Chapter 07. 비지도 학습 - 김가연

Chapter 07. 비지도 학습

> 비지도 학습이란, 레이블이 달린 데이터를 이용해 모델을 학습하는 과정 없이 **데이터로부터 의미를 이끌어내는 통계적 기법**을 의미한다.

1. 데이터의 의미 있는 그룹들을 찾기 위해 **클러스터링(clustering)**을 사용할 수 있다.
   ex. 웹사이트에서 사용자의 클릭 데이터와 인구통계 정보를 이용해 사용자들의 성격을 그룹화 → 그룹에 맞게 웹사이트를 개선

   💡 클러스터링은 특히 콜드스타트(cold-start) 문제에서 유용한 방법이다.
   ex. 새로운 형태의 사기나 스팸을 걸러내는 문제의 경우 모델을 훈련시킬 수 있는 데이터가 부족
   → 클러스터링은 패턴이 비슷한 데이터들을 분류하여 학습 과정을 더 빨리 시작할 수 있도록 도와준다.

2. 데이터의 변수들을 관리할 수 있을 만한 수준으로 **차원(dimension)을 줄이는 것**이 목표가 될 수 있다. 줄인 데이터는 회귀 혹은 분류 같은 예측 모델에 입력으로 사용할 수 있다.
   ex. 제조 공정 모니터링 中 수천개의 센서에서 나오는 데이터를 훨씬 작은 차원의 의미 있는 피처 데이터로 축소 → 좀 더 강력하고 해석하기 쉬운 공정 실패 예측 모델

3. 데이터와 다른 **변수들 사이에 서로 어떤 관계가 있는지**에 대한 통찰을 얻는 것이 목적이 될 수 있다.
   ex. 변수와 레코드의 수가 아주 큰 상황 → EDA 과정에서 변수들의 관계를 밝혀내는 데 유용

Chapter 07. 비지도 학습
7.1 주성분 분석
7.2 K-평균 클러스터링
7.3 계층적 클러스터링
7.4 모델 기반 클러스터링
7.5 스케일링과 범주형 변수
7.6 마치며

7.1 주성분 분석

Principal Components Analysis (PCA)

> PCA의 아이디어는 다수의 수치형 예측변수들을 더 적은 수의 변수들의 집합(principal components, 주성분)으로 나타내는 것 → 데이터의 차원 축소!

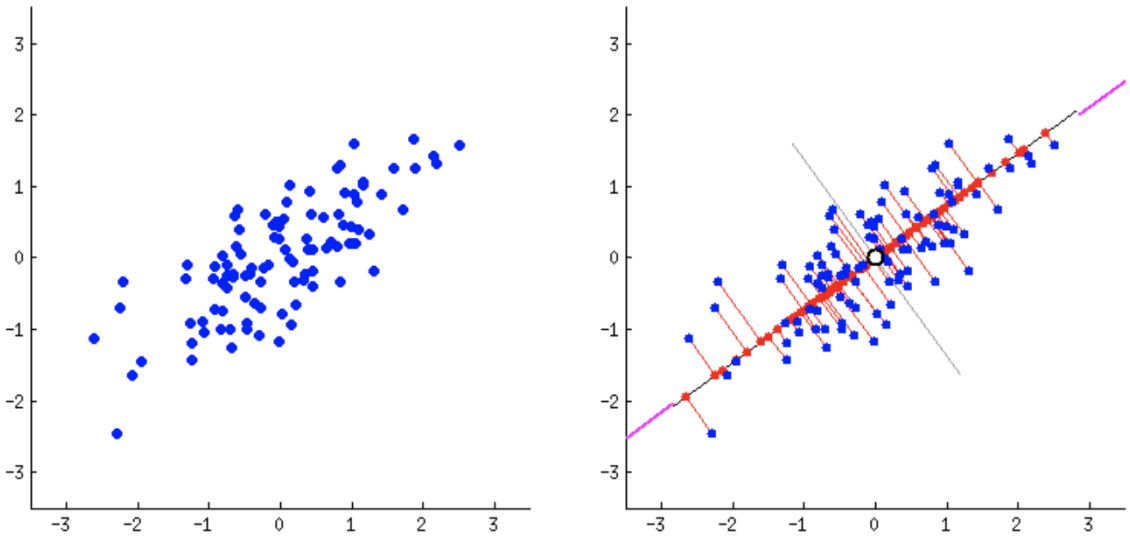
더 적은 수의 변수들의 집합(principal components, 주성분)

= 전체 변수들의 **변동성**을 거의 대부분 **설명**할 수 있는 변수들의 집합

= (이러한 변수들을 구하기 위해서는) 기존 변수들에 **가중치**를 적용한 **선형결합**

⇒ 차원 축소!!

(가중치는 새로운 주성분을 만드는 데 기존 변수들이 어느 정도 **기여**하는지 보여준다.)



PCA Before & After (2D → 1D)

🤓 간단한 예제

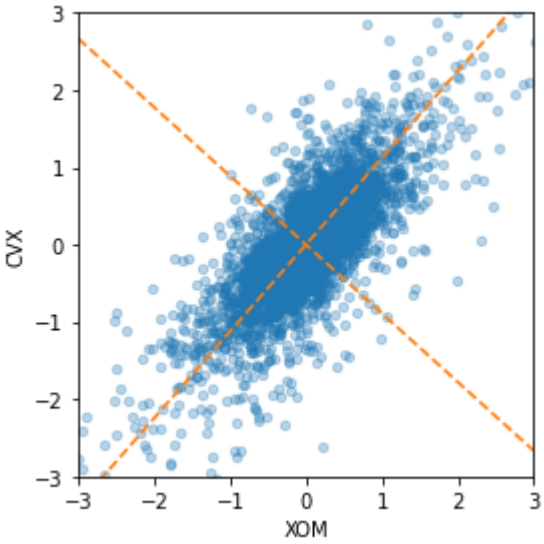
	XOM	CVX
1993-01-29	-0.016991	0.072921
1993-02-01	0.016991	0.102089
1993-02-02	0.084954	0.029168
1993-02-03	0.067964	0.058337
1993-02-04	0.034378	0.044272

oil_px.head() 출력 결과 (주가 수익 데이터)

```
from sklearn.decomposition import PCA

pcs = PCA(n_components=2) # 2개의 주성분
pcs.fit(oil_px)
loadings = pd.DataFrame(pcs.components_, columns=oil_px.columns)
# components_ : 변수에 대한 가중치

print(loadings)
```

	XOM	CVX
0	-0.664711	-0.747101
1	0.747101	-0.664711

주성분 계산 결과



데이터와 주성분을 직접 그려본 결과

- 첫 번째 주성분 (타원의 장축) : 데이터의 변동성을 가장 잘 설명하는 선형결합

  → XOM 과 CVX 두 회사의 상관관계

  ⇒ 석유 관련 주가가 한 그룹으로 움직이는 경향이 있다.

- 두 번째 주성분 (타원의 단축) : 첫 번째 주성분과 수직으로 나머지 변동성을 설명함

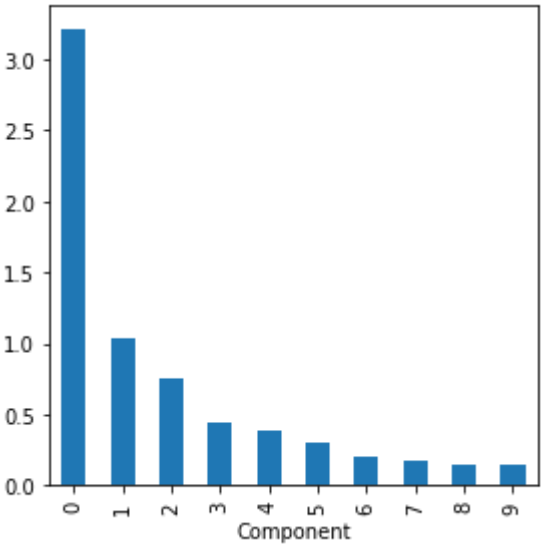
  → XOM 과 CVX 두 회사의 주가가 달라지는 경우

> 주성분 **개수**를 결정하는 데에는 각 주성분의 설명력 혹은 변수에 대한 가중치를 확인하는 방법이 있다.

🤓 주성분의 상대적인 중요도를 표시해주는 "스크리그래프"

```
syms = sorted(['AAPL', 'MSFT', 'CSCO', 'INTC', 'CVX', 'XOM', 'SLB', 'COP',
        'JPM', 'WFC', 'USB', 'AXP', 'WMT', 'TGT', 'HD', 'COST'])
top_sp = sp500_px.loc[sp500_px.index >= '2011-01-01', syms]

sp_pca = PCA()
sp_pca.fit(top_sp)
```

```
explained_variance = pd.DataFrame(sp_pca.explained_variance_)
# explained_variance_ : 주성분 벡터가 이루는 축에 투영(projection)한 결과의 분산 (설명력)

ax = explained_variance.head(10).plot.bar(legend=False, figsize=(4, 4))
ax.set_xlabel('Component')

plt.tight_layout()
plt.show()
```



주가 데이터에 대한 PCA의 스크리그래프
(0 : 첫 번째 주성분을 의미)

*왜 Scree(스크리)? 그림이 마치 절벽이 있는 산비탈 모양과 흡사하다고 해서 붙여진 이름

→ 첫 번째 주성분의 변동이 가장 크고 나머지 상위 주성분일수록 중요한 것을 볼 수 있다.

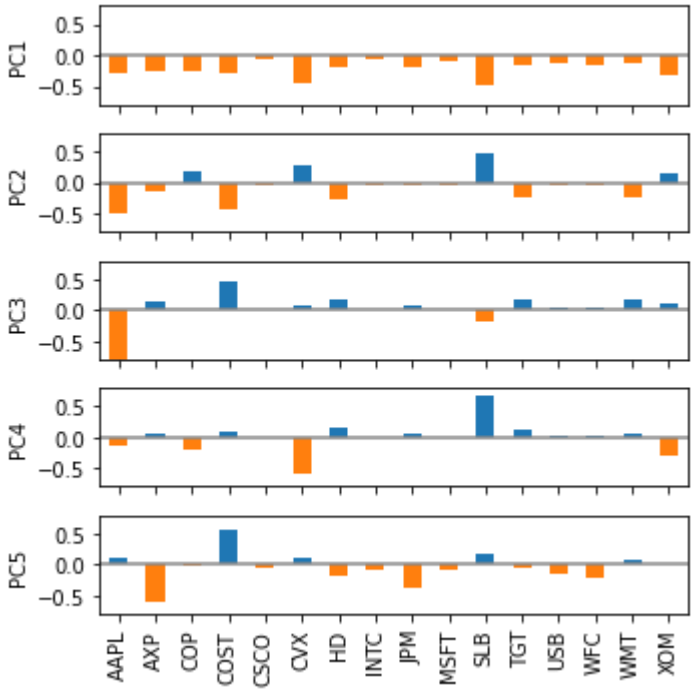
→ 보통 누적 분산(explained_variance_)이 80%가 넘어가도록 하는 개수만큼의 상위 주성분들을 선택한다. 혹은 직관적인 해석이 가능한 성분들이 있는지 알아보기 위해 가중치를 살펴볼 수 있다.

🤓 가중치 시각화를 통해 주성분 이해하기

```
# 상위 5개 주성분의 변수별 가중치 출력
loadings = pd.DataFrame(sp_pca.components_[0:5, :], columns=top_sp.columns)
print(loadings)
```

```
        AAPL       AXP       COP      COST      CSCO       CVX        HD  \
0 -0.300825 -0.246332 -0.261529 -0.273634 -0.064059 -0.444490 -0.207983
1 -0.505116 -0.139426  0.174212 -0.416307 -0.031939  0.289373 -0.278002
2 -0.786730  0.135458 -0.002367  0.465862 -0.007524  0.082374  0.166320
3 -0.120586  0.061814 -0.206026  0.092596  0.003904 -0.577665  0.162814
4  0.111576 -0.596666 -0.005813  0.555529 -0.039860  0.109016 -0.185488

        INTC       JPM      MSFT       SLB       TGT       USB       WFC  \
0 -0.076956 -0.196397 -0.105012 -0.481786 -0.148833 -0.116421 -0.145684
1 -0.033898 -0.040723 -0.053954  0.472494 -0.228123 -0.054796 -0.047427
2 -0.003518  0.062261  0.016248 -0.194822  0.160833  0.048976  0.041932
3 -0.001605  0.057687 -0.012558  0.680914  0.109895  0.016752  0.018614
4 -0.072047 -0.385160 -0.077135  0.181332 -0.055557 -0.155440 -0.216425

        WMT       XOM
0 -0.122304 -0.317952
1 -0.222889  0.154192
2  0.175806  0.090167
3  0.058439 -0.295204
4  0.091541  0.013277
```

```
maxPC = 1.01 * np.max(np.max(np.abs(loadings.loc[0:5, :])))

f, axes = plt.subplots(5, 1, figsize=(5, 5), sharex=True)

for i, ax in enumerate(axes):
    pc_loadings = loadings.loc[i, :]
    colors = ['C0' if l > 0 else 'C1' for l in pc_loadings]
    ax.axhline(color='#888888')
    pc_loadings.plot.bar(ax=ax, color=colors)
    ax.set_ylabel(f'PC{i+1}')
    ax.set_ylim(-maxPC, maxPC)

plt.tight_layout()
plt.show()
```



주가 수익 데이터의 상위 5개 주성분에 대한 변수별 가중치 시각화

- 첫 번째 주성분

  → 모두 같은 마이너스 부호로 모두가 전반적인 주식시장의 흐름에 영향을 받는다고 해석할 수 있다.

- 두 번째 주성분

  → 에너지 관련 주식들만의 가격 변동을 잡아낸다.

- 세 번째 주성분

  → 주로 애플(APPL)과 코스트코(COST)의 움직임이 서로 반대라는 사실을 보여준다.

- 네 번째 주성분

  → 슐륨베르거(SLB)와 나머지 에너지 회사들의 움직임이 반대인 것을 보여준다.

- 다섯 번째 주성분

  → 금융회사들만의 가격 변동을 잡아낸다.

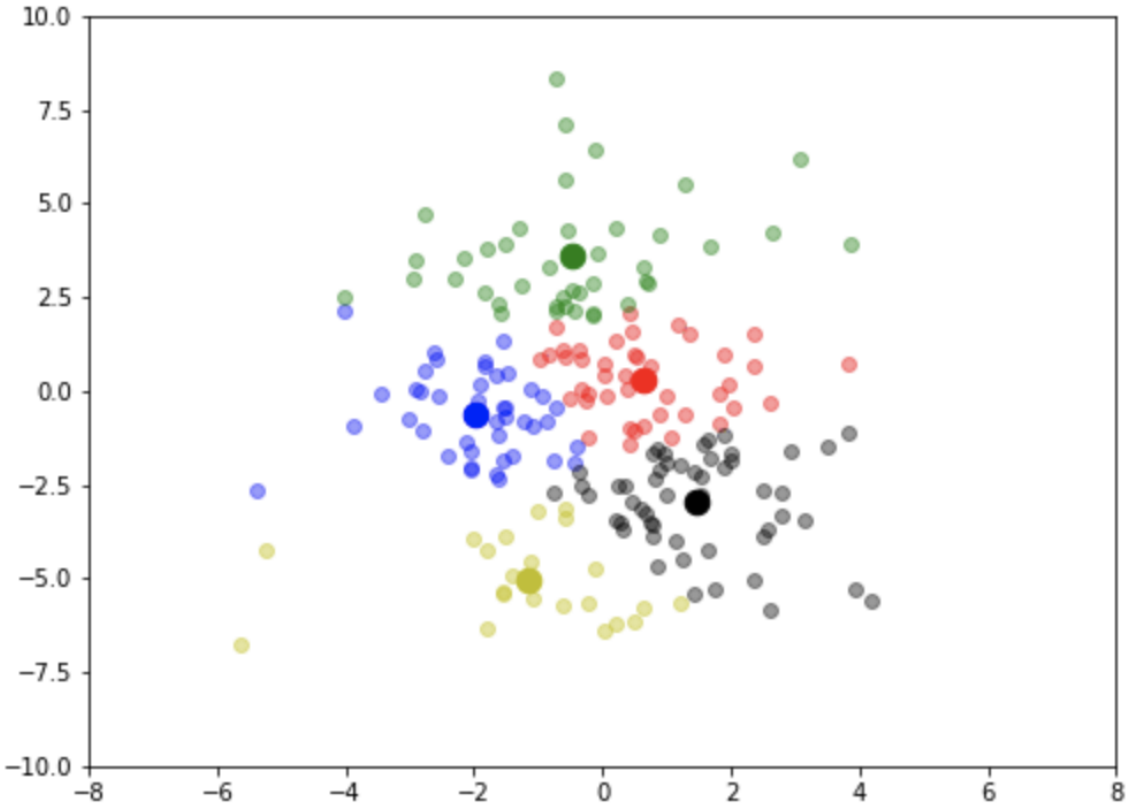
⇒ 이렇듯 주성분들은 서로 간의 상관관계가 최소화되며(각각 다른 설명), 주성분들로도 데이터의 대부분의 변동을 설명할 수 있다. 따라서 기존 변수들을 대신해 **차원이 감소된 형태로 사용**할 수 있다.

7.2 K-평균 클러스터링

K-Means Clustering

> 클러스터링의 목적은 데이터로부터 유의미한 그룹들을 구하는 것이다. K-평균은 최초로 개발된 클러스터링 기법으로 데이터를 K개의 클러스터(군집)으로 나눈다.

- 클러스터의 평균(=클러스터의 중심), 즉 한 클러스터 안에 속한 레코드들의 **평균 벡터 변수와 클러스터 내 데이터들의 거리 제곱합이 최소**가 되도록 한다.
  - 사용자가 미리 지정해 준 K 값과 클러스터 평균의 초기값(랜덤 할당)을 가지고 알고리즘을 시작한다.
  - 클러스터 내 제곱합이 최소가 될 때까지 (변동이 없을 때까지) **반복**해 클러스터 평균을 구한다.
- 각 클러스터의 크기가 동일하다는 보장은 없지만 클러스터들끼리는 최대한 멀리 떨어지도록 한다.



K=5 일때 K-평균 클러스터링 예시

🤓 간단한 예제

```
# 일별 주가수익률을 네 그룹으로 나누는 문제
# 주식 수익률은 이미 표준화된 방식이므로 따로 정규화 X
from sklearn.cluster import KMeans

df = sp500_px.loc[sp500_px.index >= '2011-01-01', ['XOM', 'CVX']]
kmeans = KMeans(n_clusters=4).fit(df)
df['cluster'] = kmeans.labels_
print(df.head())
```

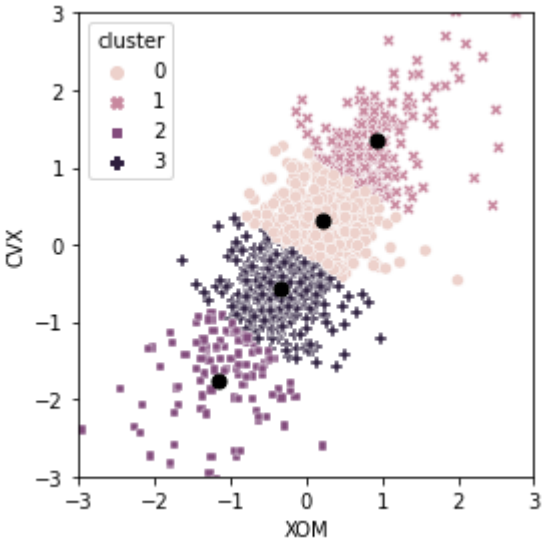
```
                XOM       CVX  cluster
2011-01-03  0.736805  0.240681        0
2011-01-04  0.168668 -0.584516        3
2011-01-05  0.026631  0.446985        0
2011-01-06  0.248558 -0.919751        3
2011-01-07  0.337329  0.180511        0
```

```
# 클러스터의 중심
centers = pd.DataFrame(kmeans.cluster_centers_, columns=['XOM', 'CVX'])
print(centers)
```

	XOM	CVX
0	0.231540	0.316965
1	0.927032	1.346412
2	-1.143980	-1.750297
3	-0.328742	-0.573470

→ 클러스터 2, 3은 하락장을 의미하고 반면에 0, 1은 상승장을 의미한다.

```
# 클러스터 및 클러스터 평균 시각화
fig, ax = plt.subplots(figsize=(4, 4))
ax = sns.scatterplot(x='XOM', y='CVX', hue='cluster', style='cluster',
                     ax=ax, data=df)
ax.set_xlim(-3, 3)
ax.set_ylim(-3, 3)
centers.plot.scatter(x='XOM', y='CVX', ax=ax, s=50, color='black')

plt.tight_layout()
plt.show()
```



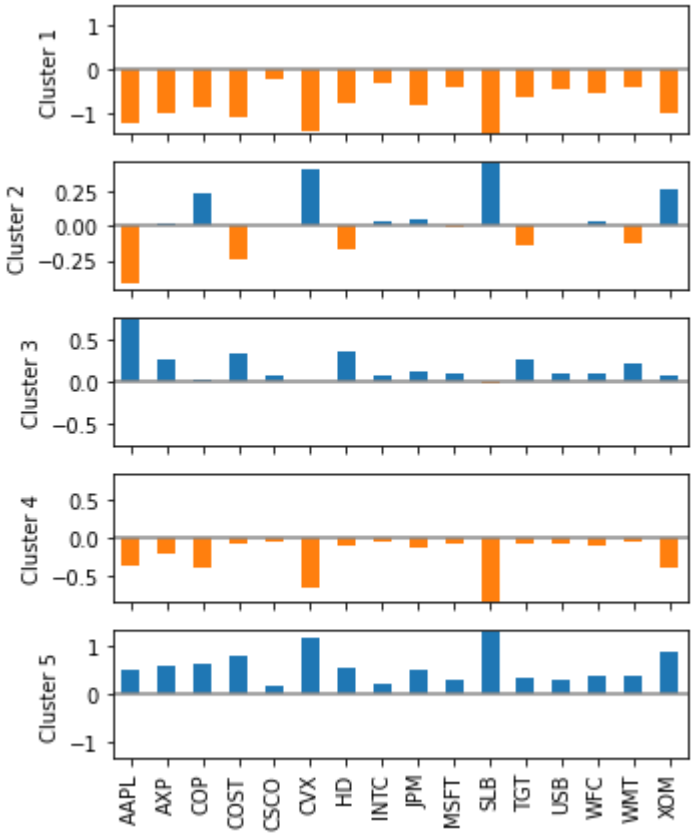
```
# K=5 클러스터링
syms = sorted(['AAPL', 'MSFT', 'CSCO', 'INTC', 'CVX', 'XOM', 'SLB', 'COP',
               'JPM', 'WFC', 'USB', 'AXP', 'WMT', 'TGT', 'HD', 'COST'])
top_sp = sp500_px.loc[sp500_px.index >= '2011-01-01', syms]
kmeans = KMeans(n_clusters=5).fit(top_sp)

# 클러스터 해석 - 크기 확인
from collections import Counter
print(Counter(kmeans.labels_))
```

```
Counter({1: 290, 2: 284, 3: 272, 4: 174, 0: 111})
```

→ 클러스터 크기가 비교적 균일하다.

→ 유난히 균형이 맞지 않는 클러스터가 존재한다면 이는 아주 멀리 떨어진 특이점(outlier)들이 있거나 아니면 어떤 레코드 그룹이 나머지 데이터로부터 아주 멀리 떨어져 있다는 것을 의미한다. (이런 경우 좀 더 자세히 들여다볼 필요가 있다.)

```
# 클러스터 해석 - 중심 확인
centers = pd.DataFrame(kmeans.cluster_centers_, columns=syms)

f, axes = plt.subplots(5, 1, figsize=(5, 6), sharex=True)
for i, ax in enumerate(axes):
    center = centers.loc[i, :]
    maxPC = 1.01 * np.max(np.max(np.abs(center)))
    colors = ['C0' if l > 0 else 'C1' for l in center]
    ax.axhline(color='#888888')
    center.plot.bar(ax=ax, color=colors)
    ax.set_ylabel(f'Cluster {i + 1}')
    ax.set_ylim(-maxPC, maxPC)

plt.tight_layout()
plt.show()
```



클러스터에서 변수들의 평균 (클러스터 평균, 즉 중심)

- 클러스터 1, 5는 각각 주식시장이 내리고 오른 날을 의미한다.
- 클러스터 4, 3은 각각 에너지 관련 주식이 내린 날과 소비재 주식이 오른 날의 특징을 보여준다.
- 클러스터 2는 에너지 주식은 오르고 소비재 주식은 내린 날을 보여준다.

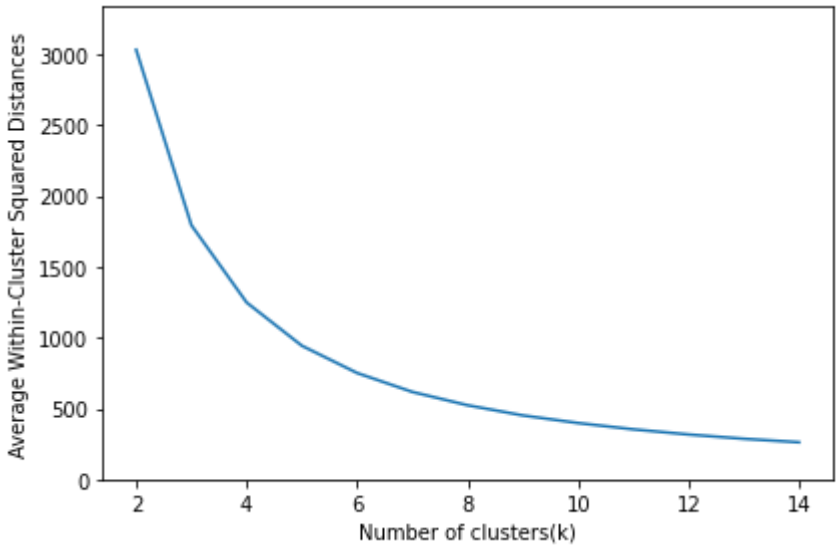
📌 PCA에서는 변동성의 주요 **방향**을 찾는 것이 목적이었다면, 클러스터링에서는 서로 가까운 위치에 있는 레코드들의 **그룹**을 찾는 것이 목적이다.

> K-평균의 K, 즉 클러스터 개수를 미리 결정하기 어려운 경우 '팔꿈치 방법(elbow method)'과 같은 방식을 사용할 수 있다. (K를 정확히 얻는 완벽한 방법은 없다!)

팔꿈치?

- 클러스터 개수에 따른 누적 분산이 가파르게 상승한 후 어느 순간 평평하게 되는 지점
- 클러스터 간 거리의 합이 급격이 떨어지는 지점

```
inertia = [] # 클러스터 간 거리의 합
for n_clusters in range(2, 15):
    kmeans = KMeans(n_clusters=n_clusters, random_state=0).fit(top_sp)
    inertia.append(kmeans.inertia_ / n_clusters)
inertias = pd.DataFrame({'n_clusters': range(2, 15), 'inertia': inertia})
ax = inertias.plot(x='n_clusters', y='inertia')
plt.xlabel('Number of clusters(k)')
plt.ylabel('Average Within-Cluster Squared Distances')
plt.ylim((0, 1.1 * inertias.inertia.max()))
ax.legend().set_visible(False)

plt.tight_layout()
plt.show()
```



팔꿈치 방법

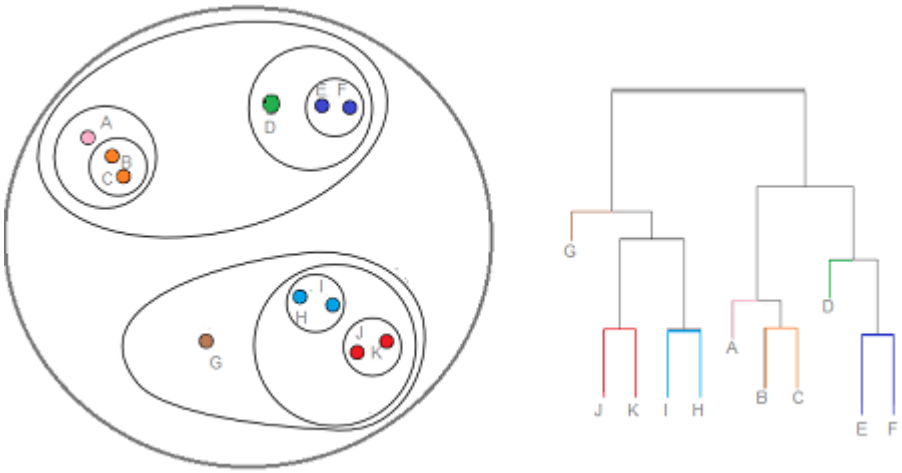
→ K=3 or 4 가 적당해 보인다.

7.3 계층적 클러스터링

Hierarchial Clustering

> 계층적 클러스터링은 K-평균 대신 사용하는 클러스터링 방법으로 상대적으로 많은 컴퓨팅 리소스가 필요할 수 있어서 대부분 데이터 크기가 작은 문제에 주로 적용된다. 직관적인 시각화가 가능해 클러스터를 해석하기가 수월하다. (ex. 덴드로그램)

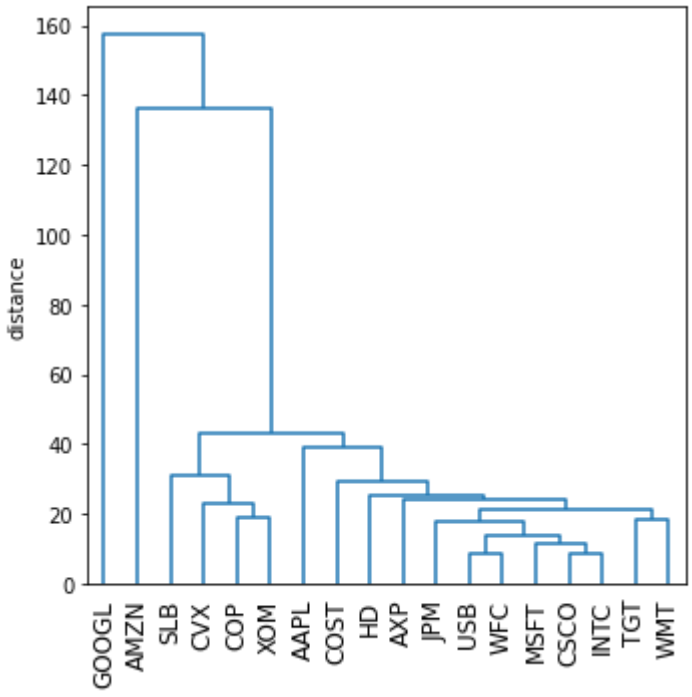
- 계층적 클러스터링은 각 레코드 자체를 개별 클러스터로 설정하여 시작하고 가장 가까운 클러스터를 **결합**해나가는 작업을 반복한다.
  - 병합(agglomerative) 알고리즘
    1. 단일 레코드로만 구성된 클러스터들로 초기 클러스터 집합을 만든다.
    2. 모든 쌍의 클러스터 간의 **비유사도**(ex. 모든 레코드 쌍의 최대 거리=완전 연결 방식)를 계산한다.
    3. 가장 가까운(비유사도가 가장 적은) 두 클러스터를 병합한다.
    4. 둘 이상의 클러스터가 남아 있으면 2단계로 다시 돌아간다. 하나 남아 있으면 알고리즘을 멈춘다.
- 계층적 클러스터링은 트리 모델과 같이 자연스러운 시각적 표현이 가능하며 이를 **덴드로그램**이라고 한다.
- K-평균 클러스터링과 달리 클러스터의 수를 미리 지정할 필요는 없다.



🤓 간단한 예제

```
# 수익이 유사한 기업들을 서로 묶는 문제
from scipy.cluster.hierarchy import linkage, dendrogram, fcluster

syms1 = ['AAPL', 'AMZN', 'AXP', 'COP', 'COST', 'CSCO', 'CVX', 'GOOGL', 'HD',
         'INTC', 'JPM', 'MSFT', 'SLB', 'TGT', 'USB', 'WFC', 'WMT', 'XOM']
df = sp500_px.loc[sp500_px.index >= '2011-01-01', syms1].transpose()
# 전치 (행을 따라 기업별 주가 정보가 오게 하고 열을 따라 날짜가 오도록)

Z = linkage(df, method='complete') # method를 'complete'로 설정해 linkage 함수를 사용
```

```
# 덴드로그램 시각화
fig, ax = plt.subplots(figsize=(5, 5))
dendrogram(Z, labels=list(df.index), color_threshold=0)
plt.xticks(rotation=90)
ax.set_ylabel('distance')

plt.tight_layout()
plt.show()
```



덴드로그램

*트리의 가지 길이는 해당 클러스터 간의 차이 정도를 나타낸다.

→ 구글(GOOGL)과 아마존(AMZN)에 대한 수익률은 서로 다르고 다른 주식에 대한 수익률과도 상당히 다른 것을 볼 수 있다.

→ 다른 주식들은 자연스럽게 그룹을 형성한다.

→ 석유 관련주들(SLB, CVX, XOM, COP)은 자신들의 클러스터가 있고 애플(AAPL)은 독자적이며 나머지는 서로 비슷하다.

```
# 원하는 개수의 클러스터를 추출
memb = fcluster(Z, 4, criterion='maxclust') # 4개의 클러스터
memb = pd.Series(memb, index=df.index)
for key, item in memb.groupby(memb):
    print(f"{key} : {', '.join(item.index)}")
```

```
1 : COP, CVX, SLB, XOM
2 : AAPL, AXP, COST, CSCO, HD, INTC, JPM, MSFT, TGT, USB, WFC, WMT
3 : AMZN
4 : GOOGL
```

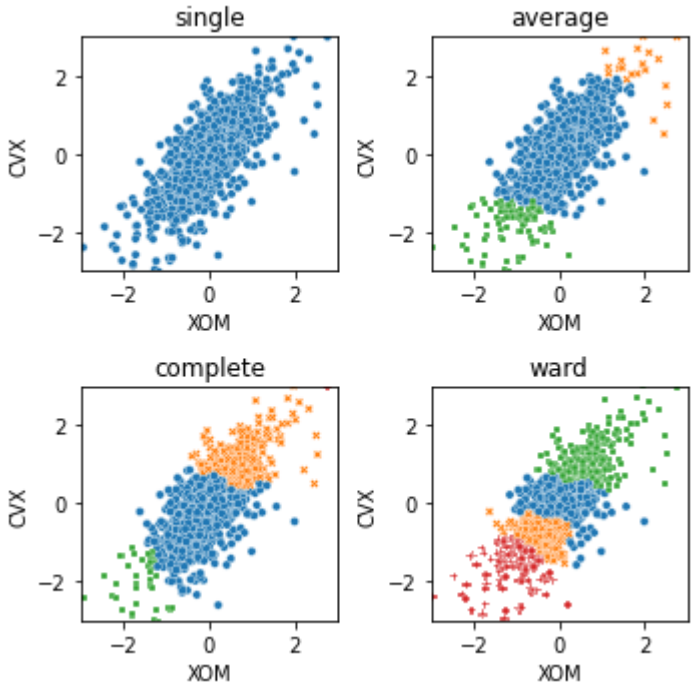
→ 구글과 아마존은 각각 독자적인 클러스터에 속해 있다. (3, 4)

→ 석유 관련 주식은 모두 또 다른 한 클러스터에 속한다. (1)

→ 나머지 주식들이 남은 하나의 클러스터에 속한다. (2)

> 계층적 클러스터링에서 클러스터 간 비유사도를 측정하는 네 가지 일반적인 지표는 완전연결, 단일연결, 평균연결, 최소분산이 있다.

```
df = sp500_px.loc[sp500_px.index >= '2011-01-01', ['XOM', 'CVX']]
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(5, 5))
for i, method in enumerate(['single', 'average', 'complete', 'ward']):
    ax = axes[i // 2, i % 2]
    Z = linkage(df, method=method)
    colors = [f'C{c+1}' for c in fcluster(Z, 4, criterion='maxclust')]
    ax = sns.scatterplot(x='XOM', y='CVX', hue=colors, style=colors,
                         size=0.5, ax=ax, data=df, legend=False)

    ax.set_xlim(-3, 3)
    ax.set_ylim(-3, 3)
    ax.set_title(method)

plt.tight_layout()
plt.show()
```



주식 데이터에 대한 비유사도 측정 지표 간 비교

- 단일연결 (single linkage) : 두 클러스터의 레코드 간 최소 거리를 사용하는 방식

  → 거의 모든 점을 하나의 클러스터에 할당했다.

- 평균연결 (average linkage) : 모든 거리 쌍의 평균을 사용하는 방식 (단일연결 + 완전연결 사이를 절충)

  → 외곽에 요소가 몇 개 없는 특이점들로 이루어진 작은 클러스터를 만들었다.

- 최소분산 (minimum variance, 워드 기법 - ward's method) : 클러스터 내의 제곱합을 최소화 (K-means 와 유사)

  → K-평균 클러스터와 가장 유사한 결과를 보인다.

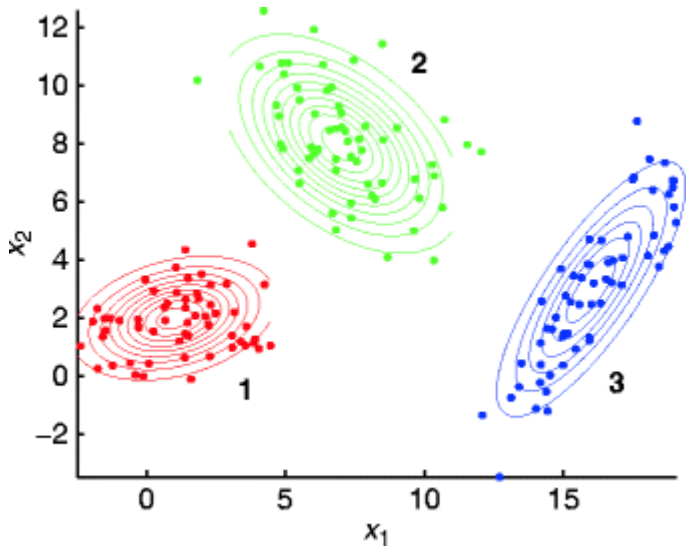
7.4 모델 기반 클러스터링

Model-based Clustering

> 계층적 클러스터링과 K-평균과 같은 *휴리스틱한 방법과 달리, 모델 기반 클러스터링은 **통계** 이론에 기초하고 있으며 클러스터의 성질과 수를 결정하는데에 더 **엄격한** 방법을 제공한다.
> (ex. 전반적으로 서로 비슷하지만 모든 데이터가 반드시 서로 가까울 필요는 없는 그룹 → 수익의 분산이 큰 기술주들)

*휴리스틱? 확률모형에 기반하지 않고 직접 관측한 데이터들이 서로 가깝게 있는 클러스터를 찾는다.

- 가장 널리 사용되는 모델 기반 클러스터링 방법은 모두 다변량정규분포(multivariate normal distribution)를 따른다.
  - 다변량정규분포 : p개의 변수 집합 X1, X2, ..., Xp 에 대해 정규분포를 일반화한 것으로 분포는 평균 집합과 공분산행렬(변수가 어떻게 상호 관련되어있는지)로 정의
- 핵심 아이디어는 각 레코드가 **K개의 다변량정규분포 중 하나로부터 발생**했다고 가정하는 것이다. 각 분포는 서로 다른 평균과 공분산행렬을 갖는다. ⇒ **정규 혼합**
- 모델 기반 클러스터링의 목적은 데이터를 가장 잘 설명하는 다변량정규분포를 찾는 것이다.

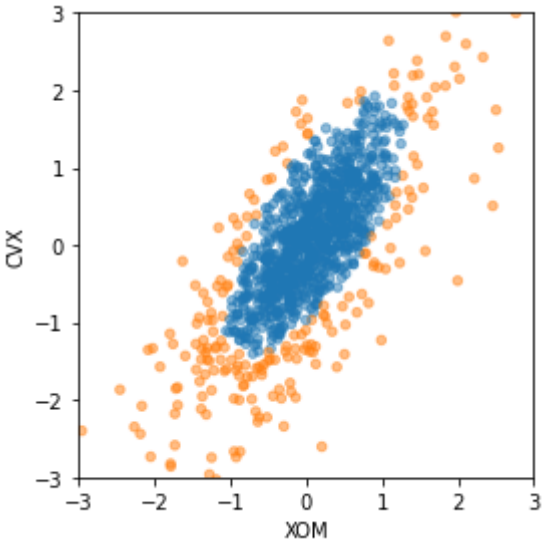


2차원 정규분포에 대한 확률 등고선 (총 3개의 다변량정규분포)

🤓 간단한 예제

```
from sklearn.mixture import GaussianMixture
# GaussianMixture : 정규 혼합

df = sp500_px.loc[sp500_px.index >= '2011-01-01', ['XOM', 'CVX']]
mclust = GaussianMixture(n_components=2).fit(df) # 다른 기법들보다 계산 시간이 오래 걸림
print(mclust.bic(df))
# 4589.8206262498725
```

```
fig, ax = plt.subplots(figsize=(4, 4))
colors = [f'C{c}' for c in mclust.predict(df)]
# predict 함수 : 할당된 클러스터 정보를 얻을 수 있다.
df.plot.scatter(x='XOM', y='CVX', c=colors, alpha=0.5, ax=ax)
ax.set_xlim(-3, 3)
ax.set_ylim(-3, 3)

plt.tight_layout()
plt.show()
```



→ 두 개의 클러스터가 있는 것을 볼 수 있다. 하나는 데이터 중심에 있고 다른 하나는 데이터 중심을 둘러싼 외곽에 존재한다.

```
# 각 정규분포의 파라미터 추출
print('Mean')
print(mclust.means_) # 평균
print('Covariances')
print(mclust.covariances_) # 공분산행렬
```

```
Mean
[[ 0.07225117  0.10452744]
 [-0.05050178 -0.21237957]]
Covariances
[[[0.26868436 0.27606914]
  [0.27606914 0.51762673]]

 [[0.97385279 0.98028909]
  [0.98028909 1.67646834]]]
```

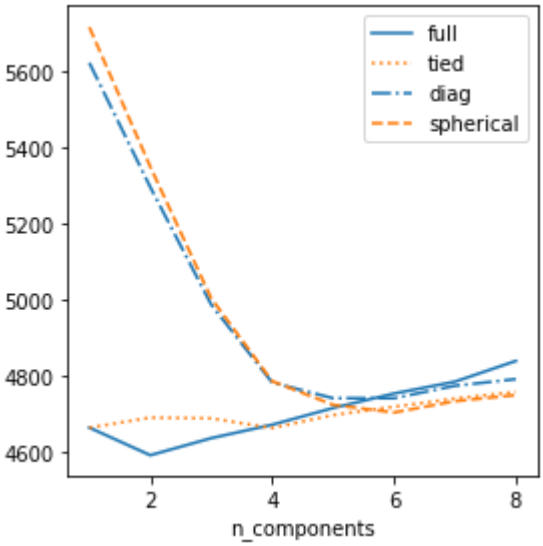
→ 두 분포는 비슷한 평균과 상관관계를 갖고 있다.

→ 두 번째 분포의 분산과 공분산이 훨씬 큰 것을 알 수 있다.

▼ *공분산 행렬 해석하기*

> K-평균이나 계층적 클러스터링과 달리 모델 기반 클러스터링은 클러스터 수를 자동으로 선택한다.
> 바로 '베이즈 정보기준 (Bayesian information criteria, **BIC**)' 값이 가장 큰 클러스터의 개수를 선택하도록 동작한다.

```
results = []
# 공분산행렬을 모수화하는 여러가지 방법
covariance_types = ['full', 'tied', 'diag', 'spherical']
for n_components in range(1, 9):
    for covariance_type in covariance_types:
        mclust = GaussianMixture(n_components = n_components, warm_start=True,
                                 covariance_type = covariance_type)
        # warm_start : 이전 피팅 정보를 재사용 -> 후속 계산의 수렴 속도 빨라짐
        mclust.fit(df)
        results.append({
            'bic': mclust.bic(df), # BIC
            'n_components': n_components,
            'covariance_type': covariance_type,
        })

results = pd.DataFrame(results)

colors = ['C0', 'C1', 'C2', 'C3']
styles = ['C0-','C1:','C0-.', 'C1--']

fig, ax = plt.subplots(figsize=(4, 4))
for i, covariance_type in enumerate(covariance_types):
    subset = results.loc[results.covariance_type == covariance_type, :]
    subset.plot(x='n_components', y='bic', ax=ax, label=covariance_type,
                kind='line', style=styles[i]) # , color=colors[i])

plt.tight_layout()
plt.show()
```



각 클러스터 크기에 대한 모델들의 BIC 값

→ 최상의 다변량정규분포를 결정하기 위해, 즉 적합한 모델을 찾기 위해 공분산행렬을 모수화하는 여러 가지 방법을 사용할 수 있다.

⇒ 이렇듯 모델 기반 클러스터링은 데이터들이 모델을 따른다는 가정이 필요하며, 계산량이 많고 대용량 데이터로 확장하기가 어렵다. 또한 알고리즘이 다른 방법들보다 더 복잡하고 이용하기가 어렵다.

*자세한 내용 : https://scikit-learn.org/stable/modules/mixture.html#gmm

7.5 스케일링과 범주형 변수

> 비지도 학습 기술을 이용할 때는 일반적으로 데이터를 적절하게 **스케일**해야 한다.

💰 예를 들어 개인 신용 대출 데이터의 변수들은 단위나 규모 면에서 서로 크게 다르다.
어떤 변수는 상대적으로 작은 값 (ex. 근속 연수), 다른 변수는 매우 큰 값 (ex. 대출 금액) 을 갖는다.

→ 데이터의 크기가 조정되지 않으면 PCA, K-평균, 기타 클러스터링 방법은 큰 값을 갖는 변수들에 의해 결과가 좌우되고 작은 값을 갖는 변수들은 무시된다!

🤓 간단한 예제

```
# 대출 데이터를 정규화하지 않고 바로 kmeans 함수를 적용했을 경우

loan_data = pd.read_csv(LOAN_DATA_CSV)
loan_data['outcome'] = pd.Categorical(loan_data['outcome'],
                                      categories=['paid off', 'default'],
                                      ordered=True)
defaults = loan_data.loc[loan_data['outcome'] == 'default',]

columns = ['loan_amnt', 'annual_inc', 'revol_bal', 'open_acc',
           'dti', 'revol_util']

df = defaults[columns]
kmeans = KMeans(n_clusters=4, random_state=1).fit(df)
counts = Counter(kmeans.labels_)

centers = pd.DataFrame(kmeans.cluster_centers_, columns=columns)
```

```
centers['size'] = [counts[i] for i in range(4)]
print(centers)
```

```
      loan_amnt     annual_inc     revol_bal   open_acc        dti  \
0  18275.132345   83354.634595  19635.189254  11.664373  16.774586
1  21852.701005  165407.730318  38907.295645  12.597152  13.466876
2  10591.893792   42453.058692  10268.048598   9.583820  17.713563
3  22570.192308  489783.403846  85161.346154  13.326923   6.907500

   revol_util   size
0   62.258588   7543
1   63.634900   1194
2   58.111226  13882
3   59.651923     52
```

클러스터 0, 1, 2, 3과 각 클러스터 중심 값

→ annual_inc 과 revol_bal 이 클러스터링 결과를 좌우하며 클러스터마다 크기가 매우 다른 것을 볼 수 있다.

→ 클러스터 3에는 비교적 높은 소득(annual_inc)과 높은 회전 신용 잔고(revol_bal)를 가진 단 52명의 데이터만 포함되어 있다.

```
# 정규화된 데이터에 kmeans 를 적용했을 경우

from sklearn import preprocessing

scaler = preprocessing.StandardScaler() # 표준화 (정규화) -> 평균을 빼고 표준편차로 나눔
df0 = scaler.fit_transform(df * 1.0)

kmeans = KMeans(n_clusters=4, random_state=1).fit(df0)
counts = Counter(kmeans.labels_)

centers = pd.DataFrame(scaler.inverse_transform(kmeans.cluster_centers_),
                       columns=columns) # 원래 단위로 재조정 -> 해석이 용이
centers['size'] = [counts[i] for i in range(4)]
print(centers)
```

```
      loan_amnt     annual_inc     revol_bal   open_acc        dti  \
0  10499.824632   51070.958451  11629.172535   7.511129  15.965747
1  10315.255666   53468.181307   6032.616033   8.637385  11.255855
2  25920.260952  116308.326663  32827.641428  12.389941  16.204021
3  13420.700048   55844.852918  16370.832021  14.334512  24.189881

   revol_util  size
0   77.806693  7405
1   31.000342  5339
2   66.172004  3701
3   59.227862  6226
```

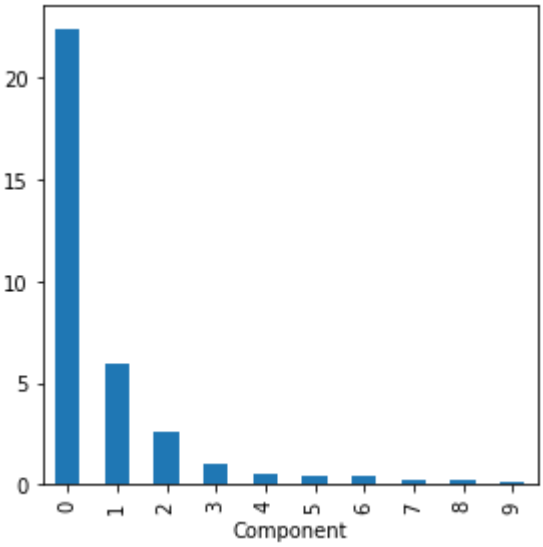
클러스터 0, 1, 2, 3과 각 클러스터 중심 값

→ 클러스터들의 크기가 좀 더 균일하며 앞선 결과와 달리 annual_inc 과 revol_bal 에 의해 큰 영향을 받지 않는다.

> 변수들이 서로 동일한 규모로 측정되고 상대적 중요성을 정확하게 반영하는 경우(ex. 주가 변동)조차도 변수의 스케일을 재조정하는 것이 유용할 수 있다.

	0	1
GOOGL	-0.857310	0.477873
AMZN	-0.444728	-0.874149
AAPL	-0.071627	-0.020802
MSFT	-0.036002	-0.006204
CSCO	-0.029205	-0.003045
INTC	-0.026666	-0.006069
CVX	-0.089548	-0.037420
XOM	-0.080336	-0.020511
SLB	-0.110218	-0.030356
COP	-0.057739	-0.024117
JPM	-0.071228	-0.009244
WFC	-0.053228	-0.008597
USB	-0.041670	-0.005952
AXP	-0.078907	-0.024027
WMT	-0.040346	-0.007141
TGT	-0.063659	-0.024662
HD	-0.051412	-0.032922
COST	-0.071403	-0.033826

PCA 결과 : GOOGL 과 AMZN 세상



스크리 그래프

→ 처음 두 가지 주성분은 GOOGL과 AMZN에 의해 거의 완전히 지배되고 있다. 이들의 주가 움직임이 전체 변동성의 대부분을 지배하기 때문이다.

⇒ 이를 해결하기 위해서는 변수를 스케일링해서 포함하거나, 지배 변수를 전체 분석에서 제외하고 별도로 처리할 수도 있다. 어떤 방법이 항상 옳다고는 할 수 없으며 응용 분야에 따라 달라진다.

> 데이터를 구성하는 변수들에 **연속형과 이진형 변수가 섞여 있는 경우**에는 비슷한 스케일이 되도록 변수의 크기를 조정해야 한다. 대표적인 방법은 고워 거리(Gower's Distance)를 사용하는 것이다.

	dti	payment_inc_ratio	home_	purpose_
0	1.00	2.39320	RENT	major_purchase
1	5.55	4.57170	OWN	small_business
2	18.08	9.71600	RENT	other
3	10.08	12.21520	RENT	debt_consolidation
4	7.06	3.90888	RENT	other

- 고워 거리의 기본 아이디어는 데이터 유형에 따라 **거리 지표**를 다르게 적용하는 것이다.
  - 수치형 변수나 순서형 요소에서 두 레코드 간의 거리는 차이의 절댓값(맨해튼 거리)으로 계산한다.
  - 범주형 변수의 경우 두 레코드 사이의 범주가 서로 다르면 거리는 1, 범주가 동일하면 0이다.

- 계산 방법
  1. 각 레코드의 변수 i와 j의 모든 쌍에 대해 거리 d를 계산
  2. 각 d의 범위가 0~1이 되도록 스케일 조정
  3. 거리 행렬을 구하기 위해 변수 간 스케일된 거리를 모두 더한 후 평균 혹은 가중 평균 계산

🤓 혼합 데이터의 클러스터링 문제

- K-평균과 PCA는 연속형 변수에 가장 적합하다.
- 실무에서는 이진형 데이터와 함께 사용하는 것이 어려울 수 있다.

  → 스케일링에 표준 z 점수를 사용할 경우 이진형 변수가 클러스터 결과에 지대한 영향을 미친다!

  → 0 또는 1의 값인 레코드가 모두 한 클러스터에 포함되므로 K-평균에서 클러스터 내 제곱합이 작아지기 때문이다.

```
columns = ['dti', 'payment_inc_ratio', 'home_', 'pub_rec_zero']
df = pd.get_dummies(defaults[columns])

scaler = preprocessing.StandardScaler()

df0 = scaler.fit_transform(df * 1.0)
kmeans = KMeans(n_clusters=4, random_state=1).fit(df0)
centers = pd.DataFrame(scaler.inverse_transform(kmeans.cluster_centers_),
                       columns=df.columns)
print(centers)
```

```
         dti  payment_inc_ratio  pub_rec_zero  home__MORTGAGE     home__OWN  \
0  16.992128           9.105395      1.000000    1.176836e-14 -1.346145e-15
1  17.456244           8.422914      1.000000    1.000000e+00 -1.193490e-15
2  16.504955           8.064247      0.000000    5.156600e-01  1.110223e-16
3  17.197993           9.266666      0.917903   -6.661338e-16  1.000000e+00

     home__RENT
0  1.000000e+00
1 -2.275957e-15
2  4.843400e-01
3  1.887379e-15
```

클러스터 0, 1, 2, 3과 각 클러스터 중심 값

→ 4개의 클러스터가 요인변수들과 밀접한 관계가 있음을 볼 수 있다. (???)

⇒ 이러한 문제를 해결하기 위해 이진형 변수의 크기를 다른 변수들보다 작은 값으로 조정하거나, 데이터의 크기가 아주 큰 경우에는 특정 범주 값들에 따라 서로 다른 하위 집합에 클러스터링을 적용할 수도 있다.

ex. 주택 담보 대출이 있는 사람인지, 주택을 완전히 소유한 사람인지, 주택을 임대하는 사람인지에 따라 대출 데이터를 쪼개서 각각 하위 그룹에 클러스터링을 개별 적용

7.6 마치며

✔️ `주성분분석`과 `K-평균 클러스터링`은 수치형 데이터의 **차원을 축소**하기 위해 주로 사용되는 방법이다.

✔️ 의미 있는 데이터 축소를 보장하기 위해서는 데이터의 **스케일을 적절히 조정**해야 한다.

✔️ 클러스터링 방법 비교

- K-평균 클러스터링

  : 매우 큰 데이터로 확장이 가능하고 이해하기 쉽다.

- 계층적 클러스터링

  : 수치형과 범주형이 혼합된 데이터 유형에 적용이 가능하며 직관적인 시각화 방법(덴드로그램)이 존재한다.

- 모델 기반 클러스터링

  : 휴리스틱한 방법들과 달리 통계 이론에 기초를 두고 있으며, 더 엄밀한 접근 방식을 제시한다. 그러나, 데이터가 커지면 K-평균이 가장 많이 사용되는 방법이다.

→ 궁극적으로 데이터 크기나 응용 분야의 목표에 따라 사용되는 방법은 달라지게 된다. 어떤 솔루션을 사용하느냐는 데이터 사이언티스트의 선택!