# 머신러닝 기반 카드 이탈고객 예측



In [ ]:

```

```

In [ ]:

```

```

## 1. 개요

- 카드 회사들은 고객 유치를 위해 많은 프로모션을 진행
- 새로운 고객을 유치하는 것보다 기존 고객을 유지하는 것이 경제적 효과큼
- 기존 고객의 이탈 여부를 사전에 예측 가능한 모델 구축

In [ ]:

```

```

In [ ]:

```

```

## 2. 데이터 수집

- 데이터 출처 (링크 kaggle)
- 은행의 고객 중에서 신용카드 고객 이탈자에 대한 자료
- 이탈 고객의 16.07%만 자료를 가지고 있어서 한계가 있는 데이터임
- 연령, 급여, 결혼 여부, 신용카드 한도, 신용카드 등급 등 여러 정보를 가지고 이탈 고객 분석

In [ ]:

```

```

## 3. 데이터 확인 및 시각화

**모듈 임포트**

In [1]:

```python
import pandas as pd # 데이터 핸들링
import numpy as np
import matplotlib.pyplot as plt # 데이터 시각화
%matplotlib inline
import seaborn as sns # 데이터 시각화(고급분석)
```

In [2]:

```python
import platform

from matplotlib import font_manager, rc
plt.rcParams['axes.unicode_minus'] = False

if platform.system() == 'Windows':  # 윈도우
    path = "c:/Windows/Fonts/malgun.ttf"
    font_name =
font_manager.FontProperties(fname=path).get_name()
    rc('font', family=font_name)
else:
    print('Unknown system...  sorry~~~')
```

```
Unknown system...  sorry~~~
```

In [ ]:

```python

```

**데이터 불러오기**

- 데이터명 : **df**
- 경로 : **"./data/BankChurners.csv"**로 통일하였다.

In [3]:

```python
df =
pd.read_csv("/Users/ds/project2/HuijinKim/data/BankChurners.csv

## 불필요한 열 2개 제거
df = df.iloc[:,:-2]
```

In [ ]:

```python

```

In [4]:

```python
df.keys()
```

Out[4]:

```
Index(['CLIENTNUM', 'Attrition_Flag', 'Customer_Age', 'Gender',
       'Dependent_count', 'Education_Level', 'Marital_Status',
       'Income_Category', 'Card_Category', 'Months_on_book',
       'Total_Relationship_Count', 'Months_Inactive_12_mon',
       'Contacts_Count_12_mon', 'Credit_Limit', 'Total_Revolving_Bal',
```

```
       'Avg_Open_To_Buy', 'Total_Amt_Chng_Q4_Q1', 'Total_Trans_Amt',
       'Total_Trans_Ct', 'Total_Ct_Chng_Q4_Q1', 'Avg_Utilization_Ratio'],
      dtype='object')
```

## 데이터 정보

- ~'CLIENTNUM' : 고객 식별 번호~
- 'Attrition_Flag' : 신용 카드 이탈 여부 `Target 값`
  - Existing Customer : 잔류
  - Attrited Customer : 이탈
- 'Customer_Age' : 고객 나이
- 'Gender' : 성별
- 'Dependent_count' : 부양 가족 수
- 'Education_Level' : 학력 수준
- 'Marital_Status' : 결혼 여부
- 'Income_Category' : 연 소득 구간
- 'Card_Category' : 카드 등급
- 'Months_on_book' : 카드 할부 기간
- 'Total_Relationship_Count' : 가입 상품 수
- 'Months_Inactive_12_mon' : 1년 동안 카드 결재 내역이 없는 비활성 기간(개월)
- 'Contacts_Count_12_mon' : 연락 빈도
- 'Credit_Limit' : 신용 한도
- 'Total_Revolving_Bal' : 할부 잔액
- ~'Avg_Open_To_Buy' : 평균 실 사용 가능 금액 : 'Credit_Limit' - 'Total_Revolving_Bal'~
- ~'Total_Amt_Chng_Q4_Q1'~ : 결제 대금 기준 1분기 대비 4분기 (비율)
- ~'Total_Trans_Amt'~ : 실제 사용 총액
- 'Total_Trans_Ct' : 실제 사용 횟수
- 'Total_Ct_Chng_Q4_Q1' : 1분기 대비 4분기 결제 대금 횟수 비율
- ~'Avg_Utilization_Ratio' : 'Total_Revolving_Bal'/ 'Credit_Limit' (할부 비율)~

| 변수명 | 변수형태 | 구분 |
|---|---|---|
| CLIENTNUM | INT | 피처변수 |
| Customer_Age | INT | 피처변수 |
| Gender | Object | 피처변수 |
| Dependent_count | INT | 피처변수 |
| Education_Level | Object | 피처변수 |
| Marital_Status | Object | 피처변수 |
| Income_Category | Object | 피처변수 |
| Card_Category | Object | 피처변수 |
| Months_on_book | INT | 피처변수 |
| Total_Relationship_Count | INT | 피처변수 |
| Months_Inactive_12_mon | INT | 피처변수 |
| Contacts_Count_12_mon | INT | 피처변수 |
| Credit_Limit | INT | 피처변수 |
| Total_Revolving_Bal | INT | 피처변수 |

| 변수명 | 변수형태 | 구분 |
|---|---|---|
| Avg_Open_To_Buy | INT | 피처변수 |
| Total_Amt_Chng_Q4_Q1 | INT | 피처변수 |
| Total_Trans_Amt | INT | 피처변수 |
| Total_Trans_Ct | INT | 피처변수 |
| Total_Ct_Chng_Q4_Q1 | INT | 피처변수 |
| Avg_Utilization_Ratio | INT | 피처변수 |

In [ ]:

```
```

### 데이터 확인

In [5]:

```
## 데이터 유형 및 결측치 확인
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10127 entries, 0 to 10126
Data columns (total 21 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   CLIENTNUM                 10127 non-null  int64  
 1   Attrition_Flag            10127 non-null  object 
 2   Customer_Age              10127 non-null  int64  
 3   Gender                    10127 non-null  object 
 4   Dependent_count           10127 non-null  int64  
 5   Education_Level           10127 non-null  object 
 6   Marital_Status            10127 non-null  object 
 7   Income_Category           10127 non-null  object 
 8   Card_Category             10127 non-null  object 
 9   Months_on_book            10127 non-null  int64  
 10  Total_Relationship_Count  10127 non-null  int64  
 11  Months_Inactive_12_mon    10127 non-null  int64  
 12  Contacts_Count_12_mon     10127 non-null  int64  
 13  Credit_Limit              10127 non-null  float64
 14  Total_Revolving_Bal       10127 non-null  int64  
 15  Avg_Open_To_Buy           10127 non-null  float64
 16  Total_Amt_Chng_Q4_Q1      10127 non-null  float64
 17  Total_Trans_Amt           10127 non-null  int64  
 18  Total_Trans_Ct            10127 non-null  int64  
 19  Total_Ct_Chng_Q4_Q1       10127 non-null  float64
 20  Avg_Utilization_Ratio     10127 non-null  float64
dtypes: float64(5), int64(10), object(6)
memory usage: 1.6+ MB
```

In [6]:

```
df.head()
```

Out[6]:

| | CLIENTNUM | Attrition_Flag | Customer_Age | Gender | Dependent_count | Education_Level | Ma |
|---|---|---|---|---|---|---|---|
| **0** | 768805383 | Existing Customer | 45 | M | 3 | High School | |
| **1** | 818770008 | Existing Customer | 49 | F | 5 | Graduate | |

| | CLIENTNUM | Attrition_Flag | Customer_Age | Gender | Dependent_count | Education_Level | Ma |
|---|---|---|---|---|---|---|---|
| **2** | 713982108 | Existing Customer | 51 | M | 3 | Graduate | |
| **3** | 769911858 | Existing Customer | 40 | F | 4 | High School | |
| **4** | 709106358 | Existing Customer | 40 | M | 3 | Uneducated | |

5 rows × 21 columns

In [7]:
```
df.describe()
```

Out[7]:

| | CLIENTNUM | Customer_Age | Dependent_count | Months_on_book | Total_Relationship_C |
|---|---|---|---|---|---|
| **count** | 1.012700e+04 | 10127.000000 | 10127.000000 | 10127.000000 | 10127.00 |
| **mean** | 7.391776e+08 | 46.325960 | 2.346203 | 35.928409 | 3.81 |
| **std** | 3.690378e+07 | 8.016814 | 1.298908 | 7.986416 | 1.55 |
| **min** | 7.080821e+08 | 26.000000 | 0.000000 | 13.000000 | 1.00 |
| **25%** | 7.130368e+08 | 41.000000 | 1.000000 | 31.000000 | 3.00 |
| **50%** | 7.179264e+08 | 46.000000 | 2.000000 | 36.000000 | 4.00 |
| **75%** | 7.731435e+08 | 52.000000 | 3.000000 | 40.000000 | 5.00 |
| **max** | 8.283431e+08 | 73.000000 | 5.000000 | 56.000000 | 6.00 |

In [ ]:
```

```

### 시각화 ( sns.pairplot(df) )

- 사용할 데이터의 상관관계를 파악하기 위해 시각화
- 모든 칼럼에 대한 시각화

In [8]:
```
### 실행시간으로 인한 주석 처리
# sns.pairplot(df)
```

In [ ]:
```

```

### 결측치 확인

- isnull().sum() 코드에서는 결측치가 존재하지 않는 것을 확인
- 각각의 피처를 분석해서 결측치 존재 여부 확인 필요

In [9]:
```
df.isnull().sum()
```

Out[9]:
```
CLIENTNUM                   0
Attrition_Flag              0
Customer_Age                0
Gender                      0
```

```
Dependent_count             0
Education_Level             0
Marital_Status              0
Income_Category             0
Card_Category               0
Months_on_book              0
Total_Relationship_Count    0
Months_Inactive_12_mon      0
Contacts_Count_12_mon       0
Credit_Limit                0
Total_Revolving_Bal         0
Avg_Open_To_Buy             0
Total_Amt_Chng_Q4_Q1        0
Total_Trans_Amt             0
Total_Trans_Ct              0
Total_Ct_Chng_Q4_Q1         0
Avg_Utilization_Ratio       0
dtype: int64
```

In [ ]:

In [ ]:

# 4. 데이터 전처리 및 피쳐 엔지니어링

## 4 (1) 타겟 데이터 전처리

- 관측 대상 : `"Attrited Customer"` 1로 설정 (카드 탈퇴)
- 비관측 대상 : `"Existing Cumstomer"` 0으로 설정 (카드 유지)

In [10]:

```
df["Attrition_Flag"].value_counts()
```

Out[10]:

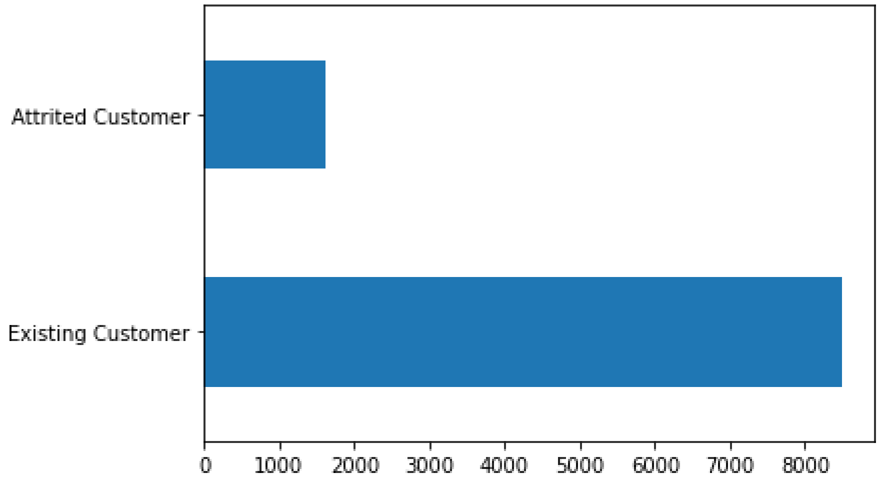
```
Existing Customer    8500
Attrited Customer    1627
Name: Attrition_Flag, dtype: int64
```

In [11]:

```
# 타겟 데이터의 분포를 바 플롯을 이용하여 시각화
df["Attrition_Flag"].value_counts().plot(kind = 'barh')
```

Out[11]:

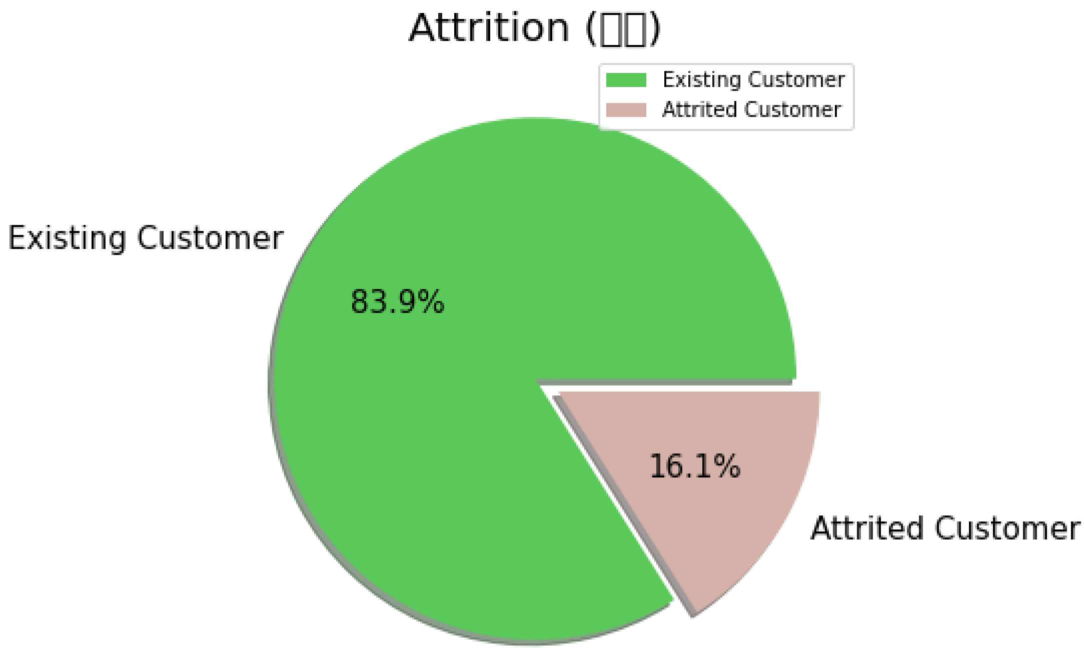
```
<AxesSubplot:>
```




In [12]:

```
# 타겟 데이터의 분포를 파이 플롯을 이용하여 시각화
labels = ['Existing Customer', 'Attrited Customer']
size = df['Attrition_Flag'].value_counts()
colors = ['#5bc959','#d5b1aa' ]
explode = [0, 0.1]

plt.style.use('seaborn-deep')
plt.rcParams['figure.figsize'] = (6, 6)
plt.pie(size, labels =labels, colors = colors, explode =
explode, autopct = "%.1f%%", shadow = True, textprops =
{'fontsize':15})
plt.axis('off')
plt.title('Attrition (비율)', fontsize = 20)
plt.legend()
plt.show()
```

```
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/matplotlib/backends/ba
ckend_agg.py:238: RuntimeWarning: Glyph 48708 missing from current font.
  font.set_text(s, 0.0, flags=flags)
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/matplotlib/backends/ba
ckend_agg.py:238: RuntimeWarning: Glyph 50984 missing from current font.
  font.set_text(s, 0.0, flags=flags)
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/matplotlib/backends/ba
ckend_agg.py:201: RuntimeWarning: Glyph 48708 missing from current font.
  font.set_text(s, 0, flags=flags)
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/matplotlib/backends/ba
ckend_agg.py:201: RuntimeWarning: Glyph 50984 missing from current font.
  font.set_text(s, 0, flags=flags)
```




**레이블 인코딩**

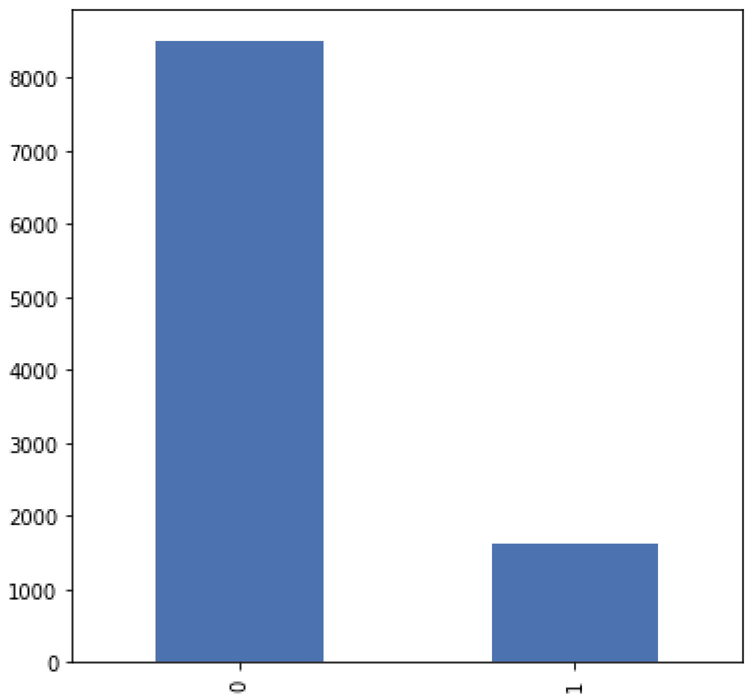
- "Existing Cumstomer" : 0 (카드 잔존)
- "Attrited Customer" : 1 (카드 탈퇴)

**주의하여 관측해야할 것이 "탈퇴"여부이기 때문에 탈퇴를 1, 잔존을 0으로 하여 인코딩**

In [13]:
```
df["Attrition_Flag"].replace({"Existing Customer":0,
                              "Attrited Customer":1,
                              },inplace=True)
```

In [14]:
```
df["Attrition_Flag"].value_counts().plot(kind = 'bar')
```

Out[14]: <AxesSubplot:>



In [15]:
```
df["Attrition_Flag"].value_counts() # 전처리 확인
```

Out[15]:
```
0    8500
1    1627
Name: Attrition_Flag, dtype: int64
```

**Existing Cumstomer(잔존)과 Attrited Customer(이탈)의 비율에 차이가 있다. 수치를 비교해보도록 한다.**

In [16]:
```
Existing = df[df["Attrition_Flag"]==0]
Attrited = df[df["Attrition_Flag"]==1]

Existing_ratio = len(Existing)/len(Existing+Attrited)
Attrited_ratio = len(Attrited)/len(Existing+Attrited)

print("카드를 유지한 고객은 {:.2f} 이고, 탈퇴 고객은 {:.2f}이므로, 카드를 유지한 고객이 {:.0f}배 많
다".format(Existing_ratio,Attrited_ratio,Existing_ratio/Attrite
```

```
카드를 유지한 고객은 0.84 이고, 탈퇴 고객은 0.16이므로, 카드를 유지한 고객이 5배 많다
```

**업샘플링(오버샘플링)이나 다운샘플링(언더샘플링)이 필요한지 확인해 보아야한다.(업샘플링을 해야한다)**

**업샘플링은 전처리 과정이 아니라, 모델을 돌리면서 성능을 높이기 위한 작업과정이다 >>** 업샘플링 하기 전에 모델링을 한번 하고 >> 모델의 성능을 높이기 위한 방법으로 업샘플링을진행한다**

In [ ]:
```

```

## 4 (2) 피처 데이터 전처리

**분류 분석에서의 피처(독립)변수들은 피처들간의 상관성이 높지 않은 이상은 웬만하면 버리지 않고(삭제하지 않고) 쓰는 방향으로 작업을 진행한다**

**피처변수들은 Labels(명목척도) , Orders(서열척도), Numerics(수치형) 변수로 구분하여 전처리를 진행 하였다.**

In [17]:
```
Labels = ['Gender','Marital_Status'] # 명목 척도
Orders =
['Education_Level','Income_Category','Card_Category'] # 서열
척도 (등간 척도)
Numerics =
['Customer_Age','Dependent_count','Months_on_book',
       'Total_Relationship_Count', 'Months_Inactive_12_mon',
       'Contacts_Count_12_mon', 'Credit_Limit',
'Total_Revolving_Bal',
       'Avg_Open_To_Buy', 'Total_Amt_Chng_Q4_Q1',
```

```
'Total_Trans_Amt',
        'Total_Trans_Ct', 'Total_Ct_Chng_Q4_Q1',
'Avg_Utilization_Ratio'] # 수치형 변수
```

In [ ]:

```

```

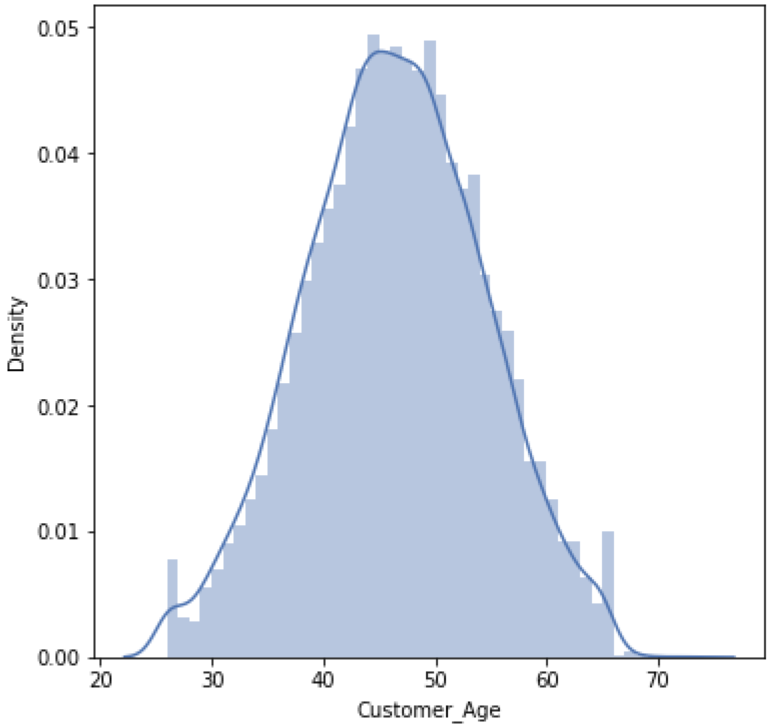
**Customer_Age(나이)**

In [18]:

```
sns.distplot(df["Customer_Age"])
```

```
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/seaborn/distributions.
py:2557: FutureWarning: `distplot` is a deprecated function and will be remove
d in a future version. Please adapt your code to use either `displot` (a figur
e-level function with similar flexibility) or `histplot` (an axes-level functi
on for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[18]: <AxesSubplot:xlabel='Customer_Age', ylabel='Density'>




**Customer_Age (나이) 칼럼은 거의 완벽한 정규분포를 따른다는 것을 알 수 있다.**

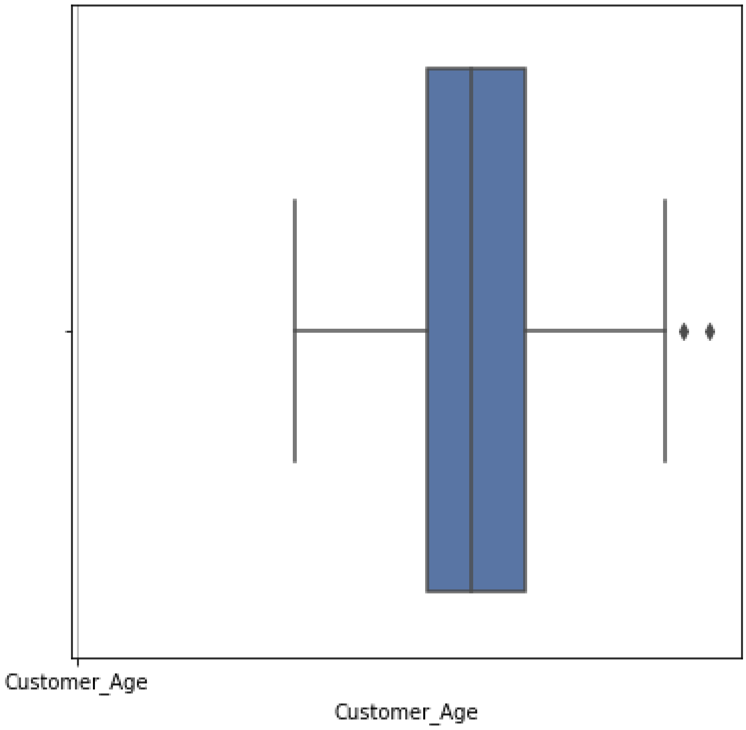
**이상치 확인(시각화)**

In [19]:

```
df[["Customer_Age"]].boxplot()
sns.boxplot(df["Customer_Age"])
```

```
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/seaborn/_decorators.p
y:36: FutureWarning: Pass the following variable as a keyword arg: x. From ver
sion 0.12, the only valid positional argument will be `data`, and passing othe
r arguments without an explicit keyword will result in an error or misinterpre
tation.
  warnings.warn(
```

Out[19]: <AxesSubplot:xlabel='Customer_Age'>




**이상치 확인 (수치 확인)** 여기서 박스플롯에서 범위 밖으로 넘어가는 2개의 plot이 보이는데, 이 값이 이 범위를 벗어난다고 하더라도, 이 수치가 데이터 분석에 큰 영향을 끼치지 않으면 ( 말이 안되는 값이거나 범위에서 너무 벗어난 값 -예를 들면 우리나라 연봉을 조사하는데 이재용 삼성 부회장의 연봉은 제외를 해야 한다. ) 그 값은 이상치로 취급하지 않는다.

In [20]:
```
df["Customer_Age"].max()
```

Out[20]: 73

**나이 칼럼에서 가장 큰 수치는 73인데 이 수치는 충분히 가능한 값이다.만약 이 값이 200이다 라고 한다면 이 이상치는 삭제나 대체가 필요하지만, 73세는 나이로서 충분히 가능한 수치이므로 전처리 하지않는다.(이상치 취급하지 않음)**

In [21]:
```
# 이상치 확인 (참고 사항)
q1 = np.quantile(df["Customer_Age"],0.25)
q3 = np.quantile(df["Customer_Age"],0.75)
iqr = q3-q1
q3+iqr*1.5
q1-iqr*1.5

cond1 = q3+iqr*1.5<df["Customer_Age"]
cond2 = df["Customer_Age"]<q1-iqr*1.5

outlier_index = df[cond1 | cond2].index
```

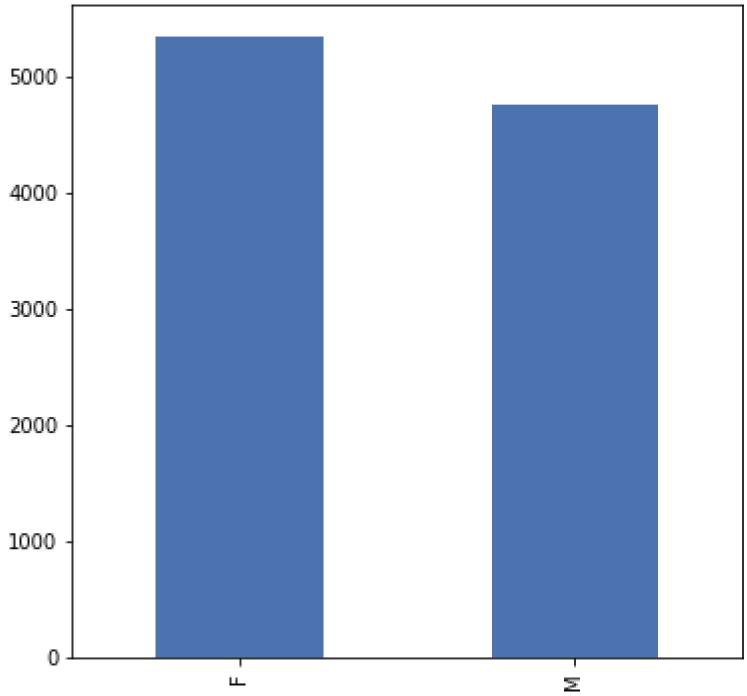
In [ ]:
```

```

**Gender(성별)**

In [22]:

```
# 성별의 분포를 bar 그래프를 이용하여 시각화
df['Gender'].value_counts().plot(kind = 'bar')
```

Out[22]: <AxesSubplot:>




In [23]:

```
# 성별 이탈고객 수
gend_df = pd.DataFrame(df.loc[:,
['Gender','Attrition_Flag']].value_counts())
gend_df
```

Out[23]:

| Gender | Attrition_Flag | 0 |
|---|---|---|
| F | 0 | 4428 |
| M | 0 | 4072 |
| F | 1 | 930 |
| M | 1 | 697 |

In [24]:

```
# 성별 평균 나이
df.groupby('Gender')['Customer_Age'].agg(**
{'Customer_Age':'mean'}).reset_index()
```

Out[24]:

| | Gender | Customer_Age |
|---|---|---|
| 0 | F | 46.456887 |
| 1 | M | 46.178863 |

In [25]:

```
# 성별 나이 분포
df[['Gender','Customer_Age']].value_counts()
```

Out[25]:

```
Gender  Customer_Age
F       44              277
        45              272
        49              263
        47              258
        48              249
                       ...
        66                2
M       67                2
        68                2
        70                1
        73                1
Length: 86, dtype: int64
```

In [26]:

```
# 성별 라벨링 F : 0, M : 1
df["Gender"].replace({"F":1, "M":0},inplace=True)
```

In [27]:

```
df["Gender"].value_counts()
# Female 아니면 Male로 이진분류가 잘 되어있다.
# Label Encoding
```

Out[27]:

```
1    5358
0    4769
Name: Gender, dtype: int64
```

In [28]:

```
# 다른 코드
# from sklearn.preprocessing import LabelEncoder

# le = LabelEncoder()
# le.fit(df["Gender"])
# df["Gender"] = le.transform(df["Gender"])
```

In [29]:

```
df["Gender"] # 라벨인코딩 된 것을 확인 할 수 있다.
```

Out[29]:

```
0        0
1        1
2        0
3        1
4        0
        ..
10122    0
10123    0
10124    1
10125    0
10126    1
Name: Gender, Length: 10127, dtype: int64
```

In [ ]:

```

```

In [30]:

```
# 부양 가족 수 값
df['Dependent_count'].value_counts().index
```

Out[30]:

```
Int64Index([3, 2, 1, 4, 0, 5], dtype='int64')
```

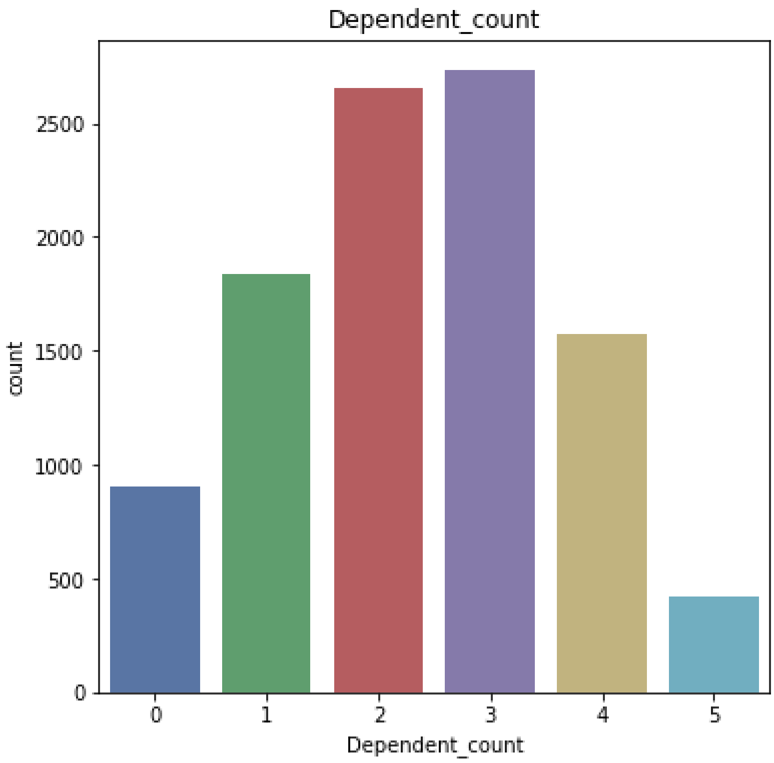
In [31]:

```
# 부양 가족 수 분포
plt.title('Dependent_count')
sns.countplot(df["Dependent_count"])
```

```
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/seaborn/_decorators.p
y:36: FutureWarning: Pass the following variable as a keyword arg: x. From ver
sion 0.12, the only valid positional argument will be `data`, and passing othe
r arguments without an explicit keyword will result in an error or misinterpre
tation.
  warnings.warn(
```

Out[31]:

```
<AxesSubplot:title={'center':'Dependent_count'}, xlabel='Dependent_count', yla
bel='count'>
```




In [32]:

```
bins = list(range(20,81,10))
bins
```

Out[32]:

```
[20, 30, 40, 50, 60, 70, 80]
```

In [33]:

```
labels = [str(i) + '대' for i in bins]
labels
```

Out[33]:

```
['20대', '30대', '40대', '50대', '60대', '70대', '80대']
```

In [34]:
```
# 나이대 라벨 추가
df["age_bin"] = pd.cut(df["Customer_Age"], bins = bins, right
= False, labels=labels[:-1])
df.head()
```

Out[34]:

| | CLIENTNUM | Attrition_Flag | Customer_Age | Gender | Dependent_count | Education_Level | Ma |
|---|---|---|---|---|---|---|---|
| **0** | 768805383 | 0 | 45 | 0 | 3 | High School | |
| **1** | 818770008 | 0 | 49 | 1 | 5 | Graduate | |
| **2** | 713982108 | 0 | 51 | 0 | 3 | Graduate | |
| **3** | 769911858 | 0 | 40 | 1 | 4 | High School | |
| **4** | 709106358 | 0 | 40 | 0 | 3 | Uneducated | |

5 rows × 22 columns

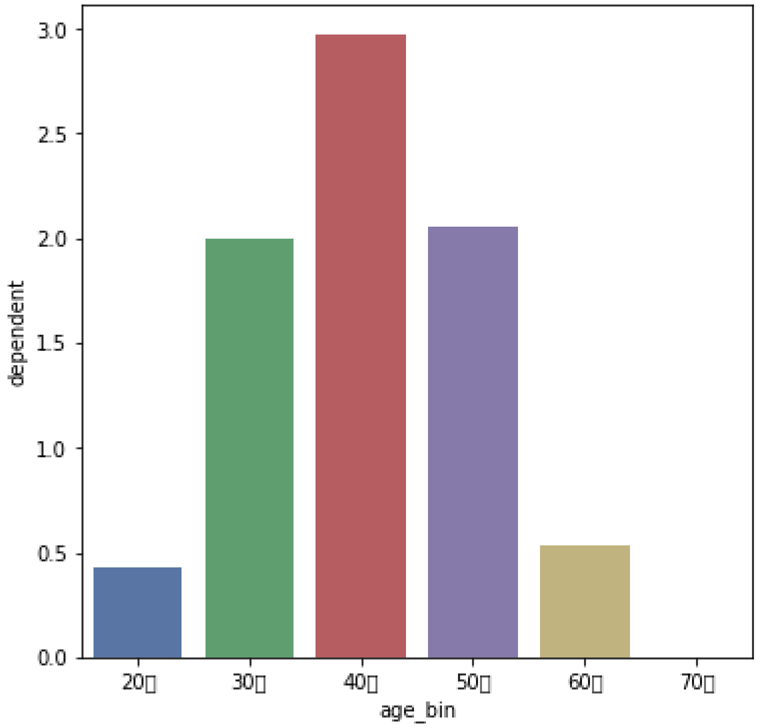
In [35]:
```
# 연령대별 평균 부양가족 수
dependent = df.groupby('age_bin')['Dependent_count'].agg(**
{'dependent':'mean'}).reset_index()
dependent
```

Out[35]:

| | age_bin | dependent |
|---|---|---|
| **0** | 20대 | 0.430769 |
| **1** | 30대 | 2.002173 |
| **2** | 40대 | 2.970401 |
| **3** | 50대 | 2.055037 |
| **4** | 60대 | 0.530189 |
| **5** | 70대 | 0.000000 |

In [36]:
```
# 연령대별 평균 부양 가족 수
sns.barplot(x='age_bin',y='dependent',data=dependent)
```

Out[36]:
```
<AxesSubplot:xlabel='age_bin', ylabel='dependent'>

/Users/heejinkim/miniforge3/lib/python3.9/site-packages/matplotlib/backends/ba
ckend_agg.py:238: RuntimeWarning: Glyph 45824 missing from current font.
  font.set_text(s, 0.0, flags=flags)
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/matplotlib/backends/ba
ckend_agg.py:201: RuntimeWarning: Glyph 45824 missing from current font.
  font.set_text(s, 0, flags=flags)
```




In [37]:

```
# 성별 평균 부양가족 수
gender = df.groupby('Gender')['Dependent_count'].agg(**
{'dependent':'mean'}).reset_index()
gender
```

Out[37]:

| | Gender | dependent |
|---|---|---|
| **0** | 0 | 2.352485 |
| **1** | 1 | 2.340612 |

In [38]:

```
# 연령대별 이탈고객
age_churn = pd.DataFrame(df.loc[:,
['age_bin','Attrition_Flag']].value_counts().sort_index(ascendi

age_churn
```

Out[38]:

| | | 0 |
|---|---|---|
| **age_bin** | **Attrition_Flag** | |
| **20대** | **0** | 178 |
| | **1** | 17 |
| **30대** | **0** | 1580 |
| | **1** | 261 |
| **40대** | **0** | 3789 |
| | **1** | 772 |
| **50대** | **0** | 2492 |

| age_bin | Attrition_Flag | 0 |
|---|---|---|
| | 1 | 506 |
| 60대 | 0 | 459 |
| | 1 | 71 |
| 70대 | 0 | 2 |

In [ ]:

```
```

In [ ]:

```
```

## 4 (2) 범주형 피처 인코딩

**Card_Category(카드 등급)**

In [39]:

```
# Card_Category(카드 등급) 은 4개의 등급(Blue, Silver, Gold,
Platinum)으로 나뉘어진 범주형 변수이다.
# 라벨링 인코딩을 해주도록 한다.
df["Card_Category"].replace({"Blue":0,
                                "Silver":1,
                                "Gold":2,
                                "Platinum":3,
                                },inplace=True)
```

In [ ]:

```
```

**타겟변수(탈퇴인지, 잔존인지)에 따른 분포를 바 그래프로 보여주는 함수**

In [40]:

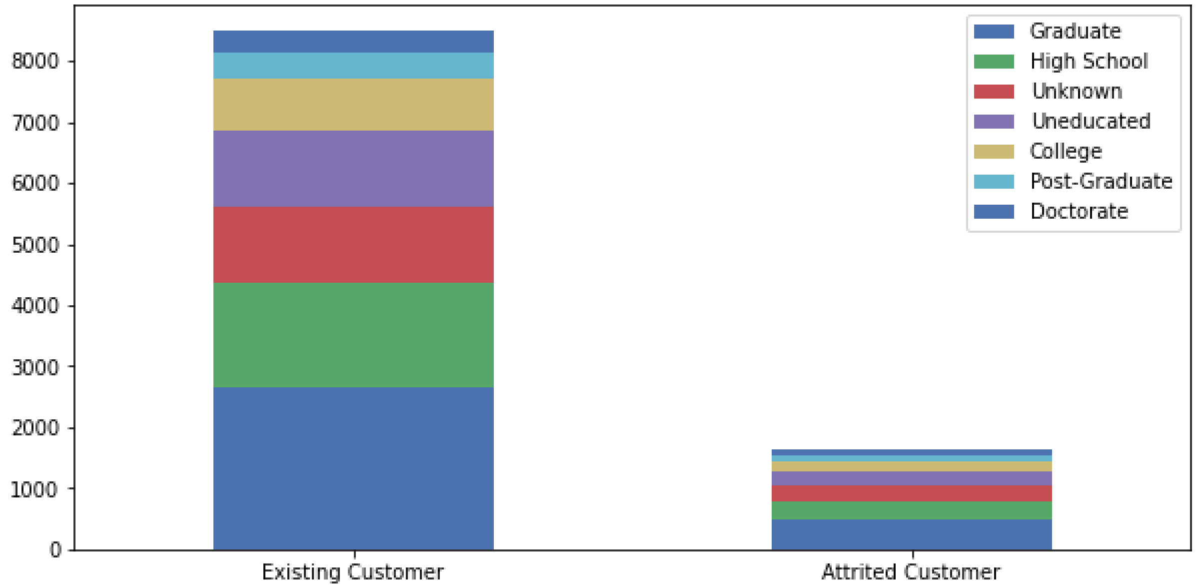
```
def bar_chart(feature):
    stay = df[df['Attrition_Flag']==0]
[feature].value_counts()
    leave = df[df['Attrition_Flag']==1]
[feature].value_counts()
    temp = pd.DataFrame([stay,leave])
    temp.index = ['Existing Customer','Attrited Customer']
    temp.plot(kind='bar',stacked=True, figsize=(10,5))
    plt.xticks(rotation=0)
```

In [ ]:

```
```

**Education_Level 수준에 따른 고객 이탈자**

In [41]:
```
bar_chart('Education_Level')
## unknown 알수 없음
## uneducated 중졸이하
## high school 고졸
## colleage 학사
## graduate / post-graduate 석사
## Docotrate 박사
```




In [42]:
```
## Unknown 학력을 알수 없는 정도가 1263
## Uneducated 1250


df[df['Attrition_Flag']==0]['Education_Level'].value_counts()
```

Out[42]:
```
Graduate         2641
High School      1707
Unknown          1263
Uneducated       1250
College           859
Post-Graduate     424
Doctorate         356
Name: Education_Level, dtype: int64
```

In [43]:
```
df.groupby(['Education_Level','Income_Category']).count()
```

Out[43]:

| | | CLIENTNUM | Attrition_Flag | Customer_Age | Gender | Deper |
|---|---|---|---|---|---|---|
| **Education_Level** | **Income_Category** | | | | | |
| **College** | **$120K +** | 70 | 70 | 70 | 70 | |
| | 40K−**60K** | 183 | 183 | 183 | 183 | |
| | 60K−**80K** | 132 | 132 | 132 | 132 | |
| | 80K−**120K** | 175 | 175 | 175 | 175 | |

| | | CLIENTNUM | Attrition_Flag | Customer_Age | Gender | Deper |
|---|---|---|---|---|---|---|
| Education_Level | Income_Category | | | | | |
| | Less than $40K | 345 | 345 | 345 | 345 | |
| | Unknown | 108 | 108 | 108 | 108 | |
| Doctorate | $120K + | 37 | 37 | 37 | 37 | |
| | $40K-$60K | 70 | 70 | 70 | 70 | |
| | $60K-$80K | 59 | 59 | 59 | 59 | |
| | $80K-$120K | 57 | 57 | 57 | 57 | |
| | Less than $40K | 158 | 158 | 158 | 158 | |
| | Unknown | 70 | 70 | 70 | 70 | |
| Graduate | $120K + | 204 | 204 | 204 | 204 | |
| | $40K-$60K | 553 | 553 | 553 | 553 | |
| | $60K-$80K | 422 | 422 | 422 | 422 | |
| | $80K-$120K | 478 | 478 | 478 | 478 | |
| | Less than $40K | 1139 | 1139 | 1139 | 1139 | |
| | Unknown | 332 | 332 | 332 | 332 | |
| High School | $120K + | 147 | 147 | 147 | 147 | |
| | $40K-$60K | 355 | 355 | 355 | 355 | |
| | $60K-$80K | 307 | 307 | 307 | 307 | |
| | $80K-$120K | 308 | 308 | 308 | 308 | |
| | Less than $40K | 671 | 671 | 671 | 671 | |
| | Unknown | 225 | 225 | 225 | 225 | |
| Post-Graduate | $120K + | 30 | 30 | 30 | 30 | |
| | $40K-$60K | 111 | 111 | 111 | 111 | |
| | $60K-$80K | 77 | 77 | 77 | 77 | |
| | $80K-$120K | 81 | 81 | 81 | 81 | |
| | Less than $40K | 170 | 170 | 170 | 170 | |
| | Unknown | 47 | 47 | 47 | 47 | |
| Uneducated | $120K + | 119 | 119 | 119 | 119 | |
| | $40K-$60K | 249 | 249 | 249 | 249 | |
| | $60K-$80K | 195 | 195 | 195 | 195 | |
| | $80K-$120K | 217 | 217 | 217 | 217 | |
| | Less than $40K | 522 | 522 | 522 | 522 | |
| | Unknown | 185 | 185 | 185 | 185 | |
| Unknown | $120K + | 120 | 120 | 120 | 120 | |
| | $40K-$60K | 269 | 269 | 269 | 269 | |
| | $60K-$80K | 210 | 210 | 210 | 210 | |
| | $80K-$120K | 219 | 219 | 219 | 219 | |

| Education_Level | Income_Category | CLIENTNUM | Attrition_Flag | Customer_Age | Gender | Deper |
|---|---|---|---|---|---|---|
| | Less than $40K | 556 | 556 | 556 | 556 | |
| | Unknown | 145 | 145 | 145 | 145 | |

In [44]:
```
df['Education_Level'].value_counts()
```

Out[44]:
```
Graduate         3128
High School      2013
Unknown          1519
Uneducated       1487
College          1013
Post-Graduate     516
Doctorate         451
Name: Education_Level, dtype: int64
```

In [45]:
```
df[['Education_Level','Income_Category']].value_counts()
```

Out[45]:
```
Education_Level  Income_Category
Graduate         Less than $40K     1139
High School      Less than $40K      671
Unknown          Less than $40K      556
Graduate         $40K - $60K         553
Uneducated       Less than $40K      522
Graduate         $80K - $120K        478
                 $60K - $80K         422
High School      $40K - $60K         355
College          Less than $40K      345
Graduate         Unknown             332
High School      $80K - $120K        308
                 $60K - $80K         307
Unknown          $40K - $60K         269
Uneducated       $40K - $60K         249
High School      Unknown             225
Unknown          $80K - $120K        219
Uneducated       $80K - $120K        217
Unknown          $60K - $80K         210
Graduate         $120K +             204
Uneducated       $60K - $80K         195
                 Unknown             185
College          $40K - $60K         183
                 $80K - $120K        175
Post-Graduate    Less than $40K      170
Doctorate        Less than $40K      158
High School      $120K +             147
Unknown          Unknown             145
College          $60K - $80K         132
Unknown          $120K +             120
Uneducated       $120K +             119
Post-Graduate    $40K - $60K         111
College          Unknown             108
Post-Graduate    $80K - $120K         81
                 $60K - $80K          77
Doctorate        Unknown              70
```

```
                 $40K - $60K        70
College          $120K +            70
Doctorate        $60K - $80K        59
                 $80K - $120K       57
Post-Graduate    Unknown            47
Doctorate        $120K +            37
Post-Graduate    $120K +            30
dtype: int64
```

In [46]:

```
cnt =
df[['Education_Level','Gender','Income_Category']].groupby(['Ed
 'Income_Category'])
cnt
```

Out[46]: <pandas.core.groupby.generic.DataFrameGroupBy object at 0x17f3b2af0>

In [47]:

```
temp =
df[['Education_Level','Marital_Status','Income_Category']].grou
 'Income_Category'])

temp.count()
```

Out[47]:

| | | Marital_Status |
|---|---|---|
| **Education_Level** | **Income_Category** | |
| **College** | **$120K +** | 70 |
| | 40K−**60K** | 183 |
| | 60K−**80K** | 132 |
| | 80K−**120K** | 175 |
| | **Less than $40K** | 345 |
| | **Unknown** | 108 |
| **Doctorate** | **$120K +** | 37 |
| | 40K−**60K** | 70 |
| | 60K−**80K** | 59 |
| | 80K−**120K** | 57 |
| | **Less than $40K** | 158 |
| | **Unknown** | 70 |
| **Graduate** | **$120K +** | 204 |
| | 40K−**60K** | 553 |
| | 60K−**80K** | 422 |
| | 80K−**120K** | 478 |
| | **Less than $40K** | 1139 |
| | **Unknown** | 332 |

| Education_Level | Income_Category | Marital_Status |
|---|---|---|
| High School | $120K + | 147 |
| | $40K-$60K | 355 |
| | $60K-$80K | 307 |
| | $80K-$120K | 308 |
| | Less than $40K | 671 |
| | Unknown | 225 |
| Post-Graduate | $120K + | 30 |
| | $40K-$60K | 111 |
| | $60K-$80K | 77 |
| | $80K-$120K | 81 |
| | Less than $40K | 170 |
| | Unknown | 47 |
| Uneducated | $120K + | 119 |
| | $40K-$60K | 249 |
| | $60K-$80K | 195 |
| | $80K-$120K | 217 |
| | Less than $40K | 522 |
| | Unknown | 185 |
| Unknown | $120K + | 120 |
| | $40K-$60K | 269 |
| | $60K-$80K | 210 |
| | $80K-$120K | 219 |
| | Less than $40K | 556 |
| | Unknown | 145 |

In [48]:

```
temp =
df[['Education_Level','Marital_Status','Income_Category']].grou
 'Income_Category'])

temp.count()
```

Out[48]:

| Education_Level | Income_Category | Marital_Status |
|---|---|---|
| College | $120K + | 70 |
| | $40K-$60K | 183 |
| | $60K-$80K | 132 |
| | $80K-$120K | 175 |

| | | Marital_Status |
|---|---|---|
| **Education_Level** | **Income_Category** | |
| **Doctorate** | **Less than $40K** | 345 |
| | **Unknown** | 108 |
| | **$120K +** | 37 |
| | $40K-$**60K** | 70 |
| | $60K-$**80K** | 59 |
| | $80K-$**120K** | 57 |
| **Graduate** | **Less than $40K** | 158 |
| | **Unknown** | 70 |
| | **$120K +** | 204 |
| | $40K-$**60K** | 553 |
| | $60K-$**80K** | 422 |
| | $80K-$**120K** | 478 |
| **High School** | **Less than $40K** | 1139 |
| | **Unknown** | 332 |
| | **$120K +** | 147 |
| | $40K-$**60K** | 355 |
| | $60K-$**80K** | 307 |
| | $80K-$**120K** | 308 |
| **Post-Graduate** | **Less than $40K** | 671 |
| | **Unknown** | 225 |
| | **$120K +** | 30 |
| | $40K-$**60K** | 111 |
| | $60K-$**80K** | 77 |
| | $80K-$**120K** | 81 |
| **Uneducated** | **Less than $40K** | 170 |
| | **Unknown** | 47 |
| | **$120K +** | 119 |
| | $40K-$**60K** | 249 |
| | $60K-$**80K** | 195 |
| | $80K-$**120K** | 217 |
| **Unknown** | **Less than $40K** | 522 |
| | **Unknown** | 185 |
| | **$120K +** | 120 |
| | $40K-$**60K** | 269 |
| | $60K-$**80K** | 210 |
| | $80K-$**120K** | 219 |

| | | Marital_Status |
|---|---|---|
| Education_Level | Income_Category | |
| | Less than $40K | 556 |
| | Unknown | 145 |

In [ ]:

```

```

**Education_Level, Marital_Status, Income_Category에서 "Unknown"이라는 결측치가 존재한다.**

In [49]:

```
df["Education_Level"].value_counts()
```

Out[49]:

```
Graduate         3128
High School      2013
Unknown          1519
Uneducated       1487
College          1013
Post-Graduate     516
Doctorate         451
Name: Education_Level, dtype: int64
```

In [50]:

```
df["Marital_Status"].value_counts()
```

Out[50]:

```
Married     4687
Single      3943
Unknown      749
Divorced     748
Name: Marital_Status, dtype: int64
```

In [51]:

```
df["Income_Category"].value_counts()
```

Out[51]:

```
Less than $40K    3561
$40K - $60K       1790
$80K - $120K      1535
$60K - $80K       1402
Unknown           1112
$120K +            727
Name: Income_Category, dtype: int64
```

**"Unknown"**에 대한 처리 방법은

1. "Unknown"도 하나의 category로 해석
2. "Unknown"값이 있는 행을 삭제하거나, 칼럼 자체(피처)를 삭제
3. 모델링을 활용하여 대체
4. 최빈값으로 대체

정해진 답은 없고 여러가지로 시도해보는것이 가장 중요한것 같다.

**결측치로 처리 되지 않았던 Unknown을 np.nan으로 결측치로 처리를 해주고, 대표값(최빈값)으로 결측치를 대체해준다.**

In [52]:

```python
df["Education_Level"].replace({"Unknown":np.nan,
                               "Graduate":0,
                               "Post-Graduate":1,
                               "Uneducated":2,
                               "College":3,
                               "Doctorate":4,
                               "High School":5,
                                },inplace=True)
```

In [53]:

```python
df["Marital_Status"].replace({"Unknown":np.nan,
                               "Married":0,
                               "Single":1,
                               "Divorced":2,
                                },inplace=True)
```

In [54]:

```python
df["Income_Category"].replace({"Unknown":np.nan,
                               "Less than $40K":0,
                               "$40K - $60K":1,
                               "$60K - $80K":2,
                               "$80K - $120K":3,
                               "$120K +":4,
                                },inplace=True)
```

In [55]:

```python
# Unknown이 결측치(np.nan)로 대체 된것을 확인할 수 있다.
df.isnull().sum()
```

Out[55]:

```
CLIENTNUM                      0
Attrition_Flag                 0
Customer_Age                   0
Gender                         0
Dependent_count                0
Education_Level             1519
Marital_Status               749
Income_Category             1112
Card_Category                  0
Months_on_book                 0
Total_Relationship_Count       0
Months_Inactive_12_mon         0
Contacts_Count_12_mon          0
Credit_Limit                   0
Total_Revolving_Bal            0
Avg_Open_To_Buy                0
Total_Amt_Chng_Q4_Q1           0
Total_Trans_Amt                0
Total_Trans_Ct                 0
```

```
Total_Ct_Chng_Q4_Q1          0
Avg_Utilization_Ratio        0
age_bin                      0
dtype: int64
```

In [56]:

```
# 대표값 이용 결측치 대체 모듈
from sklearn.impute import SimpleImputer
# 각 데이터에 사용할 인스턴스 생성
SI_mode =SimpleImputer(strategy = 'most_frequent') # 대표값 중 최빈값으로 결측치를 대체해준다.
# 학습
SI_mode.fit(df)
df = pd.DataFrame(SI_mode.transform(df),
                  columns = df.columns)
```

In [57]:

```
# 결측치가 대체를 확인할 수 있다.
df.isnull().sum()
```

Out[57]:

```
CLIENTNUM                   0
Attrition_Flag              0
Customer_Age                0
Gender                      0
Dependent_count             0
Education_Level             0
Marital_Status              0
Income_Category             0
Card_Category               0
Months_on_book              0
Total_Relationship_Count    0
Months_Inactive_12_mon      0
Contacts_Count_12_mon       0
Credit_Limit                0
Total_Revolving_Bal         0
Avg_Open_To_Buy             0
Total_Amt_Chng_Q4_Q1        0
Total_Trans_Amt             0
Total_Trans_Ct              0
Total_Ct_Chng_Q4_Q1         0
Avg_Utilization_Ratio       0
age_bin                     0
dtype: int64
```

In [ ]:

```

```

In [58]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10127 entries, 0 to 10126
Data columns (total 22 columns):
 #   Column                    Non-Null Count  Dtype
```

```
---  ------                   --------------  -----
 0   CLIENTNUM                10127 non-null  object
 1   Attrition_Flag           10127 non-null  object
 2   Customer_Age             10127 non-null  object
 3   Gender                   10127 non-null  object
 4   Dependent_count          10127 non-null  object
 5   Education_Level          10127 non-null  object
 6   Marital_Status           10127 non-null  object
 7   Income_Category          10127 non-null  object
 8   Card_Category            10127 non-null  object
 9   Months_on_book           10127 non-null  object
 10  Total_Relationship_Count 10127 non-null  object
 11  Months_Inactive_12_mon   10127 non-null  object
 12  Contacts_Count_12_mon    10127 non-null  object
 13  Credit_Limit             10127 non-null  object
 14  Total_Revolving_Bal      10127 non-null  object
 15  Avg_Open_To_Buy          10127 non-null  object
 16  Total_Amt_Chng_Q4_Q1     10127 non-null  object
 17  Total_Trans_Amt          10127 non-null  object
 18  Total_Trans_Ct           10127 non-null  object
 19  Total_Ct_Chng_Q4_Q1      10127 non-null  object
 20  Avg_Utilization_Ratio    10127 non-null  object
 21  age_bin                  10127 non-null  object
dtypes: object(22)
memory usage: 1.7+ MB
```

**중간에 dtype이 모두 object로 바뀌어서 에러가 났다.**

In [59]:

```
df[Numerics]=df[Numerics].astype("float")
df[Labels]=df[Labels].astype("int")
df[Orders]=df[Orders].astype("int")
df["Attrition_Flag"]=df["Attrition_Flag"].astype("int")


# 여기서 모든 데이터 타입을 float으로 바꾸어도 되는 것인지 궁금하다.
```

In [60]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10127 entries, 0 to 10126
Data columns (total 22 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   CLIENTNUM                10127 non-null  object
 1   Attrition_Flag           10127 non-null  int64
 2   Customer_Age             10127 non-null  float64
 3   Gender                   10127 non-null  int64
 4   Dependent_count          10127 non-null  float64
 5   Education_Level          10127 non-null  int64
 6   Marital_Status           10127 non-null  int64
 7   Income_Category          10127 non-null  int64
 8   Card_Category            10127 non-null  int64
 9   Months_on_book           10127 non-null  float64
 10  Total_Relationship_Count 10127 non-null  float64
 11  Months_Inactive_12_mon   10127 non-null  float64
```

```
 12  Contacts_Count_12_mon     10127 non-null  float64
 13  Credit_Limit              10127 non-null  float64
 14  Total_Revolving_Bal       10127 non-null  float64
 15  Avg_Open_To_Buy           10127 non-null  float64
 16  Total_Amt_Chng_Q4_Q1      10127 non-null  float64
 17  Total_Trans_Amt           10127 non-null  float64
 18  Total_Trans_Ct            10127 non-null  float64
 19  Total_Ct_Chng_Q4_Q1       10127 non-null  float64
 20  Avg_Utilization_Ratio     10127 non-null  float64
 21  age_bin                   10127 non-null  object
dtypes: float64(14), int64(6), object(2)
memory usage: 1.7+ MB
```

In [ ]:

```

```

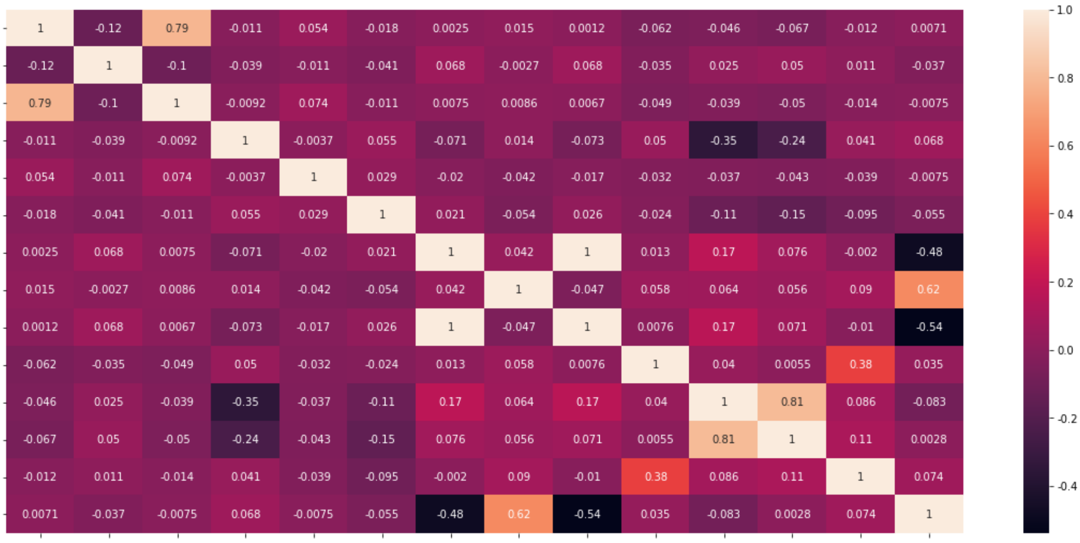
## 4 (3) 상관관계가 높은 피처 제거

In [61]:

```
plt.subplots(figsize=(20,9))
plt.tick_params(axis='x',labelcolor='white')
plt.tick_params(axis='y',labelcolor='white')
sns.heatmap(df[Numerics].corr(),annot = True )
```

Out[61]:

```
<AxesSubplot:>
```




(1) Month_on_book과 Customer_Age의 상관계수는 0.79로 높지만, 도메인 관점에서 완전히 다른 각각의 변수이므로 이 변수들을 삭제하지는 않는다.

(2) Credit_Limit와 Avg_Utilization_Ratio는 상관계수가 0.48이지만 같은 변수에서 파생되었기 때문에 둘중 하나의 변수만 나두고 나머지 하나는 삭제처리가 필요하다.

- 'Avg_Open_To_Buy' = 'Credit_Limit' - 'Total_Revolving_Bal'

- 'Avg_Utilization_Ratio' = 'Total_Revolving_Bal'/ 'Credit_Limit'

- Total_Trans_Amt, Total_Trans_Ct

- 'Total_Amt_Chng_Q4_Q1' , 'Total_Ct_Chng_Q4_Q1'

피처간의 상관관계가 높은 변수들에서 어떤 변수들을 삭제할지를 결정할 때에는 타겟변수(종속변수)와의 상관관계까지 고려하여 타겟과의 상관관계가 높은 변수들을 선택한다.

**Attrition_Flag(범주형)과 수치형 피처 변수들의 상관관계를 파악하기 위해 pointbiserialr로 상관계수를 구한다.**

In [62]:
```
from scipy.stats import pointbiserialr

features =
['Avg_Open_To_Buy','Credit_Limit','Total_Revolving_Bal','Avg_Ut


for feature in features:
    target_feature_corr =
pointbiserialr(df['Attrition_Flag'], df[feature])
    print("Attrition_Flag와",feature,"의 상관관계
는",target_feature_corr,"입니다.")
```

```
Attrition_Flag와 Avg_Open_To_Buy 의 상관관계는 PointbiserialrResult(correlation=-
0.0002850774939378458, pvalue=0.9771160894272927) 입니다.
Attrition_Flag와 Credit_Limit 의 상관관계는 PointbiserialrResult(correlation=-0.0
23872994836163616, pvalue=0.01628535720506713) 입니다.
Attrition_Flag와 Total_Revolving_Bal 의 상관관계는 PointbiserialrResult(correlati
on=-0.2630528831292179, pvalue=6.630148455008398e-160) 입니다.
Attrition_Flag와 Avg_Utilization_Ratio 의 상관관계는 PointbiserialrResult(correla
tion=-0.17841033156175928, pvalue=3.3576893281006543e-73) 입니다.
Attrition_Flag와 Total_Trans_Amt 의 상관관계는 PointbiserialrResult(correlation=-
0.16859838141009204, pvalue=1.8574386555805807e-65) 입니다.
Attrition_Flag와 Total_Trans_Ct 의 상관관계는 PointbiserialrResult(correlation=-
0.37140270118895313, pvalue=0.0) 입니다.
Attrition_Flag와 Total_Amt_Chng_Q4_Q1 의 상관관계는 PointbiserialrResult(correlat
ion=-0.13106284781448055, pvalue=4.836642703419737e-40) 입니다.
Attrition_Flag와 Total_Ct_Chng_Q4_Q1 의 상관관계는 PointbiserialrResult(correlati
on=-0.29005400688091193, pvalue=1.647724784552201e-195) 입니다.
```

**여기서 p-value의 의미 해석하기**

**상관관계가 낮은 Avg_Open_To_Buy, Avg_Utilization_Ratio, Total_Trans_Amt, Total_Amt_Chng_Q4_Q1는 삭제(drop)**

In [63]:
```
df.drop(["CLIENTNUM", # 식별자 삭제
         "Avg_Open_To_Buy",
         "Avg_Utilization_Ratio",
         "Total_Trans_Amt",
         "age_bin",
         "Total_Amt_Chng_Q4_Q1"], axis=1,inplace=True)
```

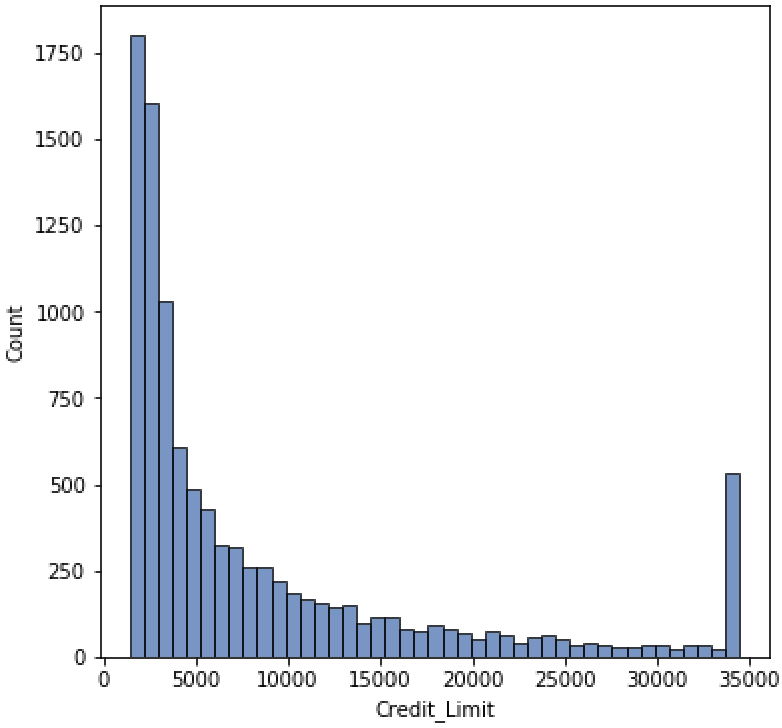
In [ ]:
```

```

## 4 (4) 수치형 변수 log변환 정규화 변환

In [64]:
```
# 수치형 변수 중에 drop시킨 변수를 제외한 나머지 변수들을 모은 리스트를 만든
다.
Numerics =
['Customer_Age','Dependent_count','Months_on_book',
       'Total_Relationship_Count', 'Months_Inactive_12_mon',
       'Contacts_Count_12_mon', 'Credit_Limit',
'Total_Revolving_Bal',
        'Total_Ct_Chng_Q4_Q1',
       'Total_Trans_Ct']
```

In [65]:
```
# Credit_Limit의 분포를 히스토그램으로 시각화
sns.histplot(df["Credit_Limit"])
```

Out[65]: <AxesSubplot:xlabel='Credit_Limit', ylabel='Count'>




In [66]:
```
# 편향이 심한 Credit_Limit은 로그 변환을 해주도록한다.


df["Credit_Limit"]=np.log1p(df["Credit_Limit"])
```

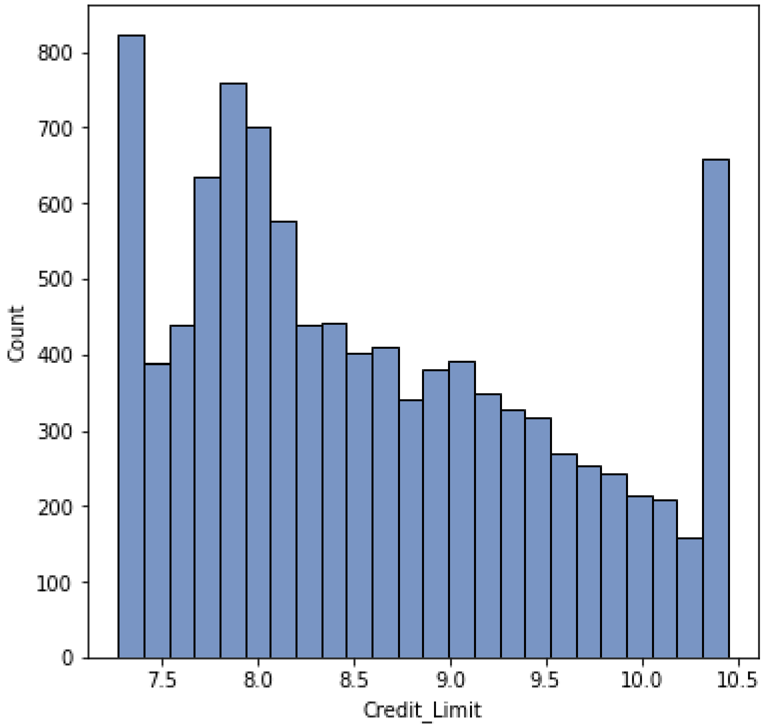
In [67]:
```
df["Credit_Limit"]
```

Out[67]:
```
0        9.448727
1        9.018817
2        8.137103
```

```
3        8.105911
4        8.458928
           ...
10122    8.295049
10123    8.361241
10124    8.596004
10125    8.572060
10126    9.248503
Name: Credit_Limit, Length: 10127, dtype: float64
```

In [68]:
```
sns.histplot(df["Credit_Limit"])
# 편향이 조금 완화 된것을 확인 할 수 있다.
```

Out[68]:
```
<AxesSubplot:xlabel='Credit_Limit', ylabel='Count'>
```




In [ ]:
```

```

In [69]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

for numeric in
['Customer_Age','Credit_Limit','Total_Revolving_Bal','Total_Ct_

    df[numeric] =
scaler.fit_transform(df[numeric].values.reshape(-1,1))
```

In [70]:
```
# 수치형 데이터 표준화 확인
df[Numerics]
```

Out[70]:

| | Customer_Age | Dependent_count | Months_on_book | Total_Relationship_Count | Months_ |
|---|---|---|---|---|---|
| **0** | -0.165406 | 3.0 | 39.0 | 5.0 | |
| **1** | 0.333570 | 5.0 | 44.0 | 6.0 | |
| **2** | 0.583058 | 3.0 | 36.0 | 4.0 | |
| **3** | -0.789126 | 4.0 | 34.0 | 3.0 | |
| **4** | -0.789126 | 3.0 | 21.0 | 5.0 | |
| **...** | ... | ... | ... | ... | |
| **10122** | 0.458314 | 2.0 | 40.0 | 3.0 | |
| **10123** | -0.664382 | 2.0 | 25.0 | 4.0 | |
| **10124** | -0.290150 | 1.0 | 36.0 | 5.0 | |
| **10125** | -2.036565 | 2.0 | 36.0 | 4.0 | |
| **10126** | -0.414894 | 2.0 | 25.0 | 6.0 | |

10127 rows × 10 columns

In [ ]:

In [ ]:

## 5. 모델링

In [ ]:

In [ ]:

### 5 (1)지도학습

## 모델링 >> smote >> 모델링 >> 하이퍼파라미터 수정 >> 모델링 >> 트레이드오프 수정>> 모델링 >> 모델링 선택

In [71]:

```
X = df.iloc[:,1:] # 피처 변수
y = df.iloc[:,0] # 타겟 변수 분리
```

In [72]:

```
X
```

Out[72]:

| | Customer_Age | Gender | Dependent_count | Education_Level | Marital_Status | Income_Ca |
|---|---|---|---|---|---|---|
| **0** | -0.165406 | 0 | 3.0 | 5 | 0 | |
| **1** | 0.333570 | 1 | 5.0 | 0 | 1 | |
| **2** | 0.583058 | 0 | 3.0 | 0 | 0 | |
| **3** | -0.789126 | 1 | 4.0 | 5 | 0 | |

| | Customer_Age | Gender | Dependent_count | Education_Level | Marital_Status | Income_Ca |
|---|---|---|---|---|---|---|
| **4** | -0.789126 | 0 | 3.0 | 2 | 0 | |
| **...** | ... | ... | ... | ... | ... | |
| **10122** | 0.458314 | 0 | 2.0 | 0 | 1 | |
| **10123** | -0.664382 | 0 | 2.0 | 0 | 2 | |
| **10124** | -0.290150 | 1 | 1.0 | 5 | 0 | |
| **10125** | -2.036565 | 0 | 2.0 | 0 | 0 | |
| **10126** | -0.414894 | 1 | 2.0 | 0 | 0 | |

10127 rows × 15 columns

In [73]:

```
y
```

Out[73]:

```
0        0
1        0
2        0
3        0
4        0
        ..
10122    0
10123    1
10124    1
10125    1
10126    1
Name: Attrition_Flag, Length: 10127, dtype: int64
```

In [ ]:

```

```

In [74]:

```
from sklearn.model_selection import train_test_split
# 학습과 데이터 세트로 분리
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size = 0.2, random_state=11, stratify =
X["Income_Category"])
```

In [ ]:

```

```

**분류 분석 평가 지표 함수 - get_clf_eval() 함수**

**분류 분석을 측정하는 방법은 accuracy(정확도), precision(정밀도), recall(재현율) 등 여러가지 지표가 있는데, 타겟 변수의 비율 차이가 크면 정확도는 높게 나올 수 밖에 없어 신뢰할 수밖에 없다.**

**정밀도와 재현율 두 가지를 평가 지표로 활용해야 하는데, 고객 이탈 데이터는 잔류 고객은 이탈 고객으로 판단하여도 별 무리가 없으나, 이탈 고객을 잔류고객으로 판다하였을때는 손해를 보게 된다. 따라서 재현율을 높이는 방향으로 모델링을 진행한다.**

In [75]:

```
from sklearn.metrics import accuracy_score, precision_score,
```

```
recall_score, confusion_matrix, f1_score, roc_auc_score

def get_clf_eval(y_test,pred=None, pred_proba=None):
    confusion = confusion_matrix(y_test, pred)
    accuracy = accuracy_score(y_test, pred)
    precision = precision_score(y_test, pred)
    recall = recall_score(y_test, pred)
    f1 = f1_score(y_test, pred)
    # ROC-AUC 추가
    roc_auc = roc_auc_score(y_test,pred_proba)
    print("오차행렬")
    print(confusion)
    print('정확도:{0:.4f}, 정밀도:{1:.4f}, 재현율:{2:.4f}, F1:{3:.4f}, AUC:{4:.4f}'.format(accuracy, precision, recall, f1, roc_auc))
```

In [ ]:

```
```

**여러 가지 분류 분석**

In [76]:

```
from sklearn.tree import DecisionTreeClassifier # 의사결정 나무
from sklearn.ensemble import RandomForestClassifier # 랜덤 포레스트
from sklearn.linear_model import LogisticRegression # 로지스틱 회귀
from sklearn.neighbors import KNeighborsClassifier # k-최근접이웃
from sklearn.ensemble import GradientBoostingClassifier # GBM
from xgboost import XGBClassifier # XGBM
from sklearn import svm



# 사이킷런 Classifier 클래스 생성
dt_clf = DecisionTreeClassifier(random_state=11)
rf_clf = RandomForestClassifier(random_state=11)
lr_clf = LogisticRegression()
kn_clf = KNeighborsClassifier(n_neighbors=5)
gb_clf = GradientBoostingClassifier(random_state=11)
svm_clf = svm.SVC(probability=True)
```

```
# DecisionTreeClassifier 학습/예측/평가
dt_clf.fit(X_train, y_train)
dt_pred = dt_clf.predict(X_test)
dt_pred_proba = dt_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, dt_pred, dt_pred_proba)

# RandomForestClassifier
rf_clf.fit(X_train, y_train)
rf_pred = rf_clf.predict(X_test)
rf_pred_proba = rf_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, rf_pred, rf_pred_proba)

# LogistRegression
lr_clf.fit(X_train, y_train)
lr_pred = lr_clf.predict(X_test)
lr_pred_proba = lr_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, lr_pred, lr_pred_proba)

# KNeighborsClassifier
kn_clf.fit(X_train, y_train)
kn_pred = kn_clf.predict(X_test)
kn_pred_proba = kn_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, kn_pred, kn_pred_proba)

# GBM
gb_clf.fit(X_train, y_train)
gb_pred = gb_clf.predict(X_test)
gb_pred_proba = gb_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, gb_pred, gb_pred_proba)

# XGBM
xgb_wrapper = XGBClassifier()
xgb_wrapper.fit(X_train, y_train)
w_preds = xgb_wrapper.predict(X_test)
w_pred_proba = xgb_wrapper.predict_proba(X_test)[:,1]
get_clf_eval(y_test, w_preds, w_pred_proba)
```

```
# SVM
svm_clf.fit(X_train, y_train)
svm_pred = svm_clf.predict(X_test)
svm_pred_proba = svm_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, svm_pred, svm_pred_proba)
```

```
오차행렬
[[1589   89]
 [ 131  217]]
정확도:0.8914, 정밀도:0.7092, 재현율:0.6236, F1:0.6636, AUC:0.7853
오차행렬
[[1651   27]
 [ 134  214]]
정확도:0.9205, 정밀도:0.8880, 재현율:0.6149, F1:0.7267, AUC:0.9542
```

```
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/sklearn/linear_model/_
logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
오차행렬
[[1628   50]
 [ 180  168]]
정확도:0.8865, 정밀도:0.7706, 재현율:0.4828, F1:0.5936, AUC:0.8985
오차행렬
[[1649   29]
 [ 208  140]]
정확도:0.8830, 정밀도:0.8284, 재현율:0.4023, F1:0.5416, AUC:0.8490
오차행렬
[[1646   32]
 [ 123  225]]
정확도:0.9235, 정밀도:0.8755, 재현율:0.6466, F1:0.7438, AUC:0.9566
[17:44:45] WARNING: /Users/ktietz/demo/mc3/conda-bld/xgboost-split_16286829080
89/work/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation
metric used with the objective 'binary:logistic' was changed from 'error' to
'logloss'. Explicitly set eval_metric if you'd like to restore the old behavio
r.
```

```
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/xgboost/sklearn.py:88
8: UserWarning: The use of label encoder in XGBClassifier is deprecated and wi
ll be removed in a future release. To remove this warning, do the following:
1) Pass option use_label_encoder=False when constructing XGBClassifier object;
and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ...,
[num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
오차행렬
[[1633   45]
 [ 109  239]]
정확도:0.9240, 정밀도:0.8415, 재현율:0.6868, F1:0.7563, AUC:0.9601
오차행렬
[[1665   13]
```

```
 [ 235  113]]
정확도:0.8776, 정밀도:0.8968, 재현율:0.3247, F1:0.4768, AUC:0.8933
```

**결정 트리(Decision Tree)**

In [77]:

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

# DecisionTree Classifier 생성
dt_clf = DecisionTreeClassifier(random_state=156)

# 학습과 데이터 세트로 분리
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size = 0.2, random_state=11, stratify = y)

# DecisionTreeClassifier 학습
dt_clf.fit(X_train,y_train)

# DecisionTreeClassifier 학습/예측/평가
dt_clf.fit(X_train, y_train)
dt_pred = dt_clf.predict(X_test)
dt_pred_proba = dt_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, dt_pred, dt_pred_proba)



# DecisionTreeClassifier의 하이퍼 파라미터 추출
print('\nDecisionTreeClassifier 기본 하이퍼 파라미터:\n',
dt_clf.get_params())
```

```
오차행렬
[[1601  100]
 [ 106  219]]
정확도:0.8983, 정밀도:0.6865, 재현율:0.6738, F1:0.6801, AUC:0.8075

DecisionTreeClassifier 기본 하이퍼 파라미터:
 {'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': No
ne, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.
0, 'min_impurity_split': None, 'min_samples_leaf': 1, 'min_samples_split': 2,
'min_weight_fraction_leaf': 0.0, 'random_state': 156, 'splitter': 'best'}
```

In [78]:

```python
from imblearn.over_sampling import SMOTE


smote = SMOTE(random_state=0)
X_train_over, y_train_over = smote.fit_resample(X_train,
```

```
y_train)
print('SMOTE 적용 전 학습용 피처/레이블 데이터 세트', X_train.shape, y_train.shape)
print('SMOTE 적용 전 학습용 피처/레이블 데이터 세트', X_train_over.shape, y_train_over.shape)
```

```
SMOTE 적용 전 학습용 피처/레이블 데이터 세트 (8101, 15) (8101,)
SMOTE 적용 전 학습용 피처/레이블 데이터 세트 (13598, 15) (13598,)
```

In [79]:

```
X_train = X_train_over
y_train = y_train_over
```

In [80]:

```
# DecisionTreeClassifier 학습/예측/평가
dt_clf.fit(X_train, y_train)
dt_pred = dt_clf.predict(X_test)
dt_pred_proba = dt_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, dt_pred, dt_pred_proba)

# RandomForestClassifier
rf_clf.fit(X_train, y_train)
rf_pred = rf_clf.predict(X_test)
rf_pred_proba = rf_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, rf_pred, rf_pred_proba)

# LogistRegression
lr_clf.fit(X_train, y_train)
lr_pred = lr_clf.predict(X_test)
lr_pred_proba = lr_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, lr_pred, lr_pred_proba)

# KNeighborsClassifier
kn_clf.fit(X_train, y_train)
kn_pred = kn_clf.predict(X_test)
kn_pred_proba = kn_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, kn_pred, kn_pred_proba)

# GBM
gb_clf.fit(X_train, y_train)
gb_pred = gb_clf.predict(X_test)
gb_pred_proba = gb_clf.predict_proba(X_test)[:,1]
```

```
get_clf_eval(y_test, gb_pred, gb_pred_proba)

# XGBM
xgb_wrapper = XGBClassifier()
xgb_wrapper.fit(X_train, y_train)
w_preds = xgb_wrapper.predict(X_test)
w_pred_proba = xgb_wrapper.predict_proba(X_test)[:,1]
get_clf_eval(y_test, w_preds, w_pred_proba)

# SVM
svm_clf.fit(X_train, y_train)
svm_pred = svm_clf.predict(X_test)
svm_pred_proba = svm_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, svm_pred, svm_pred_proba)
```

```
오차행렬
[[1555  146]
 [  86  239]]
정확도:0.8855, 정밀도:0.6208, 재현율:0.7354, F1:0.6732, AUC:0.8248
오차행렬
[[1643   58]
 [  75  250]]
정확도:0.9344, 정밀도:0.8117, 재현율:0.7692, F1:0.7899, AUC:0.9549
```

```
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/sklearn/linear_model/_
logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
오차행렬
[[1421  280]
 [  81  244]]
정확도:0.8218, 정밀도:0.4656, 재현율:0.7508, F1:0.5748, AUC:0.8793
오차행렬
[[1383  318]
 [  81  244]]
정확도:0.8031, 정밀도:0.4342, 재현율:0.7508, F1:0.5502, AUC:0.8535
오차행렬
[[1625   76]
 [  70  255]]
정확도:0.9279, 정밀도:0.7704, 재현율:0.7846, F1:0.7774, AUC:0.9552
[17:44:53] WARNING: /Users/ktietz/demo/mc3/conda-bld/xgboost-split_16286829080
89/work/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation
metric used with the objective 'binary:logistic' was changed from 'error' to
'logloss'. Explicitly set eval_metric if you'd like to restore the old behavio
r.
```

```
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/xgboost/sklearn.py:88
8: UserWarning: The use of label encoder in XGBClassifier is deprecated and wi
ll be removed in a future release. To remove this warning, do the following:
1) Pass option use_label_encoder=False when constructing XGBClassifier object;
and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ...,
[num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
오차행렬
[[1653   48]
 [  75  250]]
정확도:0.9393, 정밀도:0.8389, 재현율:0.7692, F1:0.8026, AUC:0.9613
오차행렬
[[1441  260]
 [  78  247]]
정확도:0.8332, 정밀도:0.4872, 재현율:0.7600, F1:0.5938, AUC:0.8825
```

In [81]:

```python
from sklearn.ensemble import VotingClassifier


vo_clf = VotingClassifier( estimators=[('xgb',xgb_wrapper),
('SVM',svm_clf)],voting='soft')
vo_clf.fit(X_train_over,y_train_over)
vo_pred = vo_clf.predict(X_test)
vo_pred_proba = vo_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, vo_pred, vo_pred_proba)
```

```
[17:45:05] WARNING: /Users/ktietz/demo/mc3/conda-bld/xgboost-split_16286829080
89/work/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation
metric used with the objective 'binary:logistic' was changed from 'error' to
'logloss'. Explicitly set eval_metric if you'd like to restore the old behavio
r.
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/xgboost/sklearn.py:88
8: UserWarning: The use of label encoder in XGBClassifier is deprecated and wi
ll be removed in a future release. To remove this warning, do the following:
1) Pass option use_label_encoder=False when constructing XGBClassifier object;
and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ...,
[num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
오차행렬
[[1630   71]
 [  74  251]]
정확도:0.9284, 정밀도:0.7795, 재현율:0.7723, F1:0.7759, AUC:0.9373
```

In [ ]:

```python

```

In [82]:

```python
from sklearn.ensemble import VotingClassifier
n1=('LR',lr_clf)
n2=('dt',dt_clf)
n3=('kn',kn_clf)
n4=('rf',rf_clf)
```

```
n5=('xgb',xgb_wrapper)


vo_clf = VotingClassifier( estimators=
[n1,n2,n3,n4,n5],voting='soft')
vo_clf.fit(X_train_over,y_train_over)
vo_pred = vo_clf.predict(X_test)
vo_pred_proba = vo_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, vo_pred, vo_pred_proba)
```

```
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/sklearn/linear_model/_
logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/xgboost/sklearn.py:88
8: UserWarning: The use of label encoder in XGBClassifier is deprecated and wi
ll be removed in a future release. To remove this warning, do the following:
1) Pass option use_label_encoder=False when constructing XGBClassifier object;
and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ...,
[num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
[17:45:20] WARNING: /Users/ktietz/demo/mc3/conda-bld/xgboost-split_16286829080
89/work/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation
metric used with the objective 'binary:logistic' was changed from 'error' to
'logloss'. Explicitly set eval_metric if you'd like to restore the old behavio
r.
오차행렬
[[1610   91]
 [  71  254]]
정확도:0.9200, 정밀도:0.7362, 재현율:0.7815, F1:0.7582, AUC:0.9407
```

In [83]:

```
vo_clf = VotingClassifier( estimators=[n1,n2,n3,n4#,n5
                                       ],voting='soft')
vo_clf.fit(X_train_over,y_train_over)
vo_pred = vo_clf.predict(X_test)
vo_pred_proba = vo_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, vo_pred, vo_pred_proba)
```

```
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/sklearn/linear_model/_
logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
```

```
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
오차행렬
[[1576  125]
 [  70  255]]
정확도:0.9038, 정밀도:0.6711, 재현율:0.7846, F1:0.7234, AUC:0.9308
```

In [84]:

```python
vo_clf = VotingClassifier( estimators=[n1,n2,n3,n5
                                      ],voting='soft')
vo_clf.fit(X_train_over,y_train_over)
vo_pred = vo_clf.predict(X_test)
vo_pred_proba = vo_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, vo_pred, vo_pred_proba)
```

```
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/sklearn/linear_model/_
logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/xgboost/sklearn.py:88
8: UserWarning: The use of label encoder in XGBClassifier is deprecated and wi
ll be removed in a future release. To remove this warning, do the following:
1) Pass option use_label_encoder=False when constructing XGBClassifier object;
and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ...,
[num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
[17:45:25] WARNING: /Users/ktietz/demo/mc3/conda-bld/xgboost-split_16286829080
89/work/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation
metric used with the objective 'binary:logistic' was changed from 'error' to
'logloss'. Explicitly set eval_metric if you'd like to restore the old behavio
r.
오차행렬
[[1593  108]
 [  68  257]]
정확도:0.9131, 정밀도:0.7041, 재현율:0.7908, F1:0.7449, AUC:0.9331
```

In [85]:

```python
vo_clf = VotingClassifier( estimators=[n1,n2,n5,n4
                                      ],voting='soft')
vo_clf.fit(X_train_over,y_train_over)
vo_pred = vo_clf.predict(X_test)
vo_pred_proba = vo_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, vo_pred, vo_pred_proba)
```

```
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/sklearn/linear_model/_
logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status=1):
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/xgboost/sklearn.py:88
8: UserWarning: The use of label encoder in XGBClassifier is deprecated and wi
ll be removed in a future release. To remove this warning, do the following:
1) Pass option use_label_encoder=False when constructing XGBClassifier object;
and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ...,
[num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
[17:45:26] WARNING: /Users/ktietz/demo/mc3/conda-bld/xgboost-split_16286829080
89/work/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation
metric used with the objective 'binary:logistic' was changed from 'error' to
'logloss'. Explicitly set eval_metric if you'd like to restore the old behavio
r.
오차행렬
[[1609   92]
 [  72  253]]
정확도:0.9191, 정밀도:0.7333, 재현율:0.7785, F1:0.7552, AUC:0.9458
```

In [86]:

```python
vo_clf = VotingClassifier( estimators=[n1,n5,n3,n4
                                       ],voting='soft')
vo_clf.fit(X_train_over,y_train_over)
vo_pred = vo_clf.predict(X_test)
vo_pred_proba = vo_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, vo_pred, vo_pred_proba)
```

```
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/sklearn/linear_model/_
logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/xgboost/sklearn.py:88
8: UserWarning: The use of label encoder in XGBClassifier is deprecated and wi
ll be removed in a future release. To remove this warning, do the following:
1) Pass option use_label_encoder=False when constructing XGBClassifier object;
and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ...,
[num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
[17:45:29] WARNING: /Users/ktietz/demo/mc3/conda-bld/xgboost-split_16286829080
89/work/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation
metric used with the objective 'binary:logistic' was changed from 'error' to
'logloss'. Explicitly set eval_metric if you'd like to restore the old behavio
r.
오차행렬
```

```
[[1616   85]
 [  70  255]]
정확도:0.9235, 정밀도:0.7500, 재현율:0.7846, F1:0.7669, AUC:0.9393
```

In [87]:

```
vo_clf = VotingClassifier( estimators=[n5,n2,n3,n4
                                      ],voting='soft')
vo_clf.fit(X_train_over,y_train_over)
vo_pred = vo_clf.predict(X_test)
vo_pred_proba = vo_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, vo_pred, vo_pred_proba)
```

```
[17:45:31] WARNING: /Users/ktietz/demo/mc3/conda-bld/xgboost-split_16286829080
89/work/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation
metric used with the objective 'binary:logistic' was changed from 'error' to
'logloss'. Explicitly set eval_metric if you'd like to restore the old behavio
r.
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/xgboost/sklearn.py:88
8: UserWarning: The use of label encoder in XGBClassifier is deprecated and wi
ll be removed in a future release. To remove this warning, do the following:
1) Pass option use_label_encoder=False when constructing XGBClassifier object;
and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ...,
[num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
오차행렬
[[1618   83]
 [  69  256]]
정확도:0.9250, 정밀도:0.7552, 재현율:0.7877, F1:0.7711, AUC:0.9441
```

In [88]:

```
vo_clf = VotingClassifier( estimators=[n1,n2,n3
                                      ],voting='soft')
vo_clf.fit(X_train_over,y_train_over)
vo_pred = vo_clf.predict(X_test)
vo_pred_proba = vo_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, vo_pred, vo_pred_proba)
```

```
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/sklearn/linear_model/_
logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
오차행렬
[[1556  145]
 [  75  250]]
정확도:0.8914, 정밀도:0.6329, 재현율:0.7692, F1:0.6944, AUC:0.9140
```

In [89]:

```
vo_clf = VotingClassifier( estimators=[n1,n2,n4
```

```
                                        ],voting='soft')
vo_clf.fit(X_train_over,y_train_over)
vo_pred = vo_clf.predict(X_test)
vo_pred_proba = vo_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, vo_pred, vo_pred_proba)
```

```
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/sklearn/linear_model/_
logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
오차행렬
[[1584  117]
 [  76  249]]
정확도:0.9047, 정밀도:0.6803, 재현율:0.7662, F1:0.7207, AUC:0.9341
```

In [90]:

```
vo_clf = VotingClassifier( estimators=[n1,n2,n5
                                        ],voting='soft')
vo_clf.fit(X_train_over,y_train_over)
vo_pred = vo_clf.predict(X_test)
vo_pred_proba = vo_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, vo_pred, vo_pred_proba)
```

```
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/sklearn/linear_model/_
logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/xgboost/sklearn.py:88
8: UserWarning: The use of label encoder in XGBClassifier is deprecated and wi
ll be removed in a future release. To remove this warning, do the following:
1) Pass option use_label_encoder=False when constructing XGBClassifier object;
and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ...,
[num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
[17:45:37] WARNING: /Users/ktietz/demo/mc3/conda-bld/xgboost-split_16286829080
89/work/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation
metric used with the objective 'binary:logistic' was changed from 'error' to
'logloss'. Explicitly set eval_metric if you'd like to restore the old behavio
r.
오차행렬
[[1598  103]
```

```
 [  71  254]]
정확도:0.9141, 정밀도:0.7115, 재현율:0.7815, F1:0.7449, AUC:0.9366
```

In [91]:

```
vo_clf = VotingClassifier( estimators=[n2,n3,n4
                                      ],voting='soft')
vo_clf.fit(X_train_over,y_train_over)
vo_pred = vo_clf.predict(X_test)
vo_pred_proba = vo_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, vo_pred, vo_pred_proba)
```

```
오차행렬
[[1589  112]
 [  69  256]]
정확도:0.9107, 정밀도:0.6957, 재현율:0.7877, F1:0.7388, AUC:0.9323
```

In [92]:

```
vo_clf = VotingClassifier( estimators=[n2,n3,n5
                                      ],voting='soft')
vo_clf.fit(X_train_over,y_train_over)
vo_pred = vo_clf.predict(X_test)
vo_pred_proba = vo_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, vo_pred, vo_pred_proba)
```

```
[17:45:39] WARNING: /Users/ktietz/demo/mc3/conda-bld/xgboost-split_16286829080
89/work/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation
metric used with the objective 'binary:logistic' was changed from 'error' to
'logloss'. Explicitly set eval_metric if you'd like to restore the old behavio
r.
```

```
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/xgboost/sklearn.py:88
8: UserWarning: The use of label encoder in XGBClassifier is deprecated and wi
ll be removed in a future release. To remove this warning, do the following:
1) Pass option use_label_encoder=False when constructing XGBClassifier object;
and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ...,
[num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
오차행렬
[[1606   95]
 [  71  254]]
정확도:0.9181, 정밀도:0.7278, 재현율:0.7815, F1:0.7537, AUC:0.9384
```

In [93]:

```
vo_clf = VotingClassifier( estimators=[n3,n4,n5
                                      ],voting='soft')
vo_clf.fit(X_train_over,y_train_over)
vo_pred = vo_clf.predict(X_test)
vo_pred_proba = vo_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, vo_pred, vo_pred_proba)
```

```
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/xgboost/sklearn.py:88
8: UserWarning: The use of label encoder in XGBClassifier is deprecated and wi
```

```
ll be removed in a future release. To remove this warning, do the following:
1) Pass option use_label_encoder=False when constructing XGBClassifier object;
and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ...,
[num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
[17:45:41] WARNING: /Users/ktietz/demo/mc3/conda-bld/xgboost-split_16286829080
89/work/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation
metric used with the objective 'binary:logistic' was changed from 'error' to
'logloss'. Explicitly set eval_metric if you'd like to restore the old behavio
r.
오차행렬
[[1635   66]
 [  71  254]]
정확도:0.9324, 정밀도:0.7937, 재현율:0.7815, F1:0.7876, AUC:0.9447
```

In [94]:

```
vo_clf = VotingClassifier( estimators=[n1,n2
                                      ],voting='soft')
vo_clf.fit(X_train_over,y_train_over)
vo_pred = vo_clf.predict(X_test)
vo_pred_proba = vo_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, vo_pred, vo_pred_proba)
```

```
오차행렬
[[1555  146]
 [  86  239]]
정확도:0.8855, 정밀도:0.6208, 재현율:0.7354, F1:0.6732, AUC:0.9070
```

```
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/sklearn/linear_model/_
logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
```

In [95]:

```
vo_clf = VotingClassifier( estimators=[n1,n3
                                      ],voting='soft')
vo_clf.fit(X_train_over,y_train_over)
vo_pred = vo_clf.predict(X_test)
vo_pred_proba = vo_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, vo_pred, vo_pred_proba)
```

```
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/sklearn/linear_model/_
logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
```

```
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
오차행렬
[[1444  257]
 [  75  250]]
정확도:0.8361, 정밀도:0.4931, 재현율:0.7692, F1:0.6010, AUC:0.8922
```

In [96]:

```
vo_clf = VotingClassifier( estimators=[n1,n4
                                      ],voting='soft')
vo_clf.fit(X_train_over,y_train_over)
vo_pred = vo_clf.predict(X_test)
vo_pred_proba = vo_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, vo_pred, vo_pred_proba)
```

```
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/sklearn/linear_model/_
logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
오차행렬
[[1561  140]
 [  73  252]]
정확도:0.8949, 정밀도:0.6429, 재현율:0.7754, F1:0.7029, AUC:0.9321
```

In [97]:

```
vo_clf = VotingClassifier( estimators=[n1,n5
                                      ],voting='soft')
vo_clf.fit(X_train_over,y_train_over)
vo_pred = vo_clf.predict(X_test)
vo_pred_proba = vo_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, vo_pred, vo_pred_proba)
```

```
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/sklearn/linear_model/_
logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/xgboost/sklearn.py:88
8: UserWarning: The use of label encoder in XGBClassifier is deprecated and wi
ll be removed in a future release. To remove this warning, do the following:
1) Pass option use_label_encoder=False when constructing XGBClassifier object;
```

```
and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ...,
[num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
[17:45:46] WARNING: /Users/ktietz/demo/mc3/conda-bld/xgboost-split_16286829080
89/work/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation
metric used with the objective 'binary:logistic' was changed from 'error' to
'logloss'. Explicitly set eval_metric if you'd like to restore the old behavio
r.
오차행렬
[[1631   70]
 [  74  251]]
정확도:0.9289, 정밀도:0.7819, 재현율:0.7723, F1:0.7771, AUC:0.9375
```

In [98]:

```
vo_clf = VotingClassifier( estimators=[n2,n3
                                       ],voting='soft')
vo_clf.fit(X_train_over,y_train_over)
vo_pred = vo_clf.predict(X_test)
vo_pred_proba = vo_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, vo_pred, vo_pred_proba)
```

```
오차행렬
[[1590  111]
 [  95  230]]
정확도:0.8983, 정밀도:0.6745, 재현율:0.7077, F1:0.6907, AUC:0.9013
```

In [99]:

```
vo_clf = VotingClassifier( estimators=[n2,n4
                                       ],voting='soft')
vo_clf.fit(X_train_over,y_train_over)
vo_pred = vo_clf.predict(X_test)
vo_pred_proba = vo_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, vo_pred, vo_pred_proba)
```

```
오차행렬
[[1555  146]
 [  86  239]]
정확도:0.8855, 정밀도:0.6208, 재현율:0.7354, F1:0.6732, AUC:0.9471
```

In [100…

```
vo_clf = VotingClassifier( estimators=[n2,n5
                                       ],voting='soft')
vo_clf.fit(X_train_over,y_train_over)
vo_pred = vo_clf.predict(X_test)
vo_pred_proba = vo_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, vo_pred, vo_pred_proba)
```

```
[17:45:49] WARNING: /Users/ktietz/demo/mc3/conda-bld/xgboost-split_16286829080
89/work/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation
metric used with the objective 'binary:logistic' was changed from 'error' to
'logloss'. Explicitly set eval_metric if you'd like to restore the old behavio
r.
```

```
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/xgboost/sklearn.py:88
8: UserWarning: The use of label encoder in XGBClassifier is deprecated and wi
ll be removed in a future release. To remove this warning, do the following:
1) Pass option use_label_encoder=False when constructing XGBClassifier object;
and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ...,
[num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
오차행렬
[[1555  146]
 [  86  239]]
정확도:0.8855, 정밀도:0.6208, 재현율:0.7354, F1:0.6732, AUC:0.9504
```

In [101…

```
vo_clf = VotingClassifier( estimators=[n3,n4
                                     ],voting='soft')
vo_clf.fit(X_train_over,y_train_over)
vo_pred = vo_clf.predict(X_test)
vo_pred_proba = vo_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, vo_pred, vo_pred_proba)
```

```
오차행렬
[[1556  145]
 [  74  251]]
정확도:0.8919, 정밀도:0.6338, 재현율:0.7723, F1:0.6963, AUC:0.9274
```

In [102…

```
vo_clf = VotingClassifier( estimators=[n3,n5
                                     ],voting='soft')
vo_clf.fit(X_train_over,y_train_over)
vo_pred = vo_clf.predict(X_test)
vo_pred_proba = vo_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, vo_pred, vo_pred_proba)
```

```
[17:45:51] WARNING: /Users/ktietz/demo/mc3/conda-bld/xgboost-split_16286829080
89/work/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation
metric used with the objective 'binary:logistic' was changed from 'error' to
'logloss'. Explicitly set eval_metric if you'd like to restore the old behavio
r.
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/xgboost/sklearn.py:88
8: UserWarning: The use of label encoder in XGBClassifier is deprecated and wi
ll be removed in a future release. To remove this warning, do the following:
1) Pass option use_label_encoder=False when constructing XGBClassifier object;
and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ...,
[num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
오차행렬
[[1587  114]
 [  72  253]]
정확도:0.9082, 정밀도:0.6894, 재현율:0.7785, F1:0.7312, AUC:0.9364
```

In [103…

```
vo_clf = VotingClassifier( estimators=[n4,n5
                                     ],voting='soft')
```

```
vo_clf.fit(X_train_over,y_train_over)
vo_pred = vo_clf.predict(X_test)
vo_pred_proba = vo_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, vo_pred, vo_pred_proba)
```

```
/Users/heejinkim/miniforge3/lib/python3.9/site-packages/xgboost/sklearn.py:88
8: UserWarning: The use of label encoder in XGBClassifier is deprecated and wi
ll be removed in a future release. To remove this warning, do the following:
1) Pass option use_label_encoder=False when constructing XGBClassifier object;
and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ...,
[num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
[17:45:53] WARNING: /Users/ktietz/demo/mc3/conda-bld/xgboost-split_16286829080
89/work/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation
metric used with the objective 'binary:logistic' was changed from 'error' to
'logloss'. Explicitly set eval_metric if you'd like to restore the old behavio
r.
오차행렬
[[1655   46]
 [  77  248]]
정확도:0.9393, 정밀도:0.8435, 재현율:0.7631, F1:0.8013, AUC:0.9607
```

In [104…

```
----
```

```
  File "/var/folders/zz/lwjzp_r130b4y7w9qcwwkfhr0000gn/T/ipykernel_7191/213349
6677.py", line 1
    ----
        ^
SyntaxError: invalid syntax
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```

```

결정 트리의 트리깊이(Tree Depth)가 예측 정확도에 주는 영향을 살펴보자. 결정 트리의 경우 분류를 위해 리프 노드(클래스 결정 노드)가 될 수 있는 적합한 수준이 될 때까지 계속해서 트리의 분할을 수행하면서 깊이가 깊어진다. GridSearchCV를 이용해 사이킷런 결정 트리의 깊이를 조절할 수 있는 하이퍼 파라미터인 max_depth 값을 변화시키면서 예측 성능을 확인한다. max_depth를 6,8,10,12,16,20,24로 늘려가면서 예측 성능을 측정한다.

In [ ]:

```python
from sklearn.model_selection import GridSearchCV

params = {
    'max_depth':[6,8,10,12,16,20,24]
}

grid_cv = GridSearchCV(dt_clf, param_grid=params, scoring='recall', cv=5, verbose=1)
grid_cv.fit(X_train, y_train)
print('GridSearchCV 최고 평균 재현율 수치 : {0:.4f}'.format(grid_cv.best_score_))
print('GridSearchCV 최적 하이퍼 파라미터 :',grid_cv.best_params_)
```

max_depth가 20일때 재현율이 0.687380으로 가장 높다.

In [ ]:

```python
best_dt_clf = grid_cv.best_estimator_
pred1 = best_dt_clf.predict(X_test)
pred_proba1 = best_dt_clf.predict_proba(X_test)[:,1]
dt_results = get_clf_eval(y_test, pred1, pred_proba1)
```

하이퍼 파라미터 튜닝을 했지만, 전후 차이가 크지 않다.

In [ ]:

```

```

In [ ]:

```python
from sklearn.tree import export_graphviz

# export_graphviz()의 호출 결과로 out_file로 지정된 tree.dot 파일을 생성함
export_graphviz(best_dt_clf, out_file="tree.dot", class_names=['Existiong_Customer','Attrited_Customer'], feature_names = X.keys(), impurity=True, filled=True)
```

In [ ]:

```python
import graphviz
```

```
# 위에서 생성된 tree.dot 파일을 Graphviz가 읽어서 주피터 노트북상에서 시각
화
with open('tree.dot') as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
```

In [ ]:

```
import seaborn as sns
import numpy as np
%matplotlib inline

# feature importance 추출
print("Feature
importances:\n{0}".format(np.round(best_dt_clf.feature_importa
 3)))

# feature 별 importance 매핑
for name, value in zip(X.keys(),
best_dt_clf.feature_importances_):
    print('{0}:{1:.3f}'.format(name,value))

# feature importance를 column 별로 시각화하기
sns.barplot(x=dt_clf.feature_importances_, y=X.keys())
```

In [ ]:

```
```

**GBM**

In [ ]:

```
from sklearn.ensemble import GradientBoostingClassifier
import time

# GBM 수행 시간 측정. 시작 시간 설정
start_time = time.time()

gb_clf = GradientBoostingClassifier(random_state=0)
gb_clf.fit(X_train, y_train)
gb_pred = gb_clf.predict(X_test)
gb_pred_proba = gb_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, gb_pred, gb_pred_proba)
```

```
print("GBM 수행 시간 :{0:.1F}초".format(time.time()-
start_time))
```

In [ ]:
```
import xgboost as xgb
from xgboost import plot_importance

dtrain = xgb.DMatrix(data=X_train, label=y_train)
dtest = xgb.DMatrix(data=X_test, label=y_test)
```

In [ ]:
```
params = {
    'max_depth':3,
    'objective':'binary:logistic',
    'eval_metric':'logloss',
    'early_stoppings':100
}
num_rounds = 400
```

In [ ]:
```
# train 데이터 세트는 'train', evaluation(test) 데이터 세트는
'eval'로 명기합니다.
wlist = [(dtrain, 'train'),(dtest,'eval')]
# 하이퍼 파라미터와 early stopping 파라미터를 train() 함수의 파라미터로
전달
xgb_model = xgb.train(params = params, dtrain=dtrain,
num_boost_round=num_rounds,\
                     early_stopping_rounds=100, evals=wlist)
```

In [ ]:
```
pred_probs = xgb_model.predict(dtest)
print('predict() 수행 결과값을 10개만 표시, 예측 확률값으로 표시됨')
print(np.round(pred_probs[:10],3))

# 예측 확률이 0.5 보다 크면 1, 그렇지 않으면 0 으로 예측값 결정해 리스트 객
체인 preds에 저장
preds =[1 if x > 0.5 else 0 for x in pred_probs]
print('예측값 10개만 표시:', preds[:10])
```

In [ ]:
```
get_clf_eval(y_test, preds, pred_probs)
```

In [ ]:
```
from xgboost import plot_importance
```

```
import matplotlib.pyplot as plt
%matplotlib inline

fig, ax = plt.subplots(figsize=(10,12))
plot_importance(xgb_model, ax=ax)
```

In [ ]:

```
# 사이킷런 래퍼 XGBoost 클래스인 XGBClassifier 임포트
from xgboost import XGBClassifier

xgb_wrapper = XGBClassifier(n_estimators=400,
learning_rate=0.1, max_depth=3)
xgb_wrapper.fit(X_train, y_train)
w_preds = xgb_wrapper.predict(X_test)
w_pred_proba = xgb_wrapper.predict_proba(X_test)[:,1]

get_clf_eval(y_test, w_preds, w_pred_proba)
```

In [ ]:

```
# 사이킷런 래퍼 XGBoost 클래스인 XGBClassifier 임포트
from xgboost import XGBClassifier

xgb_wrapper = XGBClassifier(n_estimators=400,
learning_rate=0.1, max_depth=3)
evals =[(X_test, y_test)]
xgb_wrapper.fit(X_train, y_train, early_stopping_rounds=100,
eval_metric="logloss",
               eval_set=evals, verbose=True)
ws100_preds = xgb_wrapper.predict(X_test)
ws100_pred_proba = xgb_wrapper.predict_proba(X_test)[:,1]

get_clf_eval(y_test, ws100_preds, ws100_pred_proba)
```

In [ ]:

```
# 사이킷런 래퍼 XGBoost 클래스인 XGBClassifier 임포트
from xgboost import XGBClassifier

xgb_wrapper = XGBClassifier(n_estimators=400,
learning_rate=0.1, max_depth=3)
evals =[(X_test, y_test)]
xgb_wrapper.fit(X_train, y_train, early_stopping_rounds=10,
```

```
eval_metric="logloss",
               eval_set=evals, verbose=True)
ws10_preds = xgb_wrapper.predict(X_test)
ws10_pred_proba = xgb_wrapper.predict_proba(X_test)[:,1]

get_clf_eval(y_test, ws10_preds, ws10_pred_proba)
```

In [ ]:
```
from xgboost import plot_importance
import matplotlib.pyplot as plt
%matplotlib inline

fig, ax = plt.subplots(figsize=(10,12))
# 사이킷런 Wrapper 클래스를 입력해도 무방
plot_importance(xgb_wrapper, ax=ax)
```

**SMOTE 오버샘플링 적용 후 모델 학습/예측/평가**

In [ ]:
```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=0)
X_train_over, y_train_over = smote.fit_resample(X_train,
y_train)
print('SMOTE 적용 전 학습용 피처/레이블 데이터 세트', X_train.shape,
y_train.shape)
print('SMOTE 적용 전 학습용 피처/레이블 데이터 세트',
X_train_over.shape, y_train_over.shape)
```

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier # 의사결정 나무
from sklearn.ensemble import RandomForestClassifier # 랜덤 포레
스트
from sklearn.linear_model import LogisticRegression # 로지스틱
회귀
from sklearn.neighbors import KNeighborsClassifier # k-최근접이
웃

# 사이킷런 Classifier 클래스 생성
dt_clf = DecisionTreeClassifier(random_state=11)
rf_clf = RandomForestClassifier(random_state=11)
lr_clf = LogisticRegression()
```

```
# DecisionTreeClassifier 학습/예측/평가
dt_clf.fit(X_train_over, y_train_over)
dt_pred = dt_clf.predict(X_test)
dt_pred_proba = dt_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test, dt_pred, dt_pred_proba)

# RandomForestClassifier
rf_clf.fit(X_train_over, y_train_over)
rf_pred = rf_clf.predict(X_test)
rf_pred_proba = rf_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test.values, rf_pred, rf_pred_proba)

# LogistRegression
lr_clf.fit(X_train_over, y_train_over)
lr_pred = lr_clf.predict(X_test)
lr_pred_proba = lr_clf.predict_proba(X_test)[:,1]
get_clf_eval(y_test.values, lr_pred, lr_pred_proba)
```

In [ ]:

```
# 사이킷런 래퍼 XGBoost 클래스인 XGBClassifier 임포트
from xgboost import XGBClassifier

xgb_wrapper = XGBClassifier(n_estimators=600,
learning_rate=0.1, max_depth=3)
xgb_wrapper.fit(X_train_over, y_train_over)
w_preds = xgb_wrapper.predict(X_test)
w_pred_proba = xgb_wrapper.predict_proba(X_test)[:,1]

get_clf_eval(y_test, w_preds, w_pred_proba)
```

In [ ]:

```
preds = w_preds
pred_proba = w_pred_proba
```

In [ ]:

```
from sklearn.preprocessing import Binarizer

binarizer = Binarizer(threshold=1.0)
from sklearn.preprocessing import Binarizer
```

```
binarizer = Binarizer(threshold=1.0)
from sklearn.metrics import accuracy_score, precision_score ,
recall_score , confusion_matrix

def get_clf_eval(y_test, pred) :
    confusion = confusion_matrix(y_test, pred)  # 오차행렬
    accuracy = accuracy_score(y_test, pred)     # 정확도
    precision = precision_score(y_test, pred)    # 정밀도
    recall = recall_score(y_test, pred)         # 재현율

    print('오차행렬')
    print(confusion)
    print('정확도: {0:.3f}, 정밀도: {1:.3f}, 재현율:
{2:.3f}'.format(accuracy, precision, recall))

from sklearn.preprocessing import Binarizer

# Binarizer의 threshold 설정값. 분류 결정 임계값 = 0.5로 설정.
c_threshold = 0.5

# predict_proba() 반환값이 [0확률, 1확률]로 반환 - positive 클래스 컬
럼만 추출해서 Binarizer를 적용
pred_proba_1 = pred_proba.reshape(-1,1)

bina = Binarizer(threshold=c_threshold).fit(pred_proba_1)
custom_predict = bina.transform(pred_proba_1)

get_clf_eval(y_test,custom_predict)
```

In [ ]:

```
pred_proba
```

In [ ]:

In [ ]:

In [ ]:

앙상블 학습

In [ ]:
```
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
```

In [ ]:
```

```

## 5 (2) 군집화

## KMeans 클래스

class sklearn.cluster.KMeans(n_cluster=2, init='k-means++', n_init=10, max_iter=3--. tol=0.0001, precompute_distances='auto',verbose=0,random_state=None, copy_x=True, n_jobs=1, algorithm='auto')

In [ ]:
```
from sklearn.preprocessing import scale
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=2, init='k-
means++',max_iter=300,random_state=0)
kmeans.fit(X)
```

In [ ]:
```
X["target"]=y
X["cluster"]=kmeans.labels_
```

In [ ]:
```
df_result = X.groupby(['target','cluster'])
['Customer_Age'].count()
print(df_result)
```

In [ ]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca_transformed = pca.fit_transform(X)
```

In [ ]:
```
import matplotlib.pyplot as plt
from matplotlib.image import imread

img = imread('/Users/heejinkim/Desktop/hayul.png')
plt.imshow(img)
plt.show()
```

In [ ]:
```

```

In [ ]:

```

```