결과
      - 기존에 진행한 클러스터링보다 실루엣 계수가 절반정도 낮음
      - 가장 높은 값을 보이는 k=3으로 클러스터링 진행
      - 성능 저하의 원인이 카테고리형 변수때문이 아닐까 예측