In [117…
```python
#머신러닝 1  고객 정보데이터, 소비 제품 데이터, 구매 채널 데이터
# 1-1 결측치 확인후 결측치 제거
# 1-2 이상치 제거 방법 서술 이상치 제거후 결과를 통계적으로 나타냄
# 1-3 전처리한 ㅈ데이터로 Keman DBSCan  군집 생성
# 2-1 군집 특성 분석
# 2-2 군집별 상품 추천
# 2-3 ID 10870 대상 상품 추천

# 1-1
import pandas as pd
import numpy as np
data=pd.read_csv("https://raw.githubusercontent.com/ADPclass/ADP_book_ver01/main/data/26_problem1.csv")
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 16 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   ID                   2240 non-null   int64
 1   Year_Birth           2240 non-null   int64
 2   Marital_Status       2240 non-null   object
 3   Income               2216 non-null   float64
 4   Kidhome              2240 non-null   int64
 5   Teenhome             2240 non-null   int64
 6   MntWines             2240 non-null   int64
 7   MntFruits            2240 non-null   int64
 8   MntMeatProducts      2240 non-null   int64
 9   MntFishProducts      2240 non-null   int64
 10  MntSweetProducts     2240 non-null   int64
 11  NumDealsPurchases    2240 non-null   int64
 12  NumWebPurchases      2240 non-null   int64
 13  NumCatalogPurchases  2240 non-null   int64
 14  NumStorePurchases    2240 non-null   int64
 15  NumWebVisitsMonth    2240 non-null   int64
dtypes: float64(1), int64(14), object(1)
memory usage: 280.1+ KB
```

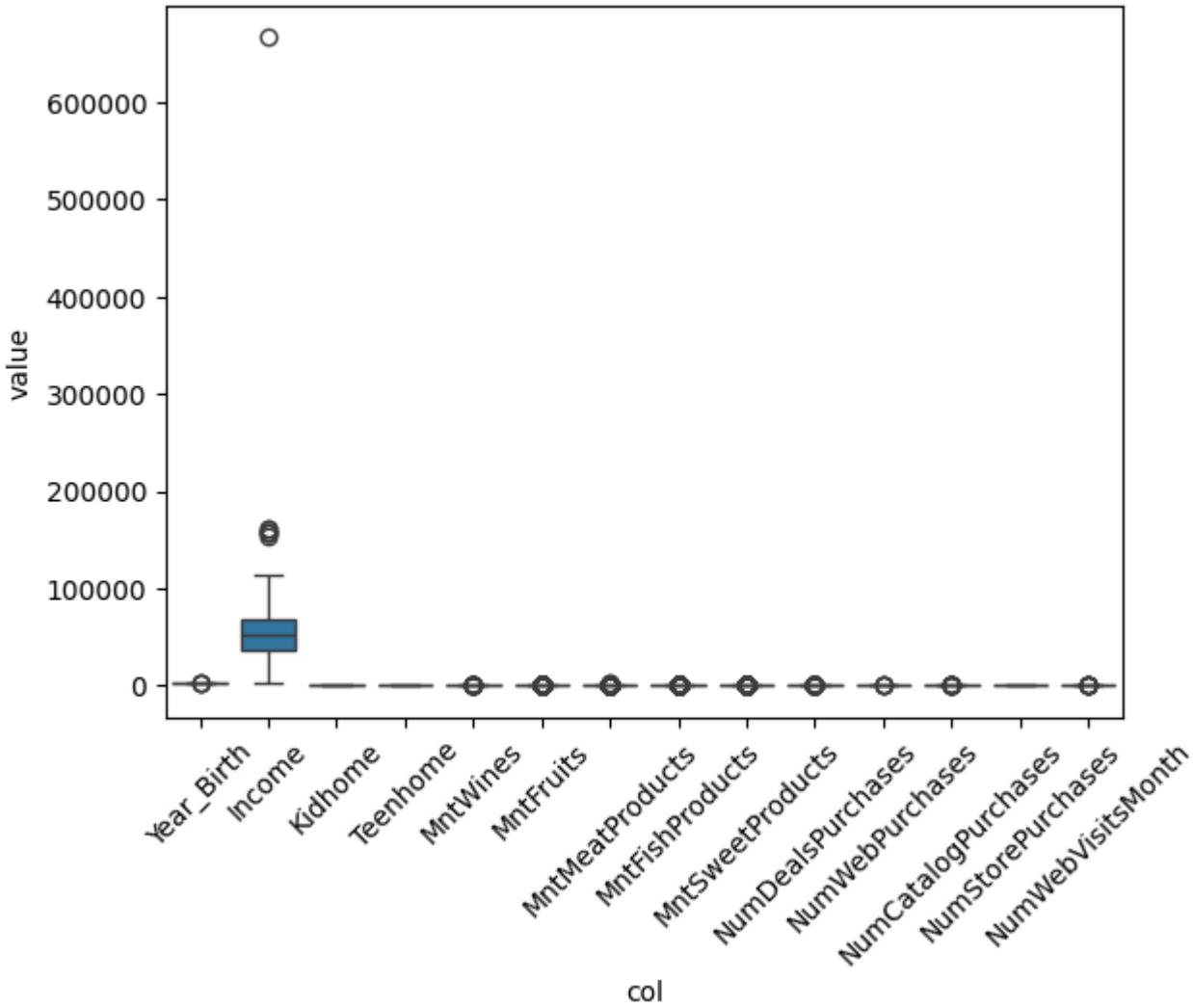
In [118…
```python
Income_mean=data["Income"].mean()
data.loc[data["Income"].isna()==True,"Income"]=Income_mean
# data.isna().sum()
# 24개로 우적어 평균으로 대치
```

In [119…
```python
#1-2 이상치 제거방법 추출 abox plot IQR로 바로 이상치 처리 할수도 있음
import matplotlib.pyplot as plt
```

```
import seaborn as sns
box_col=data.columns.drop(['ID','Marital_Status'])
```

In [120…

```
X=data[box_col]
df_v1=pd.melt(X,var_name='col',value_name='value')
plt.figure()
sns.boxplot(x='col',y='value',data=df_v1)
plt.xticks(range(len(X.columns)),X.columns,rotation=45)
plt.show()
```

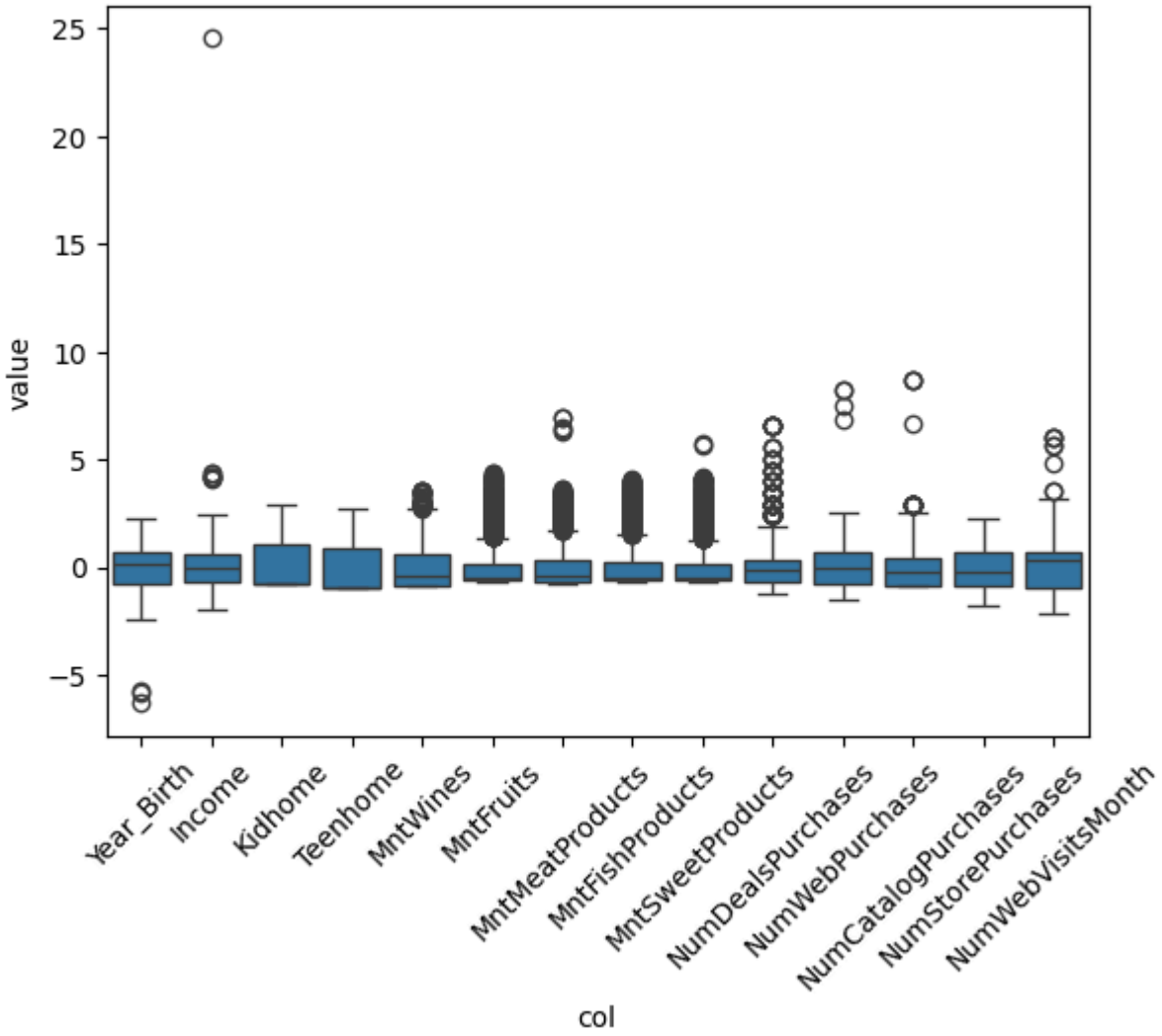


In [121…

```
from sklearn.preprocessing import StandardScaler

X=data[box_col]
```

```
X=StandardScaler().fit_transform(X)
X=pd.DataFrame(X,columns=box_col)
df_v1=pd.melt(X,var_name='col',value_name='value')
plt.figure()
sns.boxplot(x='col',y='value',data=df_v1)
plt.xticks(range(len(X.columns)),X.columns,rotation=45)
plt.show()
```



In [122…

```
def detect_outliers(df,column,weight=1.5):
    Q1=df[column].quantile(0.25)
    Q3=df[column].quantile(0.75)
    IQR=Q3-Q1
    IQR_weight = IQR*weight
    outlier_index=df[(df[column]< Q1-IQR_weight)|(df[column]> Q3+IQR_weight)].index
```

```
    return outlier_index

out_index= detect_outliers(data,'Income')
data.loc[out_index]
#분석가의 판단으로 66666 값만 평균으로 대치하고, 나머지는 특정 계층이라고 판단하여 처리하지 않았다.
```

Out[122…

	ID	Year_Birth	Marital_Status	Income	Kidhome	Teenhome	MntWines	MntFruits	MntMeatProducts	MntFishProducts	MntSweetPro
164	8475	1973	Married	157243.0	0	1	20	2	1582	1	
617	1503	1976	Together	162397.0	1	1	85	1	16	2	
655	5555	1975	Divorced	153924.0	0	0	1	1	1	1	
687	1501	1982	Married	160803.0	0	0	55	16	1622	17	
1300	5336	1971	Together	157733.0	1	0	39	1	9	2	
1653	4931	1977	Together	157146.0	0	0	1	0	1725	2	
2132	11181	1949	Married	156924.0	0	0	2	1	2	1	
2233	9432	1977	Together	666666.0	1	0	9	14	18	8	

In [123…

```
data.loc[out_index[-1],"Income"]=Income_mean
data.loc[out_index]
```

Out[123…

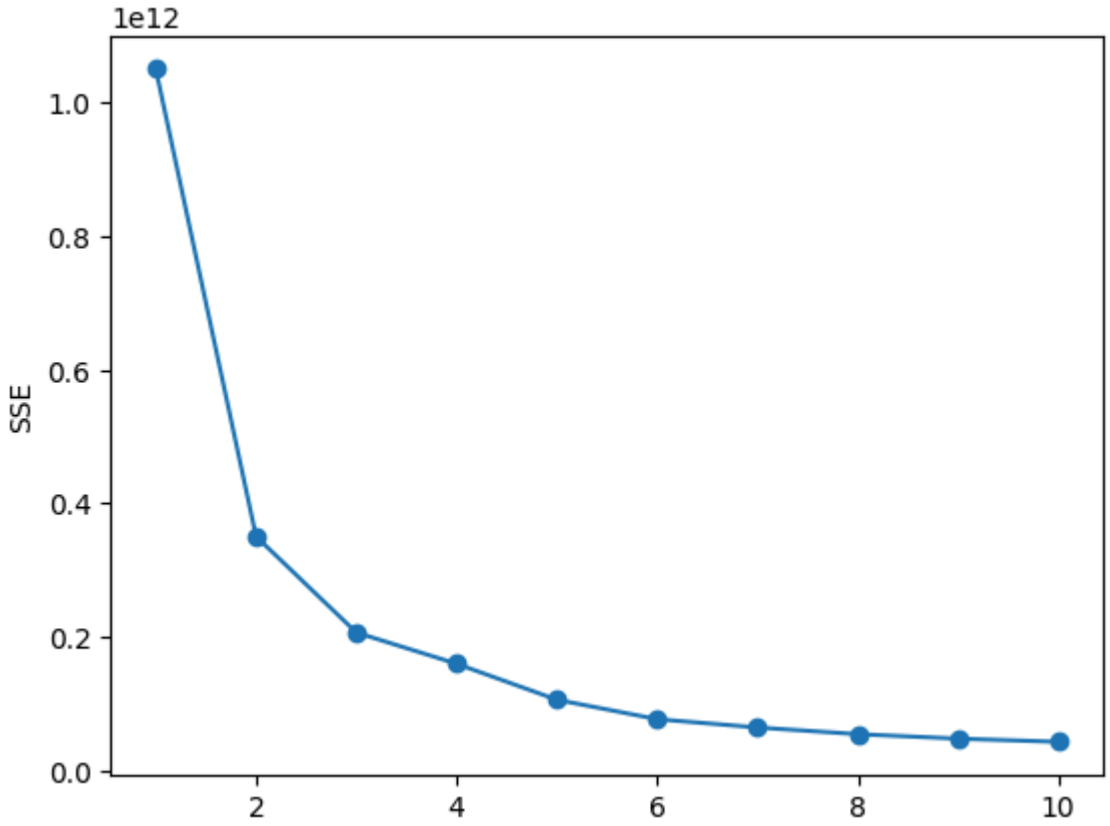
	ID	Year_Birth	Marital_Status	Income	Kidhome	Teenhome	MntWines	MntFruits	MntMeatProducts	MntFishProducts	MntSw
164	8475	1973	Married	157243.000000	0	1	20	2	1582	1	
617	1503	1976	Together	162397.000000	1	1	85	1	16	2	
655	5555	1975	Divorced	153924.000000	0	0	1	1	1	1	
687	1501	1982	Married	160803.000000	0	0	55	16	1622	17	
1300	5336	1971	Together	157733.000000	1	0	39	1	9	2	
1653	4931	1977	Together	157146.000000	0	0	1	0	1725	2	
2132	11181	1949	Married	156924.000000	0	0	2	1	2	1	
2233	9432	1977	Together	52247.251354	1	0	9	14	18	8	

In [124…

```
#1-3
# marital_status 를 엔코딩해야 한다. 최적 클러스터 개수를 결정하는데 엘보우 기법을 사용한다. 클러스터내 오차 제곱합을 클러스터 개수마다 비교
```

```
# 늘려가면서 계산한 SSE 비교 급격히 비율이 작아지는 부분을 최적 클러스터 개수로
from sklearn.cluster import KMeans
def elbow(X):
    sse=[]
    for i in range(1,11):
        km=KMeans(n_clusters=i, random_state=1)
        km.fit(X)
        sse.append(km.inertia_)

    plt.plot(range(1,11),sse,marker='o')
    plt.ylabel('SSE')
    plt.show()
df_dum=pd.get_dummies(data,columns=['Marital_Status'],drop_first=True)
elbow(df_dum)
```



In [125…

```
km=KMeans(n_clusters=3)
km.fit(df_dum)
new_label=km.labels_
data['cluster']=new_label
```

In [126…

```python
import numpy as np
import pandas as pd

from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score


def dbscan_fit_predict(
    X,
    eps: float,
    min_samples: int = 5,
    metric: str = "euclidean",
    scale: bool = True,
):
    """
    DBSCAN 학습 및 라벨 반환. (옵션으로 표준화)
    returns: (labels, X_used)  # X_used는 (표준화된) 피처
    """
    X_used = StandardScaler().fit_transform(X) if scale else X
    model = DBSCAN(eps=eps, min_samples=min_samples, metric=metric)
    # model._
    labels = model.fit_predict(X_used)
    return labels, X_used


def dbscan_silhouette(
    X,
    eps: float,
    min_samples: int = 5,
    metric: str = "euclidean",
    scale: bool = True,
    filter_noise: bool = True,
    penalize_noise: bool = True,
):
    """
    DBSCAN 실루엣 점수 계산.
    - filter_noise=True: 노이즈(-1) 샘플을 제외하고 실루엣 계산(권장)
      * 제외 후 유효 클러스터 수 < 2면 score = NaN
    - penalize_noise=True: 실루엣 * (유효비율)을 추가로 반환 (노이즈가 많으면 감점)
    returns: dict
      {
        'silhouette': float or NaN,          # 노이즈 제외한 실루엣
        'silhouette_adj': float or NaN,      # (옵션) 노이즈 패널티 반영
        'n_clusters': int,                   # 노이즈 제외한 클러스터 수
        'n_noise': int,                      # 노이즈 샘플 수
```

```
    'labels': np.ndarray,               # 원본 라벨(노이즈 = -1)
  }
"""
labels, X_used = dbscan_fit_predict(X, eps, min_samples, metric, scale)
mask = labels != -1
n_noise = int((labels == -1).sum())
n_samples = len(labels)

if filter_noise:
    if mask.sum() == 0:
        return {
            "silhouette": np.nan,
            "silhouette_adj": np.nan if penalize_noise else None,
            "n_clusters": 0,
            "n_noise": n_noise,
            "labels": labels,
        }
    labels_nn = labels[mask]
    # 유효 클러스터 수(노이즈 제외)
    n_clusters = len(set(labels_nn))
    if n_clusters < 2:
        return {
            "silhouette": np.nan,
            "silhouette_adj": np.nan if penalize_noise else None,
            "n_clusters": n_clusters,
            "n_noise": n_noise,
            "labels": labels,
        }
    score = silhouette_score(X_used[mask], labels_nn, metric=metric)
    score_adj = score * (mask.sum() / n_samples) if penalize_noise else None
    return {
        "silhouette": float(score),
        "silhouette_adj": float(score_adj) if score_adj is not None else None,
        "n_clusters": n_clusters,
        "n_noise": n_noise,
        "labels": labels,
    }
else:
    # 노이즈를 하나의 클러스터로 취급하는 것은 권장하지 않음(왜곡)
    # 그래도 원하면 아래처럼 계산 가능하지만 대부분 해석이 나빠져 생략.
    unique = set(labels)
    # -1 포함하면 클러스터 수에서 제외
    n_clusters = len(unique) - (1 if -1 in unique else 0)
    if n_clusters < 2:
        return {
            "silhouette": np.nan,
```

```
                "silhouette_adj": np.nan if penalize_noise else None,
                "n_clusters": n_clusters,
                "n_noise": n_noise,
                "labels": labels,
            }
        score = silhouette_score(X_used, labels, metric=metric)
        score_adj = score * ((n_samples - n_noise) / n_samples) if penalize_noise else None
        return {
            "silhouette": float(score),
            "silhouette_adj": float(score_adj) if score_adj is not None else None,
            "n_clusters": n_clusters,
            "n_noise": n_noise,
            "labels": labels,
        }


def dbscan_grid_compare(
    X,
    eps_list,
    min_samples_list=(5,),
    metric: str = "euclidean",
    scale: bool = True,
    filter_noise: bool = True,
    penalize_noise: bool = True,
):
    """
    여러 (eps, min_samples) 조합을 탐색하고 실루엣/클러스터수를 표로 반환.
    또한 '클러스터 개수별' 베스트 파라미터도 함께 반환.

    returns:
      results_df: DataFrame
        columns = ['eps', 'min_samples', 'n_clusters', 'n_noise',
                   'silhouette', 'silhouette_adj']
      best_by_k_df: DataFrame
        각 n_clusters별로 silhouette_adj(또는 silhouette) 최고 행
    """
    rows = []
    for eps in eps_list:
        for m in min_samples_list:
            out = dbscan_silhouette(
                X, eps, m, metric=metric, scale=scale,
                filter_noise=filter_noise, penalize_noise=penalize_noise
            )
            rows.append({
                "eps": eps,
                "min_samples": m,
```

```
                "n_clusters": out["n_clusters"],
                "n_noise": out["n_noise"],
                "silhouette": out["silhouette"],
                "silhouette_adj": out["silhouette_adj"],
            })

    results_df = pd.DataFrame(rows)

    # 우선 순위: silhouette_adj 가 있으면 그것 기준, 없으면 silhouette
    score_col = "silhouette_adj" if penalize_noise else "silhouette"
    # 유효 점수만 대상
    valid = results_df.dropna(subset=[score_col]).copy()
    # 같은 n_clusters 내 최고 조합 선택
    best_by_k_df = (
        valid.sort_values(score_col, ascending=False)
             .groupby("n_clusters", as_index=False)
             .first()
    )

    # 보기 좋게 정렬
    results_df = results_df.sort_values(
        by=[score_col if score_col in results_df else "silhouette"],
        ascending=False,
        na_position="last"
    ).reset_index(drop=True)

    return results_df, best_by_k_df
from sklearn.datasets import make_blobs

X, _ = make_blobs(n_samples=800, centers=5, cluster_std=0.80, random_state=42)

eps_list = np.round(np.linspace(0.2, 1.2, 11), 2)
min_samples_list = [3, 5, 10]

results, best_by_k = dbscan_grid_compare(
    X,
    eps_list=eps_list,
    min_samples_list=min_samples_list,
    metric="euclidean",
    scale=True,          # 보통 표준화 권장
    filter_noise=True,   # 노이즈 제외하고 실루엣
    penalize_noise=True  # 노이즈 비율로 패널티
)

print(results.head(10))   # 상위 조합 확인
print(best_by_k)          # "클러스터 개수별" 최고 조합
```

```
   eps  min_samples  n_clusters  n_noise  silhouette  silhouette_adj
0  0.3            3           4        0    0.766728        0.766728
1  0.3            5           4        0    0.766728        0.766728
2  0.3           10           4        0    0.766728        0.766728
3  0.2            3           4        2    0.767807        0.765888
4  0.2           10           4        2    0.767807        0.765888
5  0.2            5           4        2    0.767807        0.765888
6  0.4            3           3        0    0.723161        0.723161
7  0.4            5           3        0    0.723161        0.723161
8  0.4           10           3        0    0.723161        0.723161
9  1.0           10           2        0    0.586751        0.586751
   n_clusters  eps  min_samples  n_noise  silhouette  silhouette_adj
0           2  1.0           10        0    0.586751        0.586751
1           3  0.4            3        0    0.723161        0.723161
2           4  0.3            3        0    0.766728        0.766728
```

In [127…

```python
#2-1 위에서 생성한 군집들의 특성을 분석하시오
# 그룹화
# 군집별 변수의 평균을 살펴보는 것.
#
data.cluster.value_counts()
```

Out[127…

```
cluster
2    812
0    741
1    687
Name: count, dtype: int64
```

In [128…

```python
df_dum=df_dum.replace(True, 1)
df_dum=df_dum.replace(False, 0)
df_dum['cluster']=new_label
group_mean= df_dum.groupby(by=['cluster']).mean()
group_mean.reset_index(inplace=True)
group_mean
# 고객정보데이터 소비제품데이터 구매채널 데이터 로 묶어서 보면,
#그룹 0은 67년생으로 소득 평균이 가장 높은 집단이다. 집에 어린아이가 없을 확율이 높고 10대 청소년 비율이 높다.
# 소비제품 데이터로 봤을때 0 은 와인 추천 1은 고기 추천 2는 와인 추천한다. (비율로 환산할 필요가 있다. )
```

```
/var/folders/hv/lqp1gn9n1ll0lbh2pfzn9pww0000gn/T/ipykernel_42727/216373658.py:2: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a future version. To retain the old behavior, explicitly call `result.infer_objects(copy=False)`. To opt-in to the future behavior, set `pd.set_option('future.no_silent_downcasting', True)`
  df_dum=df_dum.replace(False, 0)
```

Out[128...

	cluster	ID	Year_Birth	Income	Kidhome	Teenhome	MntWines	MntFruits	MntMeatProducts	MntFishProducts	...	Num
0	0	5662.222672	1973.105263	28348.147099	0.808367	0.311741	30.568151	5.990553	25.570850	9.068826	...	
1	1	5770.388646	1967.429403	76967.652111	0.084425	0.350801	616.861718	57.052402	397.494905	82.835517	...	
2	2	5377.431034	1966.046798	52385.061926	0.416256	0.815271	288.646552	18.821429	100.912562	25.158867	...	

3 rows × 23 columns

In [129...

```
df_dum[df_dum['ID']==10870] #그룹이 2이니깐 와인상품을 추천한다.
```

Out[129...

	ID	Year_Birth	Income	Kidhome	Teenhome	MntWines	MntFruits	MntMeatProducts	MntFishProducts	MntSweetProducts	...	NumS
2235	10870	1967	61223.0	0	1	709	43	182	42	118	...	

1 rows × 23 columns

In [130...

```
# 통계 분석
# 1 한공장에서 생산된 제품에서 최근 추정 불량율은 90% 이였다. 오차 한계가 5% 이하가 되도록하는 최소 표본 사이즈
# 기출 3에서도
# import numpy as np
# std=np.std(data)
# # 정규 분포
# print((stats.t.ppf(q=0.025,df=9)*std/5)**2)# 오차 한계 5 신뢰수준 0.05 를 넣고 z 인데 양측검정이므로 변경하였음.

# Z_0.05*std/sqrt(n) <0.05
# n>=s^2 * Z^2 / 0.05^2
# s^2 = p(1-p)
p=0.9
z=1.96
d=0.05
n=p*(1-p)*(z**2)/d**2
print(n)
```

```
138.29759999999993
```

다음은 모비율(전체 비율 p)의 추정과 최소 표본수 계산을 시험 대비용으로 깔끔하게 정리한 내용이에요.

1) 모형과 추정량

- 표본 $X_1, \dots, X_n \sim \text{Bernoulli}(p)$ (성공=1, 실패=0).
- 점추정량(표본비율): $\hat{p} = \dfrac{\sum X_i}{n}$.
- 기댓값/분산: $E[\hat{p}] = p$, $\text{Var}(\hat{p}) = \dfrac{p(1-p)}{n}$.
- 표준오차(SE): $\text{SE}(\hat{p}) \approx \sqrt{\dfrac{\hat{p}(1-\hat{p})}{n}}$.

2) 신뢰구간(CI) 요약

- **Wald(가장 단순, 교과서 기본형)**

  $\hat{p} \pm z_{\alpha/2}\sqrt{\dfrac{\hat{p}(1-\hat{p})}{n}}$

  (간단하지만 p가 0 또는 1 근처이거나 n이 작을 때 성능 저하)
- **Wilson** (권장)

$$\frac{\hat{p} + \frac{z^2}{2n} \pm z\sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}}$$

- **Agresti–Coull** (간편 근사)

  $\tilde{n} = n + z^2$, $\tilde{p} = \dfrac{x + \frac{z^2}{2}}{\tilde{n}}$ 후

  $\tilde{p} \pm z\sqrt{\dfrac{\tilde{p}(1-\tilde{p})}{\tilde{n}}}$
- **Clopper–Pearson(정확, 베타분포 기반)**: 작은 표본/극단 비율에 안전하지만 구간이 넓어짐.

> 실무/시험에서 표본수 설계는 대체로 **Wald 근사**로 합니다(공식 단순). 구간 보고는 Wilson/Agresti–Coull 권장.

3) "최소 표본수" n 계산 (오차한계 E 기준)

목표: **신뢰수준 $1-\alpha$**에서 **오차한계 E**(±몇 %p) 달성.

- **Wald 근사 공식(표준)**

$$n_0 \;\geq\; \frac{z_{\alpha/2}^2\, p^*(1-p^*)}{E^2}$$

  - p^*: 기획 단계의 예상 비율(없으면 **보수적으로 0.5** 사용 → 최대 n).
  - $z_{\alpha/2}$: 90%→1.645, 95%→1.96, 99%→2.576.
  - 계산 뒤 **반드시 올림(ceil)**.
- **유한모집단 보정(FPC)** — 모집단 크기 N이 유한하고 표본비율이 큰 경우:

$$n \;=\; \frac{n_0}{1 + \dfrac{n_0 - 1}{N}} \quad \text{(마지막에 올림)}$$

- **설계효과(DEFF)** — 군집표본/가중치 등:

  $n_{\text{final}} = n \times \text{DEFF}$ (경험적으로 1.2~2+)

예시

1. 95% 신뢰, $E = \pm 3\%, p^* = 0.5$

$$n_0 = \frac{1.96^2 \cdot 0.25}{0.03^2} = \frac{3.8416 \cdot 0.25}{0.0009} = \frac{0.9604}{0.0009} \approx 1067.1 \Rightarrow \boxed{1068}$$

2. 95% 신뢰, $E = \pm 5\%, p^* = 0.9$

$$n_0 = \frac{1.96^2 \cdot 0.9 \cdot 0.1}{0.05^2} = \frac{3.8416 \cdot 0.09}{0.0025} \approx 138.3 \Rightarrow \boxed{139}$$

3. 예시 1)에 $N = 2000$ (FPC)

$$n = \frac{1068}{1 + \frac{1067}{2000}} = \frac{1068}{1.5335} \approx 696.5 \Rightarrow \boxed{697}$$

In [ ]:
```
#통계2 은의 가격 은의 가격 및 이동평균값이 3이 설정된 시계열 그래프를 그리시오
import pandas as pd
import numpy as np

data=pd.read_csv("https://raw.githubusercontent.com/ADPclass/ADP_book_ver01/main/data/26_problem4.csv")
data
```

Out[ ]:

	1M	2M	3M	4M	5M	6M	7M	8M	9M
0	12.14	42.6	34.4	35.29	30.96	57.12	37.84	42.49	31.38

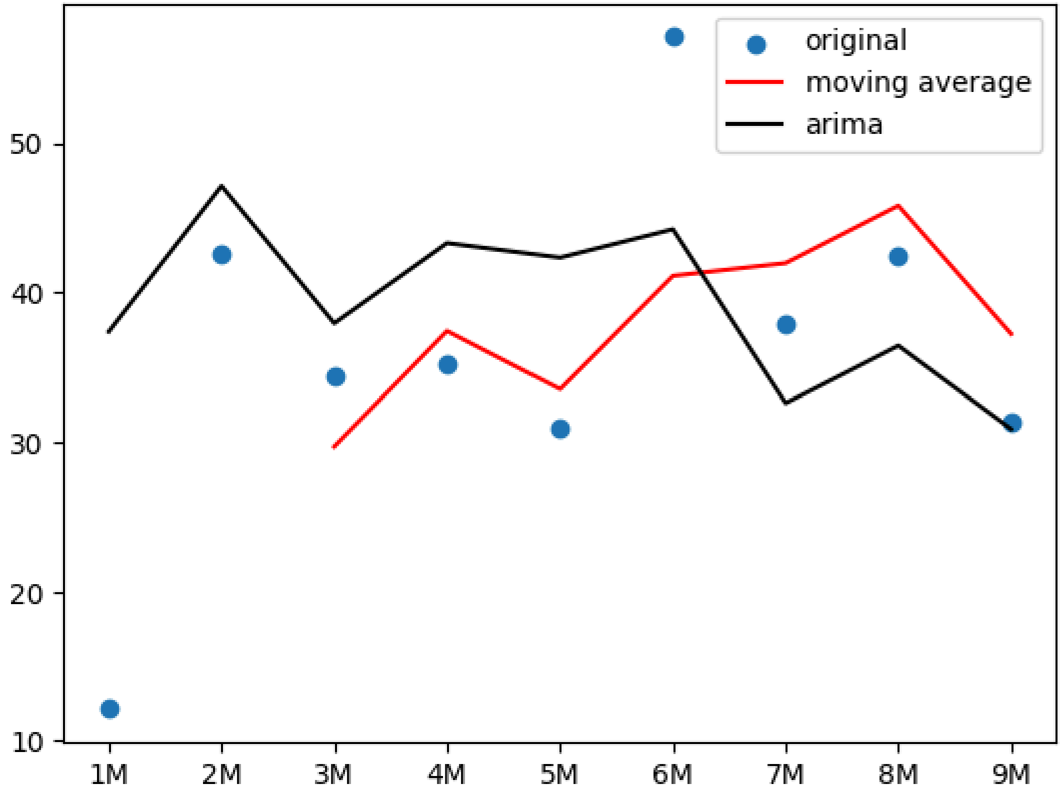
In [132…
```
m_data=data.transpose()
m_data['ma_3']=m_data.iloc[:,0].rolling(3).mean()
```

In [163…
```
from statsmodels.tsa.arima.model import ARIMA
model=ARIMA(pd.DataFrame(m_data.iloc[:,0].values,index=[f'2020-{i[0].zfill(2)}-01'for i in m_data.index]),order=(0,0,3))
res=model.fit()
predict=res.predict()
# res.forecast(steps=len(test),alpha=0.05)
```

```
/opt/homebrew/Caskroom/miniforge/base/envs/general/lib/python3.11/site-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarn
ing: No frequency information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
/opt/homebrew/Caskroom/miniforge/base/envs/general/lib/python3.11/site-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarn
ing: No frequency information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
/opt/homebrew/Caskroom/miniforge/base/envs/general/lib/python3.11/site-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarn
ing: No frequency information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
/opt/homebrew/Caskroom/miniforge/base/envs/general/lib/python3.11/site-packages/statsmodels/tsa/statespace/sarimax.py:866: UserW
arning: Too few observations to estimate starting parameters for ARMA and trend. All parameters except for variances will be set
to zeros.
  warn('Too few observations to estimate starting parameters%s.'
```

In [164…

```
import matplotlib.pyplot as plt
plt.scatter(m_data.index,m_data.iloc[:,0])
plt.plot(m_data.index,m_data.iloc[:,1],color='r')
plt.plot(m_data.index,predict,color='k')
plt.legend(['original','moving average','arima'])
plt.show()
```



In [ ]:
```python
#3 아래 그래프는 A,B,C 자치구별 H의원에 대한 찬성, 반대 투표 결과이다. 자치구별 지지율이 같은지에 대해 검정하시오
# 두개이상 범주형 변수에 대해 변수들간의 분포 차이 보는것이므로 독립성 시행을 한다.
# 귀무가설 자치구와 지지율은 독립니다.
# 자치구와 지지율은 독랍이 아니다.
import pandas as pd
import numpy as np

data=pd.DataFrame({'vote':['찬성','반대'],'A':[176,124],'B':[193,107],'C':[159,141],})
data.set_index("vote",inplace=True)

from scipy.stats import chi2_contingency
chi,p,df,expect = chi2_contingency(data) #독립이 아니라고 할수 있다. 즉 자치구별 지지율은 다르다.
# 독립성검정이 아니라 동질성 검증을 행하하네
# 요약: 네, “자치구 × 지지(찬성/반대)” 교차표로 χ² 독립성 검정(= 동질성 검정)을 하면 됩니다.
# 귀무가설 자치구와 지지여부는 독립이다 ⇔ 자치구별 지지율 분포가 같다.
# 대립가설 자치구와 지지여부는 독립이 아니다 ⇔ 자치구별 지지율 분포가 다르다(적어도 한 구가 다름).
```

In [165…
```python
#4 남녀학생들의 평균 혈압차가 있는지 여부 검정
#4-1 차이는 없다. / 있다
import pandas as pd
data=pd.read_csv("https://raw.githubusercontent.com/ADPclass/ADP_book_ver01/main/data/26_problem6.csv")
# e등분산을 만족하는 조건에 독립 t검정수행
import scipy.stats as stats
male=data.loc[data.gender=='male','pressure']
female=data.loc[data.gender=='female','pressure']
print('male',stats.shapiro(male))
print('female',stats.shapiro(female))
test_result=stats.ttest_ind(male,female,equal_var=True)
print(test_result)#차이가 없다.
```

```
male ShapiroResult(statistic=0.8797191977500916, pvalue=0.03841618821024895)
female ShapiroResult(statistic=0.814755916595459, pvalue=0.030057914555072784)
TtestResult(statistic=1.3813481801194591, pvalue=0.18044550626193742, df=23.0)
```

In [166…
```python
#4-3 평균 혈압차의 신뢰구간을 구했을때 4-2를 지지하는지?
# S_p/sqrt(1/n1+1/n2)
from numpy import array , mean
from scipy.stats import sem,t
import numpy as np
# np.ndarray.std()는 기본이 ddof=0(모표준편차). **표본표준편차는 ddof=1**로
def sp(data1,data2):
    df=len(data1)+len(data2)-2
    s1=(len(data1)-1)*(data1.std(ddof=1)**2)
    s2=(len(data2)-1)*(data2.std(ddof=1)**2)
    sp=np.sqrt((s1+s2)/df)
    return sp
```

```python
# t.interval(alpha, …)는 **신뢰수준(confidence)**을 넣습니다. 95% CI면 0.95여야 해요(α=0.05 아님).
# 실무에선 t.ppf로 임계값을 구해 mean ± tcrit*SE가 더 명확합니다.
alpha=0.05
dof=len(male)+len(female)-2
diff_mean=male.mean()-female.mean()
s=sp(male,female)*(1/len(male)+1/len(female))
CI=t.interval(1-2*alpha,dof,loc=diff_mean,scale=s)
print(CI,diff_mean)
#신뢰 구간안에 들어오고 있고. t검정 결과를 지지한다.
```

```
(2.55839113800193, 8.034664417553634) 5.296527777777783
```

`alpha_1`, `alpha_2`

- 각 **가중치 정밀도 α_i**의 감마사전(Gamma prior) 파라미터

$$p(\alpha_i) = \text{Gamma}(\text{shape} = \alpha_1, \text{scale} = \alpha_2)$$

- 즉, 역감마(InvGamma)로 치환 시 분산 사전의 **형상(shape)** 과 **척도(scale)** 역할
- α_2가 작을수록 더 "유연"해지고,

  α_2가 크면 정규화가 강해집니다.

> **권장:**
> `alpha_1=1e-6`, `alpha_2=1e-6` (기본값, 거의 비정보적)
> 또는 `alpha_2=0.005` 같이 약간의 수축을 주어 안정화

`lambda_1`, `lambda_2`

- **노이즈 정밀도(λ)** 의 감마 사전 하이퍼파라미터

$$p(\lambda) = \text{Gamma}(\text{shape} = \lambda_1, \text{scale} = \lambda_2)$$

- λ가 클수록 "잔차 분산"이 작아지도록 압박 (즉, 더 빡빡한 모델)
- `lambda_1`이 작으면 관대하게 noise를 허용

> 보통 ARDRegression 논문 예제에서는 `lambda_1=lambda_2=1e-6` 또는 `0.005`

In [ ]:
```
#5-1 회귀계수를 구해서 소수점두자리 구하시오
data=pd.read_csv("https://raw.githubusercontent.com/ADPclass/ADP_book_ver01/main/data/26_problem7.csv")
from sklearn import linear_model
from sklearn.linear_model._bayes import ARDRegression
# 1000번의 번인 이후 1만번의 mcmc 수행
# 회귀 계수의 사전분포는 부적절한 균일 분포
# 오차항의 분산의 사전분포는 역감마 분포로 지정
# 형상모수와 척도 모수는 각각 0.005로 지정
from sklearn.model_selection import train_test_split
X=data[["height","weight"]]
y=data['waistline']
X_train,X_test,Y_train,Y_test=train_test_split(X,y,test_size=0.3,random_state=1)
clf=ARDRegression(max_iter=1000,alpha_2=0.005,lambda_1=0.005, fit_intercept=False)
clf.fit(X_train,Y_train)
y_pred=clf.predict(X_test)
clf.coef_
```

Out[ ]:
```
array([ 0.54084419, -0.1964856 ])
```

In [167…
```
#5-1 회귀계수를 구해서 소수점두자리 구하시오
data=pd.read_csv("https://raw.githubusercontent.com/ADPclass/ADP_book_ver01/main/data/26_problem7.csv")
from sklearn import linear_model
from sklearn.linear_model._bayes import ARDRegression
# 1000번의 번인 이후 1만번의 mcmc 수행
# 회귀 계수의 사전분포는 부적절한 균일 분포
# 오차항의 분산의 사전분포는 역감마 분포로 지정
# 형상모수와 척도 모수는 각각 0.005로 지정
from sklearn.model_selection import train_test_split
X=data[["height","weight"]]
y=data['waistline']
X_train,X_test,Y_train,Y_test=train_test_split(X,y,test_size=0.3,random_state=1)
clf=ARDRegression(max_iter=1000,alpha_1=0.005,alpha_2=0.005,lambda_1=0.005,lambda_2=0.005, fit_intercept=False)
clf.fit(X_train,Y_train)
y_pred=clf.predict(X_test)
clf.coef_
```

Out[167…
```
array([ 0.54098204, -0.1968069 ])
```