위 문서는 해당 고객 등급 예측 모델의 데이터분석 및 모델링 근거임

```
[4]: customer = pd.read_csv("data/customer_data.csv", encoding='cp949') # 한글 포맷
     customer.head()
```

[4]:

	ID	signup_ym	birth_year	annual_income	marital_status	children	Recency	amount_alcohol	amount_fruit	amount_meat	...	num_purchase_store
0	5735	22.May	1999	117829400.0	미혼	0	29	1502800	156000	1189500	...	10
1	5350	22.May	1999	117829400.0	미혼	0	29	1502800	156000	1189500	...	10
2	1763	21.Oct	1996	113982700.0	배우자 있음	0	62	1636700	223600	1059500	...	10
3	4580	22.Jan	1977	98486700.0	배우자 있음	0	46	1812200	28600	920400	...	9
4	4475	21.May	1957	89827400.0	배우자 있음	0	82	1709500	28600	1014000	...	9

5 rows × 25 columns

데이터 전처리

```
[7]: customer.columns
```

```
[7]: Index(['ID', 'signup_ym', 'birth_year', 'annual_income', 'marital_status',
            'children', 'Recency', 'amount_alcohol', 'amount_fruit', 'amount_meat',
            'amount_fish', 'amount_snack', 'amount_general', 'Monetary',
            'num_purchase_web', 'num_purchase_store', 'num_purchase_discount',
            'Frequency', 'promotion_1', 'promotion_2', 'promotion_3', 'promotion_4',
            'promotion_5', 'promotion_6', 'revenue'],
           dtype='object')
```

1. 데이터 전처리를 위해 데이터를 파악

⚠️ ID: 고객 식별번호 # 상관x

⚠️ signup_ym: 가입 연월 (예: 20.Dec) # 크게 영향x

⚠️ birth_year: 출생연도 # 부정적 영향

✅ annual_income: 연소득 (원 단위 추정)

⚠️ marital_status: 혼인 상태 (미혼, 배우자 있음 등) # 순서형 범주x

⚠️ children: 자녀 수 # 크게 영향x

✅ Recency: 최근 구매 후 경과일수

amount_alcohol: 주류 구매 금액

amount_fruit: 과일 구매 금액

amount_meat: 육류 구매 금액

amount_fish: 생선 구매 금액

amount_snack: 디저트/과자류 구매 금액

amount_general: 일반류 구매 금액

✅ Monetary : 총 구매 합산액

✅ num_purchase_web: 온라인 구매 횟수

✅ num_purchase_store: 매장 구매 횟수

✅ num_purchase_discount: 할인 구매 횟수

✅ Frequency: 총 구매 빈도(합산)

⚠️ promotion_1 ~ promotion_5: 각 프로모션 반응 여부 (1=참여, 0=불참) # 부정적 영향

⚠️ revenue: 매출 등급/지표 # 고정 값 : 11

```
[13]: # 필요한 컬럼만 지정
      sel_columns= ['annual_income','Recency','Monetary','Frequency',
                    'num_purchase_store','num_purchase_web','num_purchase_discount']

      cc = customer[sel_columns]
```

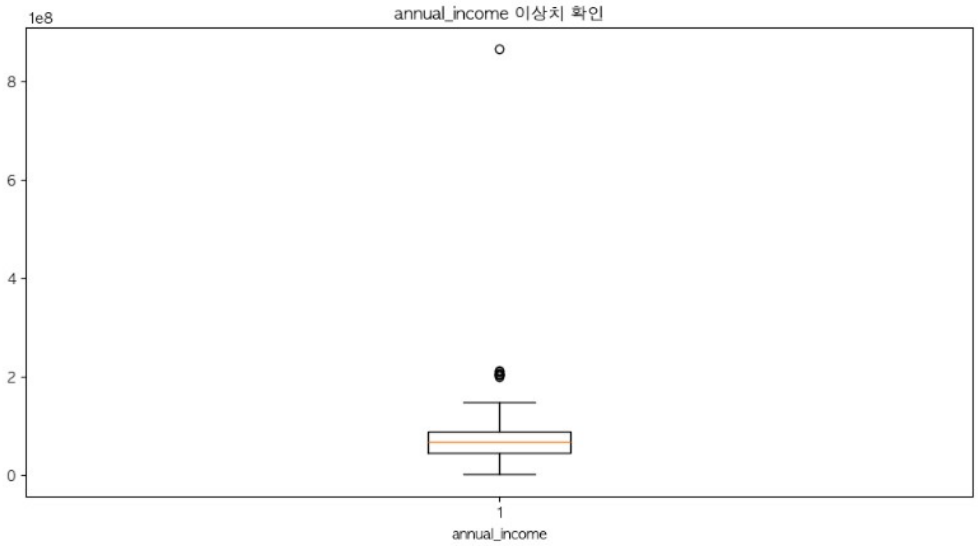
2. 클러스터링에 활용할 컬럼 선정

[21]:
```
# 결측치 제거
cc.dropna(inplace=True)
```

[23]:
```
# 이상치 확인
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))

# annual_income 박스플롯
plt.plot(1, 2)
plt.boxplot(cc['annual_income'])
plt.title('annual_income 이상치 확인')
plt.xlabel('annual_income')
plt.show()
```



3. 결측치 제거 및 이상치 확인(제거)

[35]:
```
#표준화
sc = StandardScaler()
cc_scaled = sc.fit_transform(cc)
pd.DataFrame(cc_scaled)
```

[35]:

	0	1	2	3	4	5	6
0	1.164200	-0.104870	3.121574	1.286366	0.990352	1.933281	-0.790211
1	0.843774	1.136184	3.045401	0.926388	0.990352	1.161427	-0.790211
2	1.860131	-0.794345	2.899680	0.926388	1.914632	0.003646	-0.790211
3	2.031576	-1.139083	2.894712	1.106377	1.914632	0.775500	-1.523928
4	1.519790	1.067236	2.889744	0.746398	1.298445	0.389573	-0.790211
...	...	...	...	...	...	...	...
2112	-0.912770	-1.345925	-0.980151	-1.413472	-1.166301	-1.154135	-0.790211
2113	-0.816801	-1.656189	-0.981807	-1.413472	-1.166301	-1.154135	-0.790211
2114	-0.864329	0.550130	-0.981807	-1.413472	-1.166301	-1.154135	-0.790211
2115	-2.246330	-1.001188	-0.985118	-2.133429	-1.782488	-1.540062	-1.523928
2116	-2.311704	0.239867	-0.986774	-2.133429	-1.782488	-1.540062	-1.523928

2117 rows × 7 columns

[37]:
```
# 최적의 클러스터개수를 찾는 프로그램
max_score = 0
optimized_n = 0
for k in range(2,10):

    km = KMeans(n_clusters=k,random_state=21).fit(cc_scaled)

    s_score = silhouette_score(cc_scaled,km.labels_)
    if s_score > max_score:
        max_score = s_score
        optimized_n = k
    print("score for %d clusters:%.3f" % (k, s_score))

print(f'\n 최적의 클러스터개수: {optimized_n}, 실루엣 점수: {max_score}')
```

```
score for 2 clusters:0.378
score for 3 clusters:0.351
score for 4 clusters:0.288
score for 5 clusters:0.231
score for 6 clusters:0.227
score for 7 clusters:0.216
score for 8 clusters:0.221
score for 9 clusters:0.224

 최적의 클러스터개수: 2, 실루엣 점수: 0.3775015277294302
```

4. 데이터 표준화 및 최적의 클러스터개수 찾기

- 최적의 클러스터개수는 2개지만, 회원등급을 분류하는데 2가지는 너무 단순하다 판단되어, 3개로 하였습니다. (브론즈,실버,골드)

[40]:
```
#cc_scaled 데이터를 넣어 클러스터링
kmeans = KMeans(n_clusters=3, random_state=0)
clusters = kmeans.fit(cc_scaled)

cc = cc.copy()
#클러스터링 변수인 clusters 값을 원본 데이터인 'cc'내에 넣기
cc['cluster'] = clusters.labels_
```

차원 축소

- 군집화 시각화시 Feature가 3개 이상인 경우
- 머신러닝 학습시 Feature가 너무 많아서 차원의 저주가 발생한 경우

[46]:
```
cc.columns # 총 7개의 컬럼 사용 ('cluster'제외)
```

[46]:
```
Index(['annual_income', 'Recency', 'Monetary', 'Frequency',
       'num_purchase_store', 'num_purchase_web', 'num_purchase_discount',
       'cluster'],
      dtype='object')
```

[ ]:
```
from sklearn.decomposition import PCA
X = cc_scaled.copy()

# 7차원의 데이터를 n_components 차원으로 축소
pca = PCA(n_components=2, random_state=21).fit(X)

x_pca = pca.transform(X)
x_pca
```

[48]:
```
array([[ 3.70527819, -1.16272978],
       [ 3.04348115, -1.32741299],
       [ 3.33071203, -1.95193259],
       ...,
       [-2.55066667, -0.69099931],
       [-4.00001054, -1.09104516],
       [-4.01881625, -1.06975998]])
```

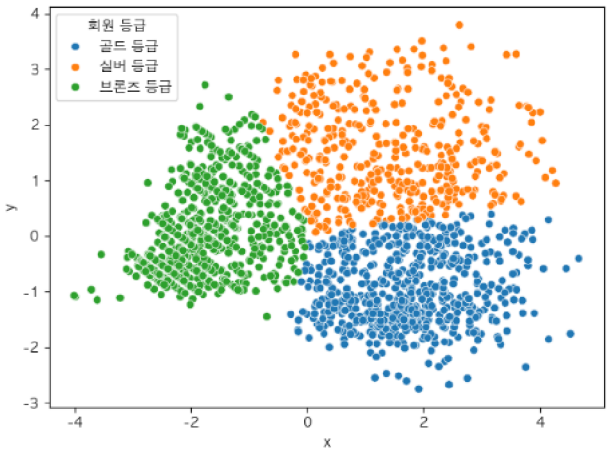
5. 최적의 클러스터는 2 개, 하지만 등급을 나누기에 단순하여 3 개로 진행함

→ 차원축소 진행

[58]:
```
import seaborn as sns

fig, ax = plt.subplots()

cluster_names = {
    2: "브론즈 등급",
    0: "실버 등급",
    1: "골드 등급",
}
sns.scatterplot(
    x='x',
    y='y',
    hue=cc_df['cluster'].map(cluster_names),
    data=cc_df
)
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.legend(title="회원 등급")
plt.tight_layout()
```



6. 클러스터링 후, 시각화

데이터분석

feature, label 설정

[68]:
```
label_name = 'cluster' # 종속변수
cc[label_name].unique() # 3개의 클러스터
```

[68]:
```
array([1, 0, 2], dtype=int32)
```

[70]:
```
features = cc[cc.columns.difference([label_name])] # 'cluster'빼고 features지정
label = cc[label_name] # 'cluster'로 label 지정
```

[72]:
```
# 훈련, 시험 데이터셋 분리
train_input, test_input, train_target, test_target = train_test_split(features, label, test_size=0.2, random_state=42)
```

7. feature 와 label 을 설정 후, 훈련 데이터셋, 시험 데이터셋 분리

LightGBM 사용

하이퍼파라메터 튜닝

[76]:
```
from scipy.stats import uniform, randint
```

[78]:
```
params = {'n_estimators': randint(50, 100),
          'learning_rate': uniform(0.001, 0.3),
          'max_depth': randint(3, 20),
          'min_child_samples': randint(10, 100),
#         'subsample': uniform(0.5, 1.0),
#         'colsample_bytree':uniform(0.5, 1.0),
          'reg_alpha':uniform(0.0, 1.0),
          'reg_lambda':uniform(0.0, 1.0),
          'num_leaves':randint(5, 50)
          }
```

[80]:
```
from sklearn.model_selection import RandomizedSearchCV
# cc 데이터 프레임에서 데이터 타입이 문자열(object)인 모든 열의 컬럼 리스트를 반환
obj_categorical_features = cc.select_dtypes(include=['object']).columns.tolist()
```

[82]:
```
obj_categorical_features # 없음
```

[82]:
```
[]
```

[84]:
```
lgbm = LGBMClassifier(random_state=42)
# n_iter=100: 하이퍼파라메터의 샘플링(rvs) 조합을 100번까지 시도
gs = RandomizedSearchCV(LGBMClassifier(random_state=42), params,
                        n_iter=100, n_jobs=-1, random_state=42)
gs.fit(train_input, train_target)
```

```
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000477 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 669
[LightGBM] [Info] Number of data points in the train set: 1355, number of used features: 7
[LightGBM] [Info] Start training from score -1.538233
[LightGBM] [Info] Start training from score -1.124782
[LightGBM] [Info] Start training from score -0.775406
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

8. LightGBM 활용, 하이퍼 파라메터 튜닝 진행

```
[86]: # 최적의 파라매터 확인
print(gs.best_params_)
```

```
{'learning_rate': 0.12411487691068891, 'max_depth': 13, 'min_child_samples': 38, 'n_estimators': 85, 'num_leaves': 17,
'reg_alpha': 0.07697990982879299, 'reg_lambda': 0.289751452913768}
```

```
[88]: # 테스트 스코어 확인
print(np.max(gs.cv_results_['mean_test_score']))
```

```
0.9875914192456058
```

- 최종 서비스 모델

```
[91]: # 최적의 파라미터를 입력
best_params = gs.best_params_
```

```
[93]: service_model = LGBMClassifier(
    n_estimators=int(best_params['n_estimators']),
    learning_rate=float(best_params['learning_rate']),
    max_depth=int(best_params['max_depth']),
    min_child_samples=int(best_params['min_child_samples']),
    reg_alpha=float(best_params['reg_alpha']),
    reg_lambda=float(best_params['reg_lambda']),
    num_leaves=int(best_params['num_leaves']),
    random_state=42
)
```

9. 최적의 파라메터 확인후, 최종 서비스 모델 설정

```
[95]: service_model.fit(train_input, train_target)
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[97]: # 훈련 스코어 확인
service_model.score(train_input,train_target)
```

```
[97]: 1.0
```

```
[99]: # 테스트 스코어 확인
service_model.score(test_input,test_target)
```

```
[99]: 0.9882075471698113
```

10. 최종 모델 훈련 스코어, 테스트 스코어 확인

서비스 모델 개발

```
[ ]: def collect_user_input():
    user_input = {}
    print("고객 등급 예측하기 위한 특성을 입력하세요.")
    for feature in cc.columns.difference([label_name]):
        val = input(f"{feature} (수치형): ").strip()
        user_input[feature] = float(val) if val != "" else None
    return user_input

input_features = collect_user_input()
```

```
고객 등급 예측하기 위한 특성을 입력하세요.
Frequency (수치형):  20
Monetary (수치형):  1200000
Recency (수치형):  30
annual_income (수치형):  45000000
num_purchase_discount (수치형):  4
num_purchase_store (수치형):  5
num_purchase_web (수치형):  8
```

```
Frequency (수치형): 구매 빈도(합산)
Monetary (수치형): 구매 총액
Recency (수치형): 최근 구매 후 경과일수
annual_income (수치형): 연봉 (원 단위)
num_purchase_discount (수치형): 할인 구매 횟수
num_purchase_store (수치형): 매장 구매 횟수
num_purchase_web (수치형): 온라인 구매 횟수
```

11. 고객 등급 예측 서비스 모델 개발

```python
# 클러스터 별 최솟값, 최댓값을 이용하여, 클러스터 특성 확인
cols = ['Recency','Monetary','Frequency','num_purchase_store',
        'num_purchase_web','num_purchase_discount','annual_income']

summary = cc.groupby('cluster', sort=True)[cols].agg(['min','max'])

for cl, row in summary.iterrows():
    print(f"\n[클러스터 {cl}]")
    for col in cols:
        vmin = row[(col,'min')]
        vmax = row[(col,'max')]
        if col in ['annual_income','Monetary']:
            print(f" - {col}의 최솟값: {vmin:,.0f}, 최댓값: {vmax:,.0f}")
        else:
            print(f" - {col}의 최솟값: {vmin:.0f}, 최댓값: {vmax:.0f}")
```

```
[클러스터 0]
 - Recency의 최솟값: 0, 최댓값: 99
 - Monetary의 최솟값: 226,200, 최댓값: 2,377,700
 - Frequency의 최솟값: 12, 최댓값: 27
 - num_purchase_store의 최솟값: 3, 최댓값: 13
 - num_purchase_web의 최솟값: 1, 최댓값: 11
 - num_purchase_discount의 최솟값: 1, 최댓값: 6
 - annual_income의 최솟값: 29,259,100, 최댓값: 105,690,000

[클러스터 1]
 - Recency의 최솟값: 0, 최댓값: 99
 - Monetary의 최솟값: 362,700, 최댓값: 3,231,800
 - Frequency의 최솟값: 7, 최댓값: 25
 - num_purchase_store의 최솟값: 4, 최댓값: 13
 - num_purchase_web의 최솟값: 1, 최댓값: 11
 - num_purchase_discount의 최솟값: 0, 최댓값: 4
 - annual_income의 최솟값: 53,303,900, 최댓값: 137,112,300

[클러스터 2]
 - Recency의 최솟값: 0, 최댓값: 99
 - Monetary의 최솟값: 6,500, 최댓값: 1,170,000
 - Frequency의 최솟값: 0, 최댓값: 13
 - num_purchase_store의 최솟값: 0, 최댓값: 8
 - num_purchase_web의 최솟값: 0, 최댓값: 6
 - num_purchase_discount의 최솟값: 0, 최댓값: 5
 - annual_income의 최솟값: 4,552,600, 최댓값: 89,880,700
```

12. 각 클러스터별 특성 파악

클러스터 별 특성

클러스터 2 - 브론즈 (저빈도·저금액·저소득 포함)

- Monetary: 최저 (6,500~117만)
- Frequency: 최저 (0~13회)
- 구매 채널: 매장·웹·할인 모두 낮음
- 소득: 저소득 포함 (약 455만~8,988만)
- 특징: 구매금액·빈도 모두 낮은 소극적 소비자

클러스터 0 - 실버 (중간 지출, 빈번 구매, 충성 고객)

- Monetary: 중간 (22만~237만)
- Frequency: 높음 (12~27회)
- 구매 채널: 매장·웹·할인 참여 중간 수준
- 소득: 중상위 (약 2,926만~1억 5600만)
- 특징: 구매 빈도가 높고 여러 채널에서 꾸준히 소비하는 충성 고객

클러스터 1 - 골드 (고소득·고액 소비, 우수 고객)

- Monetary: 최고 (36만~323만)
- Frequency: 중상위 (7~25회)
- 구매 채널: 매장·웹은 유사, 할인 참여는 낮음
- 소득: 가장 높음 (5,330만~1억 3711만)
- 특징: 고소득·고액 소비자, 활동성도 높아 핵심 VIP 고객군

등급별 마케팅 전략

브론즈 (저빈도·저금액·저소득 포함)

- 특징: 구매 횟수 적고 금액 낮음. 소득도 낮은 층 포함.

전략 - 진입 장벽 낮추고 재구매 활성화

- 재구매 활성화: 소액 프로모션, 첫 구매 후 리마인드 메시지 발송
- 가격 장벽 완화: 저가형 묶음 상품, 대량 할인 쿠폰 제공
- 체험 중심: 샘플·체험 쿠폰으로 재구매 경험 유도

실버 (중간 지출, 빈번 구매, 충성 고객)

- 특징: 구매 횟수 많고 소비 금액은 중간 수준. 충성 고객층.

전략 - 충성도 관리와 등급 승급 유도

- 충성도 강화: 멤버십·포인트 적립, 등급 승급 혜택
- 구매 확장: 자주 사는 품목과 연계한 크로스셀링 제안
- 생활 밀착: 가족 단위 혜택, 정기구독 모델 제시
- 승급 인센티브: 골드 등급으로 이동할 수 있는 인센티브 제공

골드 (고소득·고액 소비, 우수 고객)

- 특징: 소득 높고 지출 크며 활동성도 높은 VIP 층.

전략 - VIP 차별화와 프리미엄 유지

- 프리미엄 경험: VIP 전용 상담, 특별 이벤트 초대
- 맞춤형 제안: 고가·한정판 상품 추천, 개인화 마케팅 강화
- 구매 확대: 일정 금액 이상 장바구니 혜택 제공 (예: 30만 이상 사은품)
- 차별화 유지: 고급스럽고 차별화된 메시지 전달

13. 특성 파악 및 그에 따른 마케팅 전략 인사이트 도출