Chapter 07. 비지도 학습 - 진우

> **비지도학습(Unsupervised Learning)**
> 레이블이 달린 데이터를 이용해 모델을 학습하는 과정없이 데이터로부터 의미를 이끌어내는 통계적 기법

※ 비지도 학습의 활용사례

1. 클러스터링 (Clustering) : 데이터의 의미있는 그룹들을 찾을 때

예시) 웹사이트에서 사용자의 클릭 데이터와 인구통계정보를 이용해 서로 다른 성격의 사용자를 그룹화
→ 이를 통해 웹사이트를 사용자 그룹의 기호에 맞게 개선할 수 있음

▼ 콜드스타트 (cold-start) 문제에서 유용

새로운 마케팅 홍보를 론칭하거나 잠재적인 새로운 형태의 사기나 스팸을 걸러내는 유형의 문제에서는 모델을 훈련시킬 수 있는 **응답 데이터를 초기에 갖고 있지 않다**. 시간이 지나고 데이터가 쌓이면 시스템에 대해 좀 더 알 게 되고 있는 전형적인 예측 모델을 학습할 수 있다. **이럴때 클러스터링은 패턴이 비슷한 데이터들을 분류하여 학습과정을 더 빨리 시작할 수 있도록 도와준다**.

2. 차원축소 (Reducing the dimension) : 데이터의 변수들을 관리할 수 있을 만한 수준으로 줄임

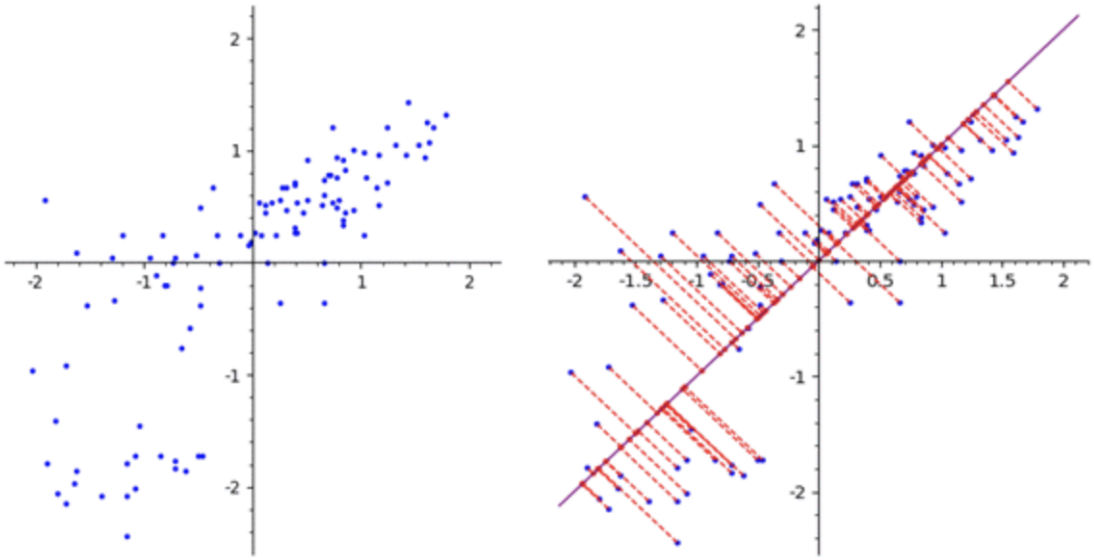
예시) 제조 공정 모니터링 하기 위해 수천 개의 센서를 사용한다고 가정
공정 실패 예측하는 모델을 만들고자 할 때, 전체 데이터의 차원을 훨씬 작은 차원의 의미 있는
피처로 데이터로 줄일 수 있다면 수천 개의 센서에서 나오는 데이터를 전부 포함하는 것 보다 강력
하면서도 쉬운 모델을 만들 수 있음

3. 데이터와 다른 변수 들 사이에 서로 어떤 관계가 있는지에 대한 통찰을 얻는 것이 목적

💡 예시) 변수와 레코드 수가 아주 큰 상황(EDA 연장)
→ 변수들을 정밀하게 조사하고 밝혀내는데 유용한 방법 제시

▼ 7.1 주성분 분석 (PCA)

> **PCA (Principal Components Analysis)**
> 다수의 수치형 예측 변수들을 더 적은 수의 변수들의 집합으로 나타내는 것
> 전체 변수들의 특징을 대부분 설명할 수 있는 적은 수의 변수들의 집합을 **주성분**이라고 함



PCA를 통해 고차원의 데이터를 저차원으로 축소

💡 주성분 분석 (참고자료)
주성분 분석(PCA)은 가장 널리 사용되는 차원 축소 기법 중 하나로, 원 데이터의 분포를 최대한 보존하면서 **고차원 공간의 데이터들을 저차원 공간으로 변환**한다.
PCA는 기존의 변수를 조합하여 서로 연관성이 없는 새로운 변수, 즉 주성분(principal component, PC)들을 만들어 낸다. **첫 번째 주성분 PC1이 원 데이터의 분포를 가장 많이 보존**하고, **두 번째 주성분 PC2가 그 다음으로 원 데이터의 분포를 많이 보존하는 식**이다. 앞서 언급한 11차원의 데이터의 경우 기존의 변수들을 조합하여 같은 개수(11개)의 주성분을 만들 수 있는데, **만일 PC1, PC2, PC3가 원 데이터의 분포(성질)의 약 90%를 보존한다면, 10% 정도의 정보는 잃어버리더라도, 합리적인 분석에 큰 무리가 없으므로, PC1, PC2, PC3만 택하여 3차원 데이터로 차원을 줄일 수 있다.**

※ 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

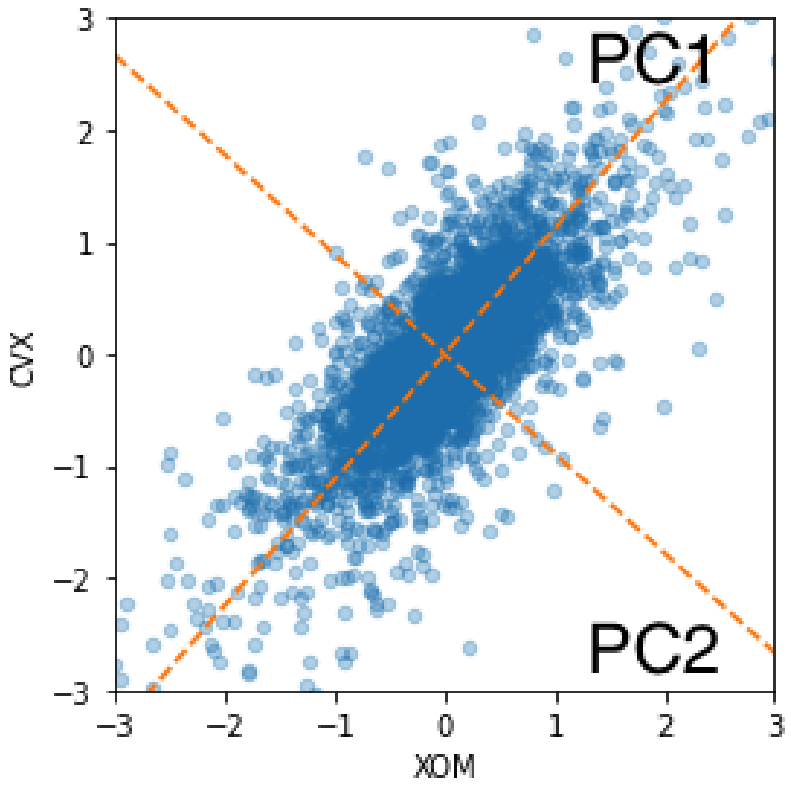
엑슨모빌(XOM), 셰브런(CVS)의 주가 데이터

```
# 주성분 분석
pcs = PCA(n_components=2)
pcs.fit(oil_px)
loadings = pd.DataFrame(pcs.components_, columns=oil_px.columns)
print(loadings)
```

종목	XOM	CVM
PC1	-0.665	-0.747
PC2	0.747	-0.665

▼ 주성분 분석 시각화 코드

```
def abline(slope, intercept, ax):
    """Calculate coordinates of a line based on slope and intercept"""
    x_vals = np.array(ax.get_xlim())
    return (x_vals, intercept + slope * x_vals)

ax = oil_px.plot.scatter(x='XOM', y='CVX', alpha=0.3, figsize=(4, 4))
ax.set_xlim(-3, 3)
ax.set_ylim(-3, 3)
ax.plot(*abline(loadings.loc[0, 'CVX'] / loadings.loc[0, 'XOM'], 0, ax),
        '--', color='C1')
ax.plot(*abline(loadings.loc[1, 'CVX'] / loadings.loc[1, 'XOM'], 0, ax),
        '--', color='C1')

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



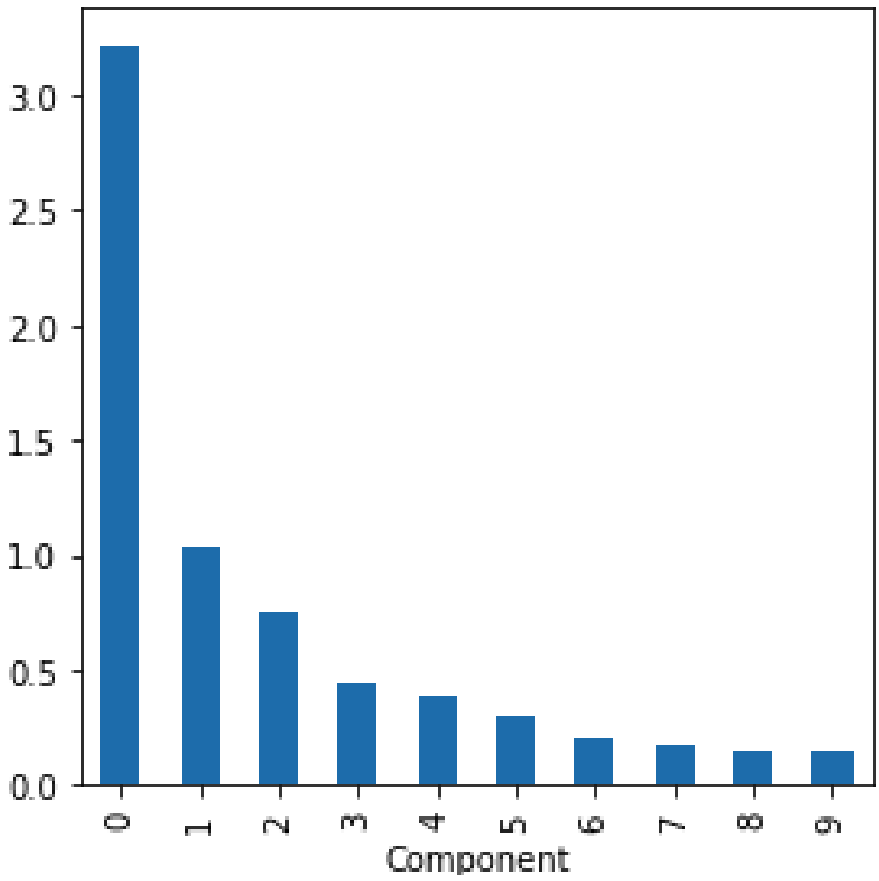
주성분 분석 시각화 (엑슨모빌과 셰브런의 주가 수익에 대한 주성분)

- PC1(타원의 장축) : 두 회사 사이의 상관관계를 반영하는 XOM과 CVM의 평균을 의미
- PC2(타원의 단축) : XOM과 CVM의 주가가 달라지는 지점을 반영

※ **스크리그래프 (Scree Graph)** : **주성분의** 이해를 돕기 위해 사용되는 것, **상대적인 중요도**를 표시해 줌

(그림이 절벽이 있는 산비탈 모양과 흡사하다고 해서 붙여짐)

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

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



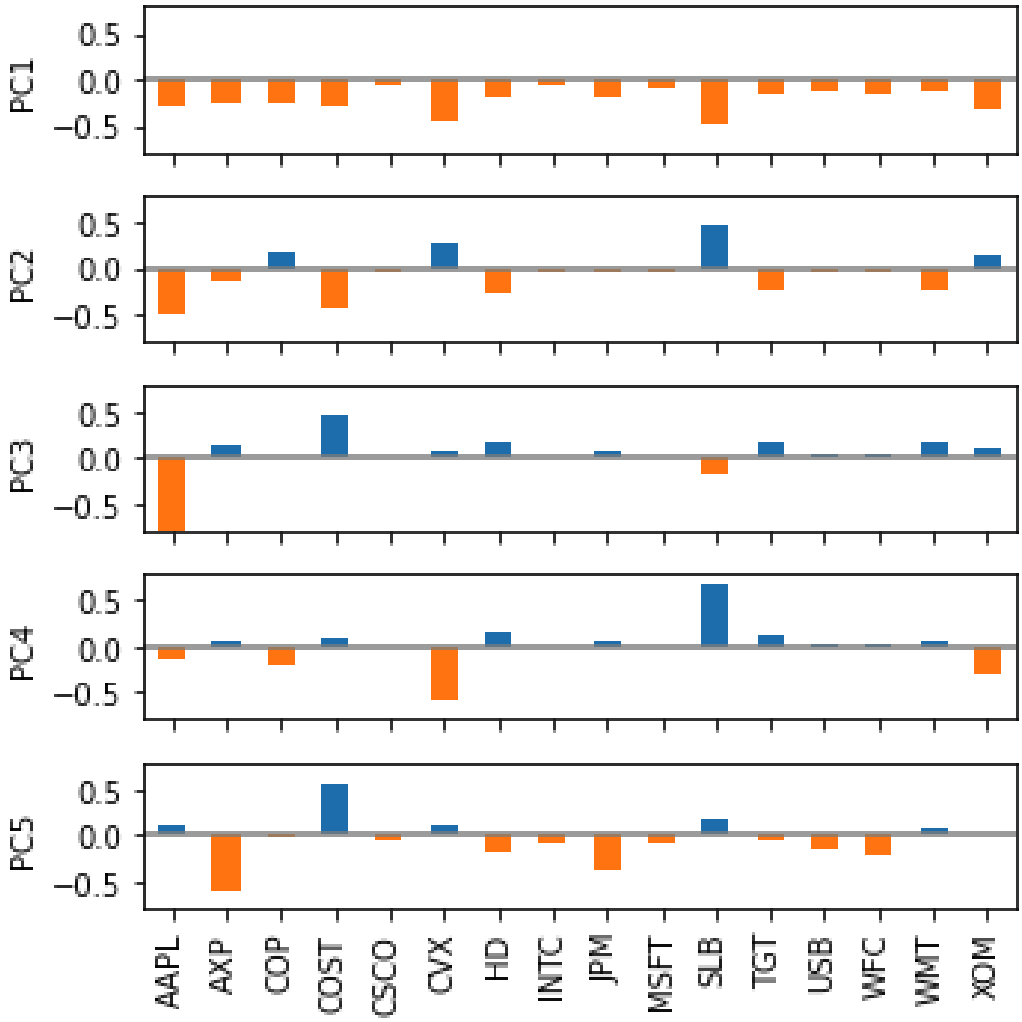
S&P 500 상위권 주가에 대한 PCA의 Scree plot

- 상위 주성분들의 가중치를 표시해 보는 것도 주성분을 이해하는데 도움이 됨

```
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()
```



- PC1 : 모든 변수로 부터 비슷한 정도의 영향을 공유한다는 것을 알 수 있음
- PC2 : 에너지 관련 주식과 다른주식의 가격 변동을 잡아냄
- PC3 : 애플(AAPL), 코스트코(COST)의 움직임이 서로 반대라는 사실을 알려줌
- PC4 : 슬륨베르거(SLB)와 나머지 에너지 회사들의 움직임이 반대라는 것을 보여줌
- PC5 : 금융회사들이 주를 이루고 있다는 것을 보여줌

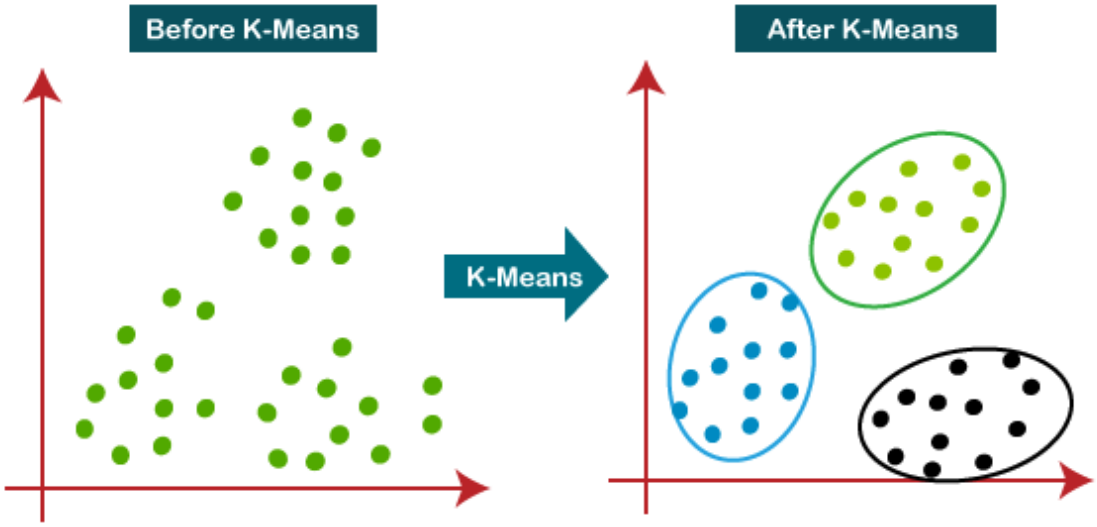
💡 성분을 몇 개까지 골라야 할까?
1. 가장 일반적 방법 : 대부분의 변동성을 설명할 수 있는 성분들을 고르기 위한 특별한 규칙을 사용
ex) 스크리 플롯 활용, 상위 N개 주성분 분석 제한, 누적 분산 임계치 이상 성분 등
2. 직관적인 해석이 가능한 성분들 알아보기 위해 가중치 참고
교차타당성검사를 통해 중요한 주성분들의 개수를 결정하기 위한 공식적인 방법

▼ 7.2 K-평균 클러스터링

> **K-Means Clustering**
> 클러스터링(군집화)은 데이터의 유의미한 그룹들을 구분하는 것으로 각 그룹에 비슷한 데이터들이 속함
> k-means는 최초로 개발된 클러스터링 기법으로 알고리즘이 상대적으로 간단하고 데이터 크기가 커져도 손쉽게 사용할 수 있다는 점에서 널리 사용 중

- k-means는 데이터 k개를 클러스터로 나눔
- 이때 할당된 클러스터 평균과 포함된 데이터들의 거리 제곱합이 최소가 되도록 함

💡 정규화
데이터 값에서 평균을 빼고 그 편차를 표준편차로 나눠주는 방법이 가장 일반적
이렇지 않으면 스케일이 가장 큰 변수가 클러스터링 독점하게 됨!



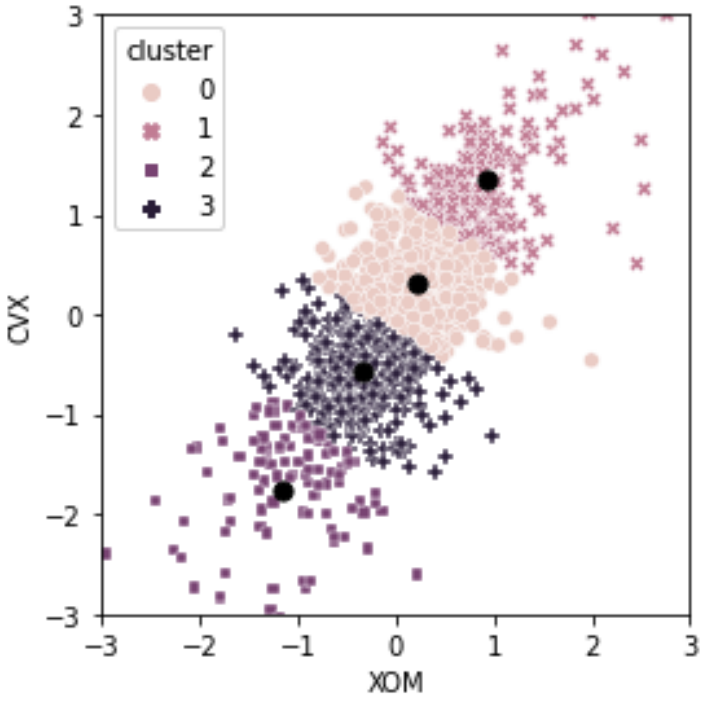
※ 예제

```
# 엑슨모빌, 셰브런 일변 주가 수익률을 변수고 놓고 4개 클러스터 분류
# 주식 수익률은 이미 표준화된 방식으로 보고되므로 정규화 필요 없음
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())

# 클러스터 중심
centers = pd.DataFrame(kmeans.cluster_centers_, columns=['XOM', 'CVX'])
print(centers)
# 클러스터 0,1은 상승장, 2,3은 하락장 의미
```

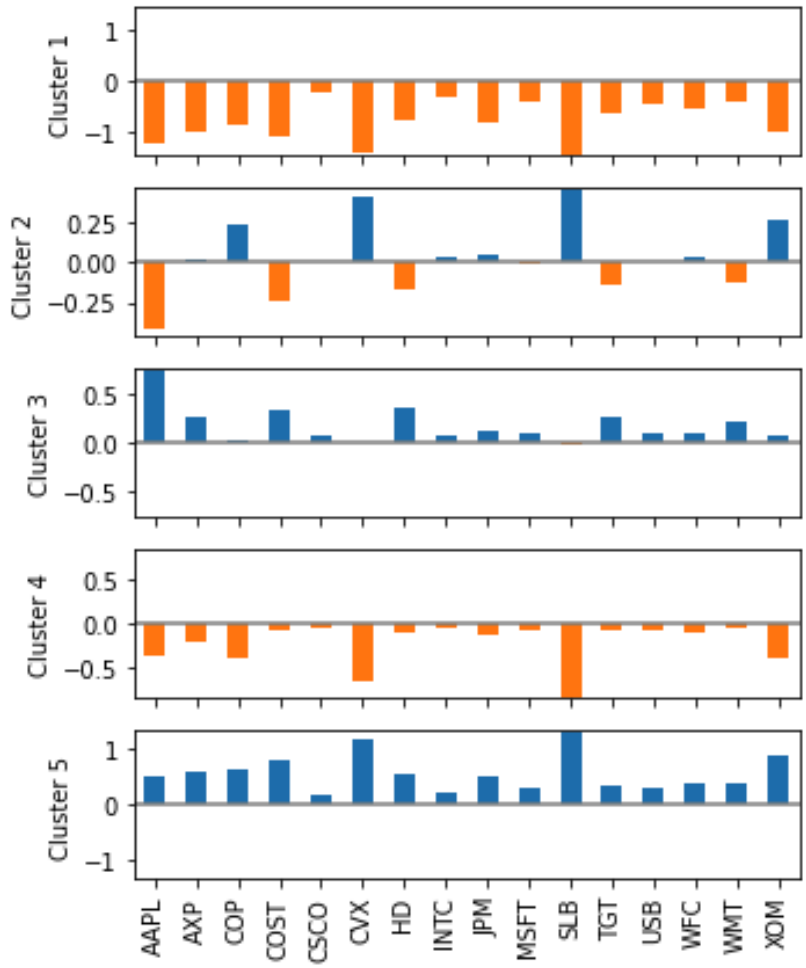
	XOM	CVX	cluster
2011-01-03	0.736805	0.240681	0
2011-01-04	0.168668	-0.584516	3
2011-01-05	0.026631	0.446985	0
2011-01-06	0.248558	-0.919751	3
2011-01-07	0.337329	0.180511	0

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

```
# k-means clustering 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()
```




```
# 클러스터 해석
# PCA와 유사
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()
```



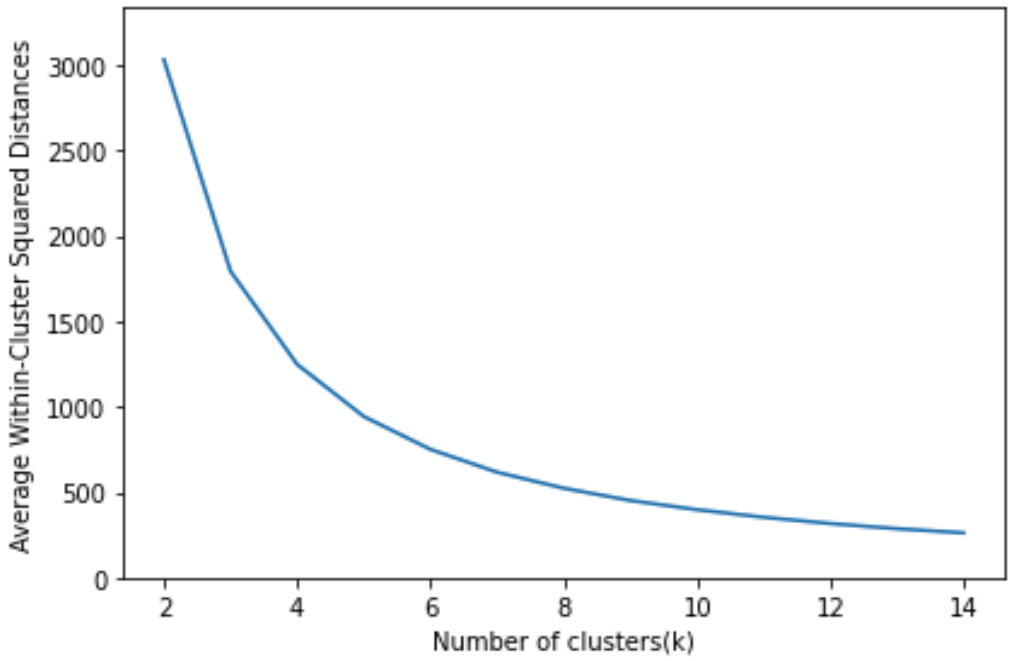
- Cluster 1, 5 : 주식시장이 내리고 오른날
- Cluster 2 : 에너지 주식은 오르고 소비재 주식은 내린날
- Cluster 3, 4 : 에너지 주식이 내린날과 소비재 주식이 오른날

※ 클러스터 개수 선정 (elbow method)

> 언제 클러스터 세트가 데이터 분산의 '대부분'을 설명하는지 알려주는 방법

```
inertia = []
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()
```



💡 주요 개념
- 사용자가 원하는 클러스터 k개를 선택
- k-means 알고리즘은 클러스터가 더는 변하지 않을 때까지 반복해서 클러스터 평균이 가장 가까운 클러스터에 할당
- 실무적인 상황을 고려해 k개를 선택하는 것이 가장 일반적, 통계적으로 최적의 수를 구하는 방법은 없다.

▼ 7.3 계층적 클러스터링

> **계층적 클러스터링 (Hierarchical Clustering)** : k-means와는 아주 다른 결과
> 서로 다른 수의 클러스터를 지정하는 효과를 시각화

> 특이점이나 비정상적인 그룹이나 레코드를 발견하는 데 더 민감함
> 직관적인 시각화가 가증하여 클러스터를 해석하기 수월

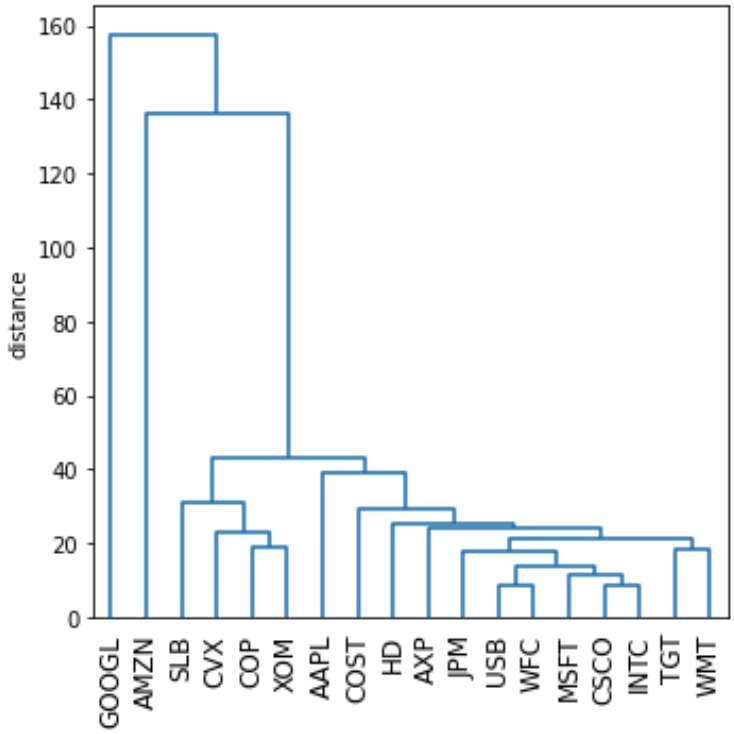
- 덴드로그램(dendrogram) : 레코드들이 속한 계층적 클러스터를 시각적으로 표현
- 거리(distance) : 한 레코드가 다른 레코드들과 얼마나 가까운지 보여주는 측정 지표
- 비유사도(dissimilarity) : 한 클러스터가 다른 클러스터들과 얼마나 가까운지 보여주는 측정 지표

※ 예제

```
# 예제 샘플
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()

Z = linkage(df, method='complete')
print(Z.shape)
```

▼ 덴드로그램

```
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()
```



- 구글, 아마존에 대한 수익률은 다르고 다른 주식에 대한 수익률과 상당히 다름
- 다른주식들은 자연스럽게 그룹 형성
- 애플(AAPL)은 독자적, 석유 관련주(SLB, CVX, XOM, COP)그룹, 나머지는 서로 비슷함

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

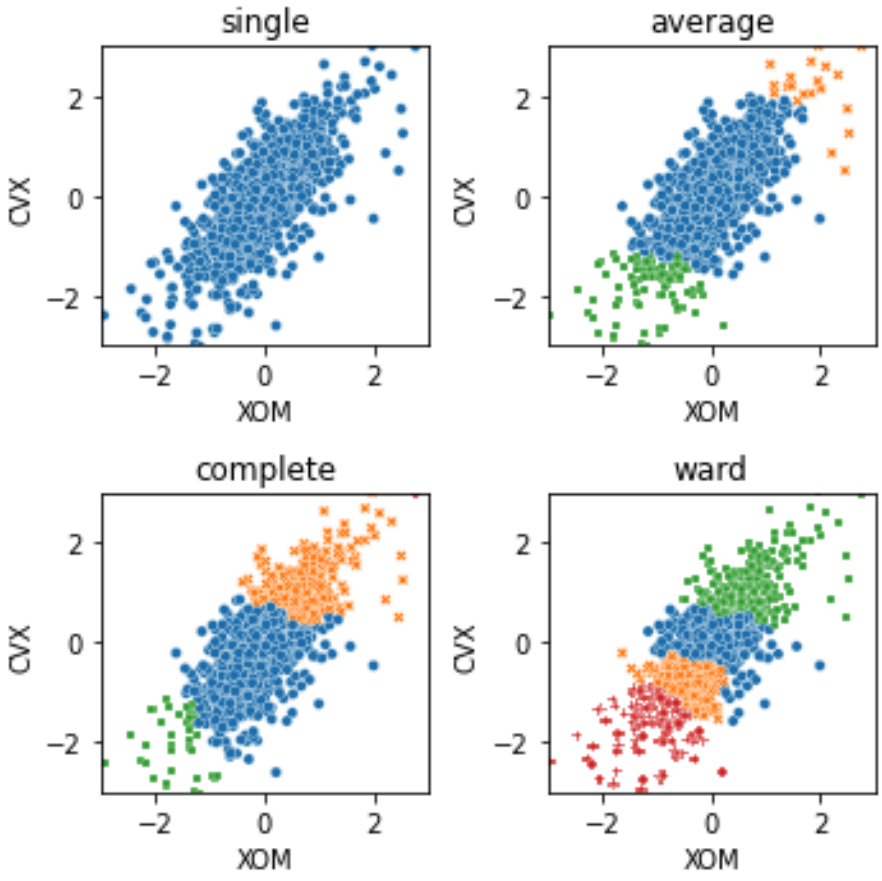
```
1 : COP, CVX, SLB, XOM
2 : AAPL, AXP, COST, CSCO, HD, INTC, JPM, MSFT, TGT, USB, WFC, WMT
3 : AMZN
4 : GOOGL
```

▼ 비유사도 측정

- 완전연결 : 비슷한 맴버가 있는 클러스터를 만드는 경향
- 단일연결 : 두 클러스터의 레코드간 최소 거리를 사용하는 방식
    - 탐욕적 방법으로 결과로 나온 클러스터는 서로 크게 다른 요소들을 포함하는 일도 발생

- 평균연결 : 모든 거리 쌍의 평균을 사용하는 방법으로 단일연결과 완전연결을 절충한 방법
- 워드기법(최소분산) : 클러스터 내의 제곱합을 최소화 하므로 k-means와 유사

```
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()
```



▼ 7.4 모델 기반 클러스터링

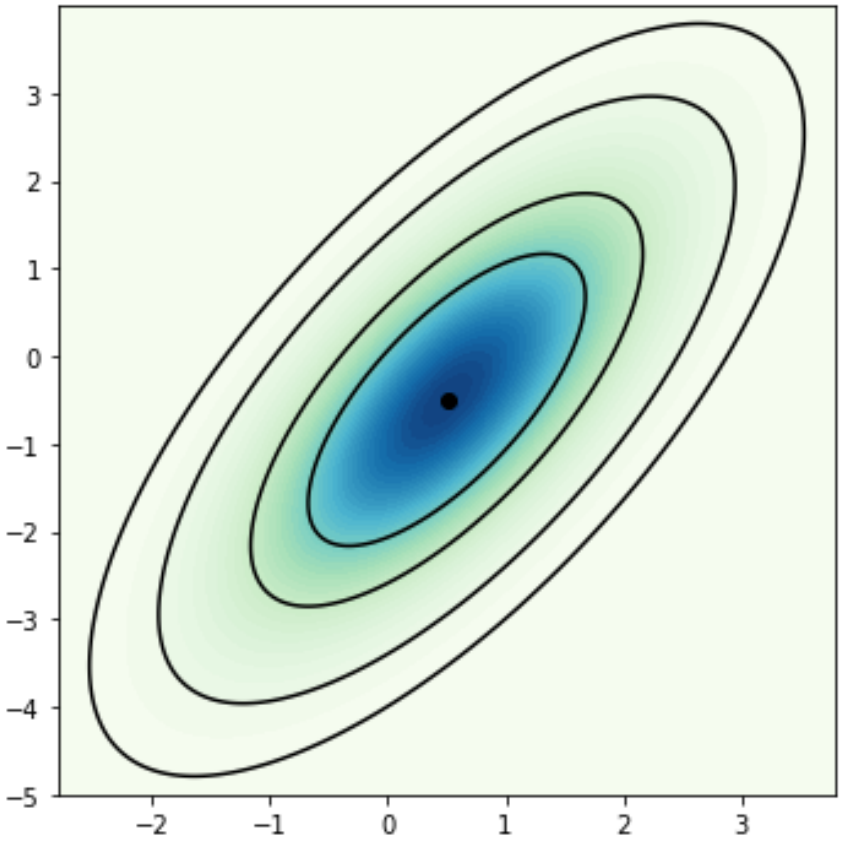
> k-means, 계층적 클러스터링은 **휴리스틱**한 방법
> 모델 기반 클러스터링은 **통계 이론에 기초하고 클러스터의 성질과 수를 결정하는데 엄격한 방법은 제공**

💡 **휴리스틱**(heuristics) 또는 **발견법**(發見法)이란 불충분한 시간이나 정보로 인하여 합리적인 판단을 할 수 없거나, 체계적이면서 합리적인 판단이 굳이 필요하지 않은 상황에서 사람들이 빠르게 사용할 수 있게 보다 용이하게 구성된 간편추론의 방법이다.

- 대부분 모델 기반 클러스터링 방법은 모두 **다변량 정규분포를 따름**
- 다변량 정규 분포는 p개의 변수 집합 $X_1, X_2, ...X_p$에 대해 정규분포를 일반화 한 것
- 모델 기반 클러스터링의 핵심 아이디어는 각 레코드가 k개의 다변량정규분포 중 하나로부터 발생했다고 가정하는 것, 각 분포는 서로다른 평균과 공분산행렬을 갖는다. **(정규혼합)**

※ 예제

```
# 다변량 정규 분포 예시
mean = [0.5, -0.5]
cov = [[1, 1], [1, 2]]
probability = [.5, .75, .95, .99]
def probLevel(p):
    D = 1
    return (1 - p) / (2 * math.pi * D)
levels = [probLevel(p) for p in probability]

fig, ax = plt.subplots(figsize=(5, 5))

x, y = np.mgrid[-2.8:3.8:.01, -5:4:.01]
pos = np.empty(x.shape + (2,))
pos[:, :, 0] = x; pos[:, :, 1] = y
rv = multivariate_normal(mean, cov)

CS = ax.contourf(x, y, rv.pdf(pos), cmap=cm.GnBu, levels=50)
ax.contour(CS, levels=levels, colors=['black'])
ax.plot(*mean, color='black', marker='o')

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



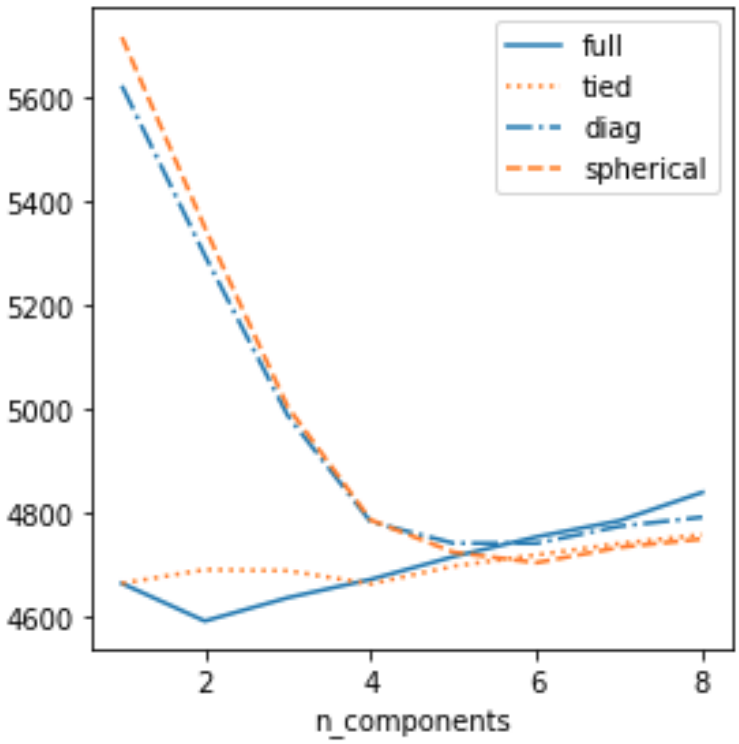
모델 기반 클러스터링의 목적은 **데이터를 가장 잘 설명하는 다변량 정규분포를 찾는 것**, 그림과 같은 등고선을 보면 **정규분포 모양을 갖고 있는 것 처럼 보임**

※ 클러스터 개수 결정

> k-means나 계층적 클러스터링과 달리 **모델 기반 클러스터링은 클러스터 수를 자동으로 선택**
> 베이즈 정보기준(BIC) 값이 가장 큰 클러스터의 개수를 선택하도록 동작하기 때문!

```
results = []
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)
        mclust.fit(df)
        results.append({
            'bic': mclust.bic(df),
            '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()
```



- 최상의 다변량 정규분포를 결정하기 위해서는 공분산행렬을 모수화하는 여러가지 방법들이 있기 때문에 여러 모델을 사용한다.
- 위 그림에서 BIC에 따르면 full, spherical 두가지 모델이 4가지 구성요소를 사용할 때 가장 적합??
- 하지만 모델 기반 클러스터링 기술에는 몇가지 한계점이 있음
    - 데이터들이 모델을 따른다는 가정이 필요하며 결과는 이 가정에 따라 매우 다름
    - 필요한 계산량 역시 계층적 클러스터링보다 높으므로 대용량 데이터로 확장하기 어려움

- 알고리즘이 다른 방법들보다 더 복잡하고 이용하기 어려움

▼ 7.5 스케일링과 범주형 변수

> 비지도 학습 기술을 이용할 때는 일반적으로 데이터를 적절하게 스케일 해야함

💡 스케일링 : 데이터의 범위를 늘리거나 줄이는 방식으로 여러변수들이 같은 스케일에 나오게 하는 것
정규화 : 원래 변수 값에서 평균을 뺀 후에 표준편차로 나누는 방법(유의어 : 표준화)
고워 거리 : 수치형과 범주형 데이터가 섞여 있을 경우 모든 변수가 0~1 사이로 나오도록 하는 것

- 예를 들어 어떤 변수는 상대적으로 작은 값인 반면 다른 변수는 매우 큰 값을 갖게 된다면 PCA, k-means 혹은 기타 클러스터링 방법은 큰 값을 갖는 변수들에 좌우되고 작은 값을 갖는 변수는 무시됨
- 범주형 데이터는 일반적으로 원-핫인코딩을 사용하여 이진 변수 집합으로 변환하는데 이러한 이진변수는 다른 데이터와 스케일이 다를 뿐만 아니라 PCA, k-means와 같은 기법을 사용할 때 이진 변수가 두 가지의 값만 가질 수 있다는 것 때문에 문제가 될 수 있음

※ 예제1(정규화)

```
# 정규화 전 대출 데이터 k-means 클러스터링
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

큰 값을 가지는 annual_inc, revol_bal이 클러스터링 결과를 좌우함
클러스터 3에는 비교적 높은 소득(annual_inc)와 높은 회전 신용잔고(revol_bal)을 가진 52명만 포함됨

```
# 정규화 후 k-means 클러스터링 진행
scaler = preprocessing.StandardScaler()
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)
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

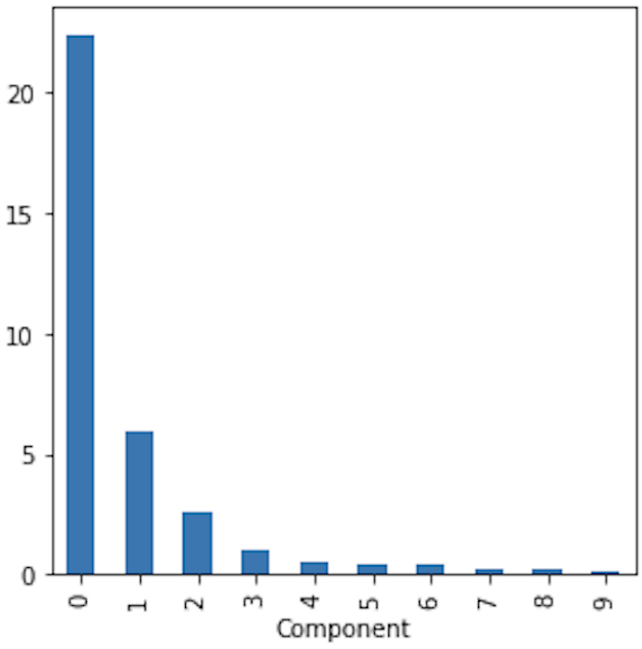
	revol_util	size
0	77.806693	7405
1	31.000342	5339
2	66.172004	3701
3	59.227862	6226

annual_inc와 revol_bal에 큰 영향을 받지 않고 클러스터 4개에 적절히 분배되어 있는 결과를 확인할 수 있음

※ 예제2(지배변수)

```
# 구글과 아마존 주가를 추가한 주가 (PCA에서 사용)
syms = ['GOOGL', 'AMZN', 'AAPL', 'MSFT', 'CSCO', 'INTC', 'CVX', 'XOM',
```

```
        'SLB', 'COP', 'JPM', 'WFC', 'USB', 'AXP', 'WMT', 'TGT', 'HD', 'COST']
top_sp1 = sp500_px.loc[sp500_px.index >= '2005-01-01', syms]

sp_pca1 = PCA()
sp_pca1.fit(top_sp1)

explained_variance = pd.DataFrame(sp_pca1.explained_variance_)
ax = explained_variance.head(10).plot.bar(legend=False, figsize=(4, 4))
ax.set_xlabel('Component')

plt.tight_layout()
plt.show()

loadings = pd.DataFrame(sp_pca1.components_[0:2, :], columns=top_sp1.columns)
print(loadings.transpose())
```



	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

- 스크리그래프는 첫번째 주성분에 대한 분산 표시, 하나 혹은 두개의 변수가 전체를 지배하는 것을 나타냄
- 오른쪽 수치에서는 GOOGL, AMZN에 의해 거의 완전 지배되고 있는 것을 확인할 수 있음
- 이러한 경우에는 변수 스케일링을 포함하거나, 지배 변수를 전체 분석에서 제외하고 별도 처리 할 수도 있음 → 어떤 방법이 항상 옳다고는 할수 없음

※ 예제3(고워거리, 파이썬에서는 고워 거리 계산 지원하지 않고 있음)

- 범주형 데이터가 있는 경우에는 순서형, 이진형 변수를 사용하여 수치형 데이터로 변환

- 연속형, 이진형 변수가 섞여 있을 경우 비슷한 스케일이 되도록 변수 크기를 조정해야함

  → 고워 거리 사용!

> 💡 고워거리
> • 수치형 변수나 순서형 요소에서 두 레코드 간의 거리는 차이이의 절댓값(맨해튼 거리)으로 계산
> • 범주형 변수의 경우 두 레코드 사이의 범주가 서로 다르면 거리가 1이고 범주가 동일하면 거리는 0
>
> ※ 계산 방식
> 1. 각 레코드 변수 i와 j의 모든 쌍에 대해 거리 $d_{i,j}$를 계산한다
> 2. 각 $d_{i,j}$의 크기를 최솟값이 0이고 최댓값이 1이되도록 스케일을 조정한다
> 3. 거리 행렬을 구하기 위해 변수 간에 스케일된 거리를 모두 더한 후 평균 혹은 가중 평균을 계산

▼ 7.6 마치며

- 주성분 분석과 k-means 클러스터링은 수치형 데이터의 차원을 축소하기 위해 주로 사용되는 방법이므로 의미 있는 데이터 축소를 보장하기 위해서는 데이터 스케일을 적절히 조정해야함
- k-means는 매우 큰 데이터로 확장이 가능하고 이해하기 쉽다
- 계층적 클러스터링은 수치형과 범주형이 혼합된 데이터 유형에 적용이 가능하며 직관적인 시각화 방법이 존재함(덴드로그램)
- 모델 기반 클러스터링은 통계 이론에 기초를 두고 있으며 엄밀한 접근방식을 제시