

AIVLE School 미니프로젝트

VOC를 제기한 고객의 **해지 여부** 예측 문제**[미션 안내]**

- VOC를 제기한 고객의 데이터를 읽어들이어 데이터를 분석 및 전처리한 후 머신러닝과 딥러닝으로 해지 여부를 예측하고 결과를 분석하세요.

[유의 사항]

- 각 문항의 답안코드는 반드시 '#여기에 답안코드를 작성하세요'로 표시된 cell에 작성해야 합니다.
- 제공된 cell을 추가/삭제하고 다른 cell에 답안코드를 작성 시 채점되지 않습니다.
- 반드시 문제에 제시된 가이드를 읽고 답안 작성하세요.
- 문제에 변수명이 제시된 경우 반드시 해당 변수명을 사용하세요.
- 문제와 데이터는 제3자에게 공유하거나 개인적인 용도로 사용하는 등 외부로 유출할 수 없으며 유출로 인한 책임은 응시자 본인에게 있습니다.

```
In [1]: # 코드실행시 경고 메시지 무시
import warnings
warnings.filterwarnings(action='ignore')
```

1. 필요한 라이브러리 설치

1-1. pip 이용해서 seaborn을 설치하세요.

```
In [2]: # 여기에 답안코드를 작성하세요.
!pip install seaborn
```

Requirement already satisfied: seaborn in c:\users\user\anaconda3\lib\site-packages (0.13.2)
 Requirement already satisfied: numpy!=1.24.0,>=1.20 in c:\users\user\anaconda3\lib\site-packages (from seaborn) (1.24.3)
 Requirement already satisfied: pandas>=1.2 in c:\users\user\anaconda3\lib\site-packages (from seaborn) (2.0.3)
 Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in c:\users\user\anaconda3\lib\site-packages (from seaborn) (3.8.3)
 Requirement already satisfied: contourpy>=1.0.1 in c:\users\user\anaconda3\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.0.5)
 Requirement already satisfied: cycler>=0.10 in c:\users\user\anaconda3\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (0.11.0)
 Requirement already satisfied: fonttools>=4.22.0 in c:\users\user\anaconda3\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (4.25.0)
 Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\user\anaconda3\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.4.4)
 Requirement already satisfied: packaging>=20.0 in c:\users\user\anaconda3\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (23.1)
 Requirement already satisfied: pillow>=8 in c:\users\user\anaconda3\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (10.0.1)
 Requirement already satisfied: pyparsing>=2.3.1 in c:\users\user\anaconda3\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (3.0.9)
 Requirement already satisfied: python-dateutil>=2.7 in c:\users\user\anaconda3\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (2.8.2)
 Requirement already satisfied: pytz>=2020.1 in c:\users\user\anaconda3\lib\site-packages (from pandas>=1.2->seaborn) (2023.3.post1)
 Requirement already satisfied: tzdata>=2022.1 in c:\users\user\anaconda3\lib\site-packages (from pandas>=1.2->seaborn) (2023.3)
 Requirement already satisfied: six>=1.5 in c:\users\user\anaconda3\lib\site-packages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn) (1.16.0)

1-2. numpy 별칭을 np로, pandas 별칭을 pd로 해서 임포트 하세요

```
In [3]: # 여기에 답안코드를 작성하세요.
import numpy as np
import pandas as pd
```

1-3. matplotlib 라이브러리를 plt로, seaborn을 sns로 해서 임포트 하세요

```
In [4]: # 여기에 답안코드를 작성하세요.
import seaborn as sns
import matplotlib.pyplot as plt
```

2. Tabular 데이터 로딩

2-1. pandas read_csv 함수를 사용하여 voc_data.csv 파일을 읽어온 후 df에 저장하세요.

```
In [5]: # 여기에 답안코드를 작성하세요.
df = pd.read_csv('voc_data.csv')
```

3. 데이터의 구성 확인

3-1. "df" DataFrame 이용해서 읽어들이는 파일의 앞부분 5줄, 뒷부분 5줄을 출력하세요

```
In [6]: # 여기에 답안코드를 작성하세요.
df.head()
```

```
Out[6]:
```

	voc_trt_perd_itg_cd	voc_prod_sbt_id	voc_wjt_sorc_id	voc_type_itg_cd	voc_sttus_itg_cd	voc_trt_reslt
0	-	1000665328	2153	10009	10002	
1	-	1001028714	3311	10009	10002	
2	-	1001028567	1575	10009	10002	
3	10000	1000665328	3546	10009	10002	
4	-	1000779276	3086	10009	10002	

5 rows × 24 columns

```
In [7]: # 여기에 답안코드를 작성하세요.
df.tail()
```

```
Out[7]:
```

	voc_trt_perd_itg_cd	voc_prod_sbt_id	voc_wjt_sorc_id	voc_type_itg_cd	voc_sttus_itg_cd	voc_trt_reslt
9995	-	1000811136	2123	10009	10002	
9996	-	1001047799	2153	10009	10002	
9997	-	1001027819	379	10009	10002	
9998	-	1001027819	314	10009	10002	
9999	-	1001047802	2266	10009	10002	

5 rows × 24 columns

3-2. 데이터프레임 정보(컬럼정보, Null 여부, 타입) 출력하세요

```
In [8]: # 여기에 답안코드를 작성하세요.
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 24 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   voc_trt_perd_itg_cd                  10000 non-null  object
1   voc_prod_sbt_id                     10000 non-null  int64
2   voc_wjt_sorc_id                     10000 non-null  int64
3   voc_type_itg_cd                     10000 non-null  int64
4   voc_sttus_itg_cd                    10000 non-null  int64
5   voc_trt_reslt_itg_cd                 10000 non-null  object
6   cust_clas_itg_cd                     10000 non-null  object
7   bprod_sbt_id                        10000 non-null  int64
8   age_itg_cd                           10000 non-null  object
9   cont_sttus_itg_cd                   10000 non-null  object
10  new_date                             10000 non-null  int64
11  opn_nfl_chg_date                     10000 non-null  int64
12  cust_dtl_ctg_itg_cd                 10000 non-null  object
13  voc_trt_degr_div_itg_cd             10000 non-null  int64
14  voc_dupl_tmscnt                     10000 non-null  int64
15  oos_cause_type_itg_cd               10000 non-null  object
16  voc_trt_need_time_itg_cd            10000 non-null  int64
17  engt_cperd_type_itg_cd              10000 non-null  object
18  engt_tgt_div_itg_cd                 10000 non-null  object
19  cont_fns_pam_date                   10000 non-null  int64
20  voc_mis_pbls_yn                     10000 non-null  object
21  fclt_oos_yn                         10000 non-null  object
22  cust_snsry_base_conf_need_time      10000 non-null  int64
23  trm_yn                              10000 non-null  object
dtypes: int64(12), object(12)
memory usage: 1.8+ MB
```

3-3. 데이터프레임 인덱스를 확인하세요

```
In [9]: # 여기에 답안코드를 작성하세요.
df.index
```

```
Out[9]: RangeIndex(start=0, stop=10000, step=1)
```

3-4. 데이터프레임 컬럼을 확인하세요

```
In [10]: # 여기에 답안코드를 작성하세요.
df.columns
```

```
Out[10]: Index(['voc_trt_perd_itg_cd', 'voc_prod_sbt_id', 'voc_wjt_sorc_id',
               'voc_type_itg_cd', 'voc_sttus_itg_cd', 'voc_trt_reslt_itg_cd',
               'cust_clas_itg_cd', 'bprod_sbt_id', 'age_itg_cd', 'cont_sttus_itg_cd',
               'new_date', 'opn_nfl_chg_date', 'cust_dtl_ctg_itg_cd',
               'voc_trt_degr_div_itg_cd', 'voc_dupl_tmscnt', 'oos_cause_type_itg_cd',
               'voc_trt_need_time_itg_cd', 'engt_cperd_type_itg_cd',
               'engt_tgt_div_itg_cd', 'cont_fns_pam_date', 'voc_mis_pbls_yn',
               'fclt_oos_yn', 'cust_snsry_base_conf_need_time', 'trm_yn'],
              dtype='object')
```

3-5. 데이터프레임 값(value)을 확인하세요

```
In [11]: # 여기에 답안코드를 작성하세요.
df.values
```

```
Out[11]: array([[ '_', 1000665328, 2153, ..., '_', 0, 'N'],
        [ '_', 1001028714, 3311, ..., '_', 0, 'N'],
        [ '_', 1001028567, 1575, ..., 'N', 0, 'N'],
        ...,
        [ '_', 1001027819, 379, ..., 'N', 0, 'N'],
        [ '_', 1001027819, 314, ..., '_', 0, 'N'],
        [ '_', 1001047802, 2266, ..., '_', 0, 'N']], dtype=object)
```

3-6. 데이터프레임의 계산 가능한 값들에 대한 통계치를 확인하세요

```
In [12]: # 여기에 답안코드를 작성하세요.
df.describe()
```

```
Out[12]:
```

	voc_prod_sbt_id	voc_wjt_sorc_id	voc_type_itg_cd	voc_sttus_itg_cd	bprod_sbt_id	new_date
count	1.000000e+04	10000.00000	10000.00000	10000.000000	1.000000e+04	1.000000e+04
mean	9.907267e+08	1578.29170	10008.52360	10002.043400	8.838173e+08	1.744283e+07
std	1.005780e+08	1078.63717	1.57927	0.314843	3.214229e+08	6.849207e+06
min	-9.980000e+02	126.00000	10003.00000	10002.000000	-9.980000e+02	1.010100e+04
25%	1.000782e+09	360.00000	10009.00000	10002.000000	1.000003e+09	2.008053e+07
50%	1.001028e+09	2056.00000	10009.00000	10002.000000	1.000749e+09	2.014121e+07
75%	1.001036e+09	2153.00000	10009.00000	10002.000000	1.001044e+09	2.018010e+07
max	1.001079e+09	3856.00000	10017.00000	10005.000000	1.001078e+09	2.020041e+07

3-7. DataFrame 컬럼 항목에 Null 존재하는지 확인하세요. (null값의 합계 포함)

```
In [13]: # 여기에 답안코드를 작성하세요.
df.isna().sum()
```

```

Out[13]: voc_trt_perd_itg_cd      0
          voc_prod_sbt_id      0
          voc_wjt_sorc_id      0
          voc_type_itg_cd      0
          voc_sttus_itg_cd      0
          voc_trt_reslt_itg_cd  0
          cust_clas_itg_cd      0
          bprod_sbt_id         0
          age_itg_cd           0
          cont_sttus_itg_cd     0
          new_date             0
          opn_nfl_chg_date      0
          cust_dtl_ctg_itg_cd   0
          voc_trt_degr_div_itg_cd 0
          voc_dupl_tmscnt       0
          oos_cause_type_itg_cd 0
          voc_trt_need_time_itg_cd 0
          engt_cperd_type_itg_cd 0
          engt_tgt_div_itg_cd   0
          cont_fns_pam_date     0
          voc_mis_pbls_yn       0
          fclt_oos_yn           0
          cust_snsry_base_conf_need_time 0
          trm_yn               0
          dtype: int64

```

3-8. voc_trt_perd_itg_cd 컬럼의 데이터를 확인하세요

```

In [14]: # 여기에 답안코드를 작성하세요.
         df['voc_trt_perd_itg_cd']

```

```

Out[14]: 0      -
          1      -
          2      -
          3    10000
          4      -
          ...
          9995   -
          9996   -
          9997   -
          9998   -
          9999   -
          Name: voc_trt_perd_itg_cd, Length: 10000, dtype: object

```

3-9. voc_trt_perd_itg_cd 컬럼 데이터별 건수를 나열하세요

```

In [15]: # 여기에 답안코드를 작성하세요.
         df['voc_trt_perd_itg_cd'].value_counts()

```

```
Out[15]: voc_trt_perd_itg_cd
_
10000    4283
10001     163
10002      58
10003      25
10004      16
10005      10
10006       6
10008       3
10009       3
10016       2
10011       2
10012       2
10007       2
10014       1
10013       1
10015       1
Name: count, dtype: int64
```

컬럼에서 '_' 값이 차지하는 비율

- voc_trt_perd_itg_cd : 0.54
- voc_trt_reslt_itg_cd : 0.88
- oos_cause_type_itg_cd : 0.9
- engt_cperd_type_itg_cd : 0.63
- engt_tgt_div_itg_cd : 0.63
- fclt_oos_yn : 0.90
- cust_clas_itg_cd : 0.2
- age_itg_cd : 0.22
- cont_sttus_itg_cd : 0.11
- cust_dtl_ctg_itg_cd : 0.11
- voc_mis_pbls_yn : 0.008

4. 데이터 결측치 처리

4-1. voc_trt_perd_itg_cd 컬럼에서 '_' 값이 차지하는 비율이 50%가 넘는 것을 확인하고, 이 voc_trt_perd_itg_cd 컬럼을 삭제하세요. (컬럼이 삭제된 데이터를 df1에 저장하세요)

```
In [16]: # 여기에 답안코드를 작성하세요.
voc_mean = df['voc_trt_perd_itg_cd'].str.count('_').sum() / len(df)
print(voc_mean)

df1 = df.drop('voc_trt_perd_itg_cd', axis=1)

0.5422
```

4-2. 'df1' DataFrame에서 '_' 값이 50% 이상되는 나머지 컬럼도 삭제하세요

```
In [17]: # 여기에 답안코드를 작성하세요.
drop_cols = ['voc_trt_reslt_itg_cd', 'oos_cause_type_itg_cd', 'engt_cperd_type_itg_cd', 'engt_
df1.drop(drop_cols, axis=1, inplace=True)
```

4-3. 'df1' DataFrame의 'cust_clas_itg_cd' 컬럼에 '_' 값이 몇 개 있는지 확인하여 출력하세요

```
In [18]: # 여기에 답안코드를 작성하세요.
df1['cust_clas_itg_cd'].str.count('_').sum()
```

```
Out[18]: 1934
```

4-4. df1의 남아있는 '_'값을 null로 변경: DataFrame replace 함수를 사용해서 모든 컬럼에 대해 '_'값을 null로 변경하고 df2에 저장하세요.

```
In [19]: # 여기에 답안코드를 작성하세요.
df2 = df1.replace({'_': None})
```

4-5. df2의 컬럼별 Null 갯수를 확인해보세요.

```
In [20]: # 여기에 답안코드를 작성하세요.
df.isna().sum()
```

```
Out[20]: voc_trt_perd_itg_cd      0
voc_prod_sbt_id      0
voc_wjt_sorc_id      0
voc_type_itg_cd      0
voc_sttus_itg_cd      0
voc_trt_reslt_itg_cd  0
cust_clas_itg_cd      0
bprod_sbt_id      0
age_itg_cd      0
cont_sttus_itg_cd      0
new_date      0
opn_nfl_chg_date      0
cust_dtl_ctg_itg_cd      0
voc_trt_degr_div_itg_cd  0
voc_dupl_tmscnt      0
oos_cause_type_itg_cd  0
voc_trt_need_time_itg_cd  0
engt_cperd_type_itg_cd  0
engt_tgt_div_itg_cd      0
cont_fns_pam_date      0
voc_mis_pbls_yn      0
fclt_oos_yn      0
cust_snsry_base_conf_need_time  0
trm_yn      0
dtype: int64
```

4-6. df2 데이터프레임 컬럼들의 데이터타입을 확인하세요.


```
In [21]: # 여기에 답안코드를 작성하세요.
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 24 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   voc_trt_perd_itg_cd                  10000 non-null  object
1   voc_prod_sbt_id                     10000 non-null  int64
2   voc_wjt_sorc_id                     10000 non-null  int64
3   voc_type_itg_cd                     10000 non-null  int64
4   voc_sttus_itg_cd                     10000 non-null  int64
5   voc_trt_reslt_itg_cd                 10000 non-null  object
6   cust_clas_itg_cd                     10000 non-null  object
7   bprod_sbt_id                        10000 non-null  int64
8   age_itg_cd                           10000 non-null  object
9   cont_sttus_itg_cd                   10000 non-null  object
10  new_date                             10000 non-null  int64
11  opn_nfl_chg_date                     10000 non-null  int64
12  cust_dtl_ctg_itg_cd                 10000 non-null  object
13  voc_trt_degr_div_itg_cd              10000 non-null  int64
14  voc_dupl_tmcsnt                     10000 non-null  int64
15  oos_cause_type_itg_cd                10000 non-null  object
16  voc_trt_need_time_itg_cd             10000 non-null  int64
17  engt_cperd_type_itg_cd               10000 non-null  object
18  engt_tgt_div_itg_cd                 10000 non-null  object
19  cont_fns_pam_date                   10000 non-null  int64
20  voc_mis_pbls_yn                     10000 non-null  object
21  fclt_oos_yn                         10000 non-null  object
22  cust_snsry_base_conf_need_time       10000 non-null  int64
23  trm_yn                              10000 non-null  object
dtypes: int64(12), object(12)
memory usage: 1.8+ MB
```

4-7. df2 데이터프레임에 대해 먼저, 'cust_clas_itg_cd' 컬럼의 최빈값을 확인하는 코드로 확인하고 다음으로, 이 컬럼의 Null 값을 최빈값으로 변경하세요(fillna 함수 사용). 처리된 데이터프레임은 df3에 저장하세요

```
In [22]: # 여기에 답안코드를 작성하세요.
df2['cust_clas_itg_cd'] = df2['cust_clas_itg_cd'].fillna(df2['cust_clas_itg_cd'].mode()[0])
df3 = df2.copy()
```

4-8. df3에 대해 'age_itg_cd'의 null 값을 중앙값(median)으로 변경하고 데이터 타입을 정수(int)로 변경하세요. 데이터 처리 후 데이터프레임을 df4에 저장하세요.

```
In [23]: # 여기에 답안코드를 작성하세요.
df3['age_itg_cd'] = df3['age_itg_cd'].fillna(df3['age_itg_cd'].median())
df4 = df3.copy()
```

4-9. df4에 대해 'cont_sttus_itg_cd'의 null 값을 최빈값(mode)으로 변경하세요. 데이터 처리 후 데이터프레임을 df5에 저장하세요.

```
In [24]: # 여기에 답안코드를 작성하세요.
df4['cont_sttus_itg_cd'] = df4['cont_sttus_itg_cd'].fillna(df4['cont_sttus_itg_cd'].mode()[0])
df5 = df4.copy()
```

4-10. df5에 대해 'cust_dtl_ctg_itg_cd'의 null 값을 최빈값(mode)으로 변경하세요

```
In [25]: # 여기에 답안코드를 작성하세요.
df5['cust_dtl_ctg_itg_cd'] = df5['cust_dtl_ctg_itg_cd'].fillna(df5['cust_dtl_ctg_itg_cd'].mode
```

4-11. df5에 대해 다음 날짜 관련 컬럼을 확인 후 삭제하세요. (날짜 관련 컬럼: new_date, opn_nfl_chg_date, cont_fns_pam_date)

```
In [26]: # 여기에 답안코드를 작성하세요.
df5.info()
drop_cols = ['new_date', 'opn_nfl_chg_date', 'cont_fns_pam_date']
df5.drop(drop_cols, axis=1, inplace=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 18 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   voc_prod_sbt_id                       10000 non-null  int64
1   voc_wjt_sorc_id                       10000 non-null  int64
2   voc_type_itg_cd                       10000 non-null  int64
3   voc_sttus_itg_cd                      10000 non-null  int64
4   cust_clas_itg_cd                      10000 non-null  object
5   bprod_sbt_id                         10000 non-null  int64
6   age_itg_cd                           10000 non-null  object
7   cont_sttus_itg_cd                     10000 non-null  object
8   new_date                             10000 non-null  int64
9   opn_nfl_chg_date                      10000 non-null  int64
10  cust_dtl_ctg_itg_cd                   10000 non-null  object
11  voc_trt_degr_div_itg_cd               10000 non-null  int64
12  voc_dupl_tmcsnt                       10000 non-null  int64
13  voc_trt_need_time_itg_cd              10000 non-null  int64
14  cont_fns_pam_date                     10000 non-null  int64
15  voc_mis_pbls_yn                       9914 non-null   object
16  cust_snsry_base_conf_need_time        10000 non-null  int64
17  trm_yn                               10000 non-null  object
dtypes: int64(12), object(6)
memory usage: 1.4+ MB
```

4-12. df5에 대해 'voc_mis_pbls_yn' 컬럼을 삭제하세요.

```
In [27]: # 여기에 답안코드를 작성하세요.
df5.drop('voc_mis_pbls_yn', axis=1, inplace=True)
```

5. 라벨 인코딩, 원핫 인코딩

5-1. df5에 대해 object 타입 컬럼을 cat_cols에 저장하세요. 그 중 cat_cols의 cust_clas_itg_cd 컬럼에 대해 LabelEncoder를 적용해보세요. (적용 후 df5에 저장)

```
In [28]: # 여기에 답안코드를 작성하세요.
from sklearn.preprocessing import LabelEncoder

dump_cols = ['cust_clas_itg_cd', 'age_itg_cd', 'cont_status_itg_cd', 'cust_dtl_ctg_itg_cd', 'trm_yn_Y']
cat_cols = df5.select_dtypes(include='object')

label_cols = LabelEncoder()
df5['cust_clas_itg_cd'] = label_cols.fit_transform(cat_cols['cust_clas_itg_cd'])
```

5-2. df5의 나머지 object 컬럼에 대해서 One-Hot-Encoding될수 있도록 Pandas의 get_dummies 함수를 적용하세요. (적용 후 df6에 저장)

```
In [29]: # 여기에 답안코드를 작성하세요.
df6 = pd.get_dummies(df5, columns=dump_cols, drop_first=True, dtype=int)
df6.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Columns: 128 entries, voc_prod_sbt_id to trm_yn_Y
dtypes: int32(119), int64(9)
memory usage: 5.2 MB
```

```
In [30]: df6.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Columns: 128 entries, voc_prod_sbt_id to trm_yn_Y
dtypes: int32(119), int64(9)
memory usage: 5.2 MB
```

6. x,y 데이터 분리

6-1. df6에 대해 X, y 값을 가지고 8:2 비율로 Train, Test Dataset으로 나누세요. (y 클래스 비율에 맞게 분리, y 값은 'trm_yn_Y' 컬럼, random_state는 42)

```
In [31]: # 여기에 답안코드를 작성하세요.
from sklearn.model_selection import train_test_split

target = 'trm_yn_Y'
x = df6.drop(target, axis=1)
y = df6.loc[:, target]

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.2, random_state=42, strat
```

7. 데이터 정규분포화, 표준화

7-1. 사이킷런의 StandardScaler로 훈련데이터셋은 정규분포화(fit_transform)하고 테스트 데이터셋은 표준화(transform)하세요.

```
In [32]: # 여기에 답안코드를 작성하세요.
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

8. 머신러닝 모델링 & 모델 성능평가 및 그래프 출력

로지스틱 회귀 (LogisticRegression, 분류)

8-1. LogisticRegression 모델을 만들고 학습을 진행하세요 (단, 규제강도C는 10으로 설정, 계산에 사용할 작업수 max_iter는 2000으로 설정하세요)

```
In [33]: # 여기에 답안코드를 작성하세요.
from sklearn.linear_model import LogisticRegression

model_lr = LogisticRegression(C=10, max_iter=2000)
model_lr.fit(x_train, y_train)
model_lr.score(x_test, y_test)
```

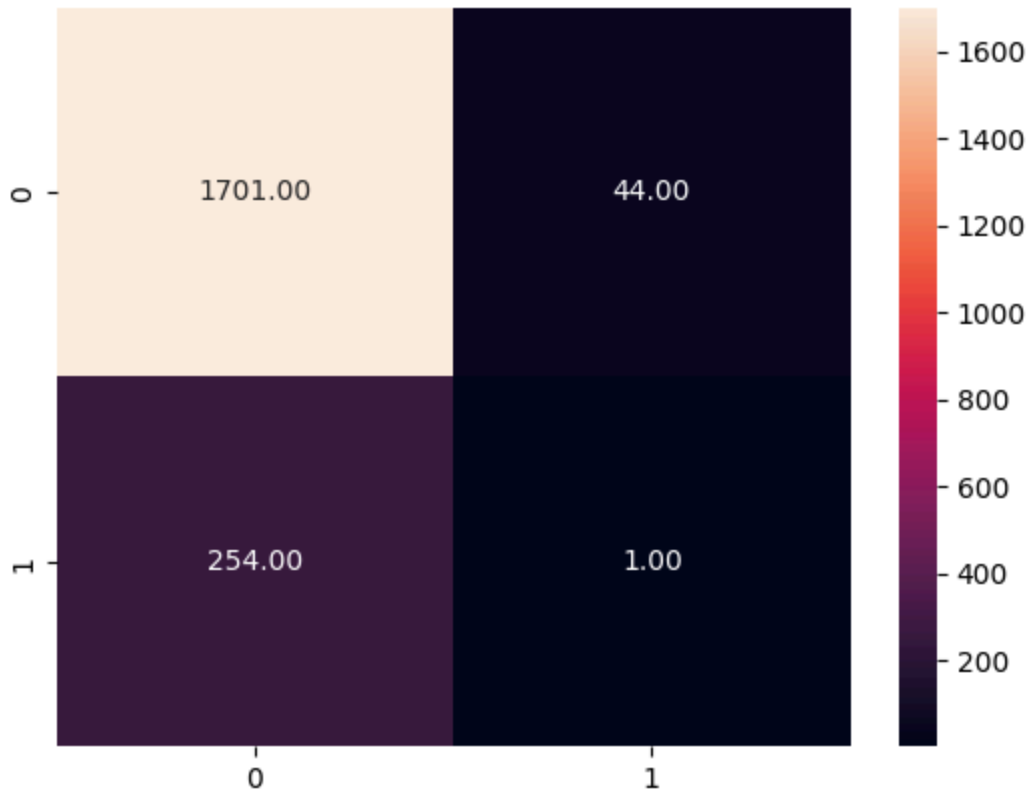
```
Out[33]: 0.851
```

8-2. 위 모델의 성능을 평가하려고 합니다. y값을 예측하여 confusion matrix를 구하고 heatmap 그래프로 시각화하세요. 그리고 Scikit-learn의 classification_report를 활용하여 성능을 출력하세요.

```
In [34]: # 여기에 답안코드를 작성하세요.
from sklearn.metrics import *
import matplotlib.pyplot as plt
import seaborn as sns

pred = model_lr.predict(x_test)
print(confusion_matrix(pred, y_test))
sns.heatmap(confusion_matrix(pred, y_test), annot=True, fmt='.2f')
plt.show()
print(classification_report(pred, y_test))
```

```
[[1701  44]
 [ 254   1]]
```



	precision	recall	f1-score	support
0	0.87	0.97	0.92	1745
1	0.02	0.00	0.01	255
accuracy			0.85	2000
macro avg	0.45	0.49	0.46	2000
weighted avg	0.76	0.85	0.80	2000

8-3. DecisionTree 모델을 만들고 학습을 진행하세요. (단, max_depth는 10, random_state는 42로 설정)

```
In [35]: # 여기에 답안코드를 작성하세요.
from sklearn.tree import DecisionTreeClassifier

model_lr = DecisionTreeClassifier(max_depth=10, random_state=42)
model_lr.fit(x_train, y_train)
model_lr.score(x_test, y_test)
```

Out[35]: 0.9765

8-4. RandomForest 모델을 만들고 학습을 진행하세요. (단, n_estimators=100, random_state=42 설정)

```
In [36]: # 여기에 답안코드를 작성하세요.
from sklearn.ensemble import RandomForestClassifier

model_rf = RandomForestClassifier(n_estimators=100, random_state=42)
```

```
model_rf.fit(x_train, y_train)
model_lr.score(x_test, y_test)
```

Out[36]: 0.9765

8-5. XGBoost 모델을 만들고 학습을 진행하세요. (단, n_estimators=5 설정)

```
In [37]: # 여기에 답안코드를 작성하세요.
from xgboost import XGBClassifier

model_xgb = XGBClassifier(n_estimators=5)
model_xgb.fit(x_train, y_train)
model_lr.score(x_test, y_test)
```

Out[37]: 0.9765

8-6. Light GBM 모델을 만들고 학습을 진행하세요. (단, n_estimators=3 설정)

```
In [38]: # 여기에 답안코드를 작성하세요.
from lightgbm import LGBMClassifier

model_lgbm = LGBMClassifier(n_estimators=3)
model_lgbm.fit(x_train, y_train)
model_lgbm.score(x_test, y_test)
```

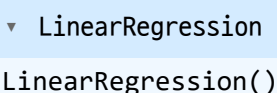
```
[LightGBM] [Info] Number of positive: 179, number of negative: 7821
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.001332
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 813
[LightGBM] [Info] Number of data points in the train set: 8000, number of used features: 90
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.022375 -> initscore=-3.777182
[LightGBM] [Info] Start training from score -3.777182
```

Out[38]: 0.9775

8-7. Linear Regression 모델을 연습으로 만들고 학습을 진행하세요.

```
In [39]: # 이 데이터로 연습하세요.
x_data = np.array([1.6, 2.3, 3.5, 4.6]).reshape(-1,1)
y_data = np.array([3.3, 5.5, 7.2, 9.9])
```

```
In [40]: # 여기에 답안코드를 작성하세요.
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(x_data, y_data)
```

Out[40]:  LinearRegression()

9. 딥러닝 모델링 & 모델 성능평가 및 그래프 출력

9-1. 아래 가이드대로 해지여부를 분류하는 딥러닝 모델을 만드세요.

- 첫번째 Hidden Layer : unit 64 , activation='relu'
- 두번째 Hidden Layer : unit 32 , activation='relu'
- 세번째 Hidden Layer : unit 16 , activation='relu'
- 각 Hidden Layer 마다 Dropout 0.2 비율로 되도록 하세요.
- EarlyStopping 콜백을 적용하고 ModelCheckpoint 콜백으로 validation performance가 좋은 모델을 h5 모델로 저장하세요.
- batch_size는 10, epochs는 10으로 설정하세요.

```
In [41]: # 여기에 답안코드를 작성하세요.
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

n = x_train.shape[1]
model = Sequential([Dense(64, input_shape=(n, ), activation='relu'),
                    Dense(32, activation='relu'),
                    Dense(16, activation='relu'),
                    Dropout(0.2),
                    Dense(1, activation='sigmoid')
                    ])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
es = EarlyStopping(monitor='val_loss', patience=5, mode='min', verbose=1)
mcp = ModelCheckpoint('best_model.keras', monitor='val_loss', verbose=1, save_best_only=True)

history = model.fit(x_train, y_train, epochs=10, batch_size=10, callbacks=[es, mcp], validation
```

```

Epoch 1/10
780/800 ————— 0s 2ms/step - accuracy: 0.9556 - loss: 818572.9375
Epoch 1: val_loss improved from inf to 0.44711, saving model to best_model.keras
800/800 ————— 4s 2ms/step - accuracy: 0.9558 - loss: 802399.1875 - val_accuracy: 0.9745 - val_loss: 0.4471
Epoch 2/10
800/800 ————— 0s 2ms/step - accuracy: 0.9666 - loss: 895.2879
Epoch 2: val_loss improved from 0.44711 to 0.24837, saving model to best_model.keras
800/800 ————— 2s 2ms/step - accuracy: 0.9666 - loss: 895.0818 - val_accuracy: 0.9775 - val_loss: 0.2484
Epoch 3/10
763/800 ————— 0s 2ms/step - accuracy: 0.9807 - loss: 0.2537
Epoch 3: val_loss improved from 0.24837 to 0.17068, saving model to best_model.keras
800/800 ————— 2s 2ms/step - accuracy: 0.9805 - loss: 0.2548 - val_accuracy: 0.9775 - val_loss: 0.1707
Epoch 4/10
779/800 ————— 0s 2ms/step - accuracy: 0.9783 - loss: 0.2418
Epoch 4: val_loss improved from 0.17068 to 0.13738, saving model to best_model.keras
800/800 ————— 2s 2ms/step - accuracy: 0.9783 - loss: 0.2411 - val_accuracy: 0.9775 - val_loss: 0.1374
Epoch 5/10
782/800 ————— 0s 2ms/step - accuracy: 0.9734 - loss: 0.1621
Epoch 5: val_loss improved from 0.13738 to 0.12118, saving model to best_model.keras
800/800 ————— 2s 2ms/step - accuracy: 0.9735 - loss: 0.1620 - val_accuracy: 0.9775 - val_loss: 0.1212
Epoch 6/10
781/800 ————— 0s 2ms/step - accuracy: 0.9763 - loss: 0.1235
Epoch 6: val_loss improved from 0.12118 to 0.11318, saving model to best_model.keras
800/800 ————— 2s 2ms/step - accuracy: 0.9764 - loss: 0.1234 - val_accuracy: 0.9775 - val_loss: 0.1132
Epoch 7/10
780/800 ————— 0s 2ms/step - accuracy: 0.9762 - loss: 0.1158
Epoch 7: val_loss improved from 0.11318 to 0.10949, saving model to best_model.keras
800/800 ————— 2s 2ms/step - accuracy: 0.9762 - loss: 0.1157 - val_accuracy: 0.9775 - val_loss: 0.1095
Epoch 8/10
797/800 ————— 0s 2ms/step - accuracy: 0.9792 - loss: 0.1025
Epoch 8: val_loss improved from 0.10949 to 0.10789, saving model to best_model.keras
800/800 ————— 2s 2ms/step - accuracy: 0.9792 - loss: 0.1026 - val_accuracy: 0.9775 - val_loss: 0.1079
Epoch 9/10
789/800 ————— 0s 2ms/step - accuracy: 0.9744 - loss: 0.1166
Epoch 9: val_loss improved from 0.10789 to 0.10726, saving model to best_model.keras
800/800 ————— 2s 2ms/step - accuracy: 0.9745 - loss: 0.1165 - val_accuracy: 0.9775 - val_loss: 0.1073
Epoch 10/10
798/800 ————— 0s 2ms/step - accuracy: 0.9773 - loss: 0.1062
Epoch 10: val_loss improved from 0.10726 to 0.10706, saving model to best_model.keras
800/800 ————— 3s 2ms/step - accuracy: 0.9773 - loss: 0.1062 - val_accuracy: 0.9775 - val_loss: 0.1071

```

9-2. `y_train`, `y_test`를 원핫 인코딩 후 다중 분류하는 딥러닝 모델을 만드세요. 9-1과 동일한 가이드 적용

```

In [42]: # 여기에 답안코드를 작성하세요.
from keras.utils import to_categorical

y_train_oh = to_categorical(y_train)

```



```

y_test_oh = to_categorical(y_test)

model = Sequential([Dense(64, input_shape=(n, ), activation='relu'),
                    Dense(32, activation='relu'),
                    Dense(16, activation='relu'),
                    Dropout(0.2),
                    Dense(2, activation='softmax')
                    ])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
history = model.fit(x_train, y_train_oh, batch_size=10, epochs=10, callbacks=[es,mcp], validate_

Epoch 1/10
786/800 ————— 0s 2ms/step - acc: 0.9389 - loss: 477854.9688
Epoch 1: val_loss did not improve from 0.10706
800/800 ————— 4s 2ms/step - acc: 0.9391 - loss: 471408.1875 - val_acc: 0.9745 - val_loss: 0.4691
Epoch 2/10
792/800 ————— 0s 2ms/step - acc: 0.9557 - loss: 1623.9908
Epoch 2: val_loss did not improve from 0.10706
800/800 ————— 2s 2ms/step - acc: 0.9558 - loss: 1613.0741 - val_acc: 0.9775 - val_loss: 0.2480
Epoch 3/10
787/800 ————— 0s 2ms/step - acc: 0.9809 - loss: 5.5546
Epoch 3: val_loss did not improve from 0.10706
800/800 ————— 2s 2ms/step - acc: 0.9808 - loss: 5.6532 - val_acc: 0.9775 - val_loss: 0.1318
Epoch 4/10
792/800 ————— 0s 2ms/step - acc: 0.9777 - loss: 15.8619
Epoch 4: val_loss did not improve from 0.10706
800/800 ————— 2s 2ms/step - acc: 0.9777 - loss: 15.8192 - val_acc: 0.9775 - val_loss: 0.1125
Epoch 5/10
799/800 ————— 0s 2ms/step - acc: 0.9776 - loss: 4.5134
Epoch 5: val_loss did not improve from 0.10706
800/800 ————— 2s 2ms/step - acc: 0.9775 - loss: 4.5099 - val_acc: 0.9775 - val_loss: 0.1077
Epoch 5: early stopping

```

```

In [43]: # 참고
# Y 레이블 One-Hot-Encoding 되지 않았으면 loss='sparse_categorical_crossentropy' 사용
# model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['acc'])
# history = model.fit(X_train, y_train, batch_size=batch_size, epochs=epochs, callbacks=[es,mc

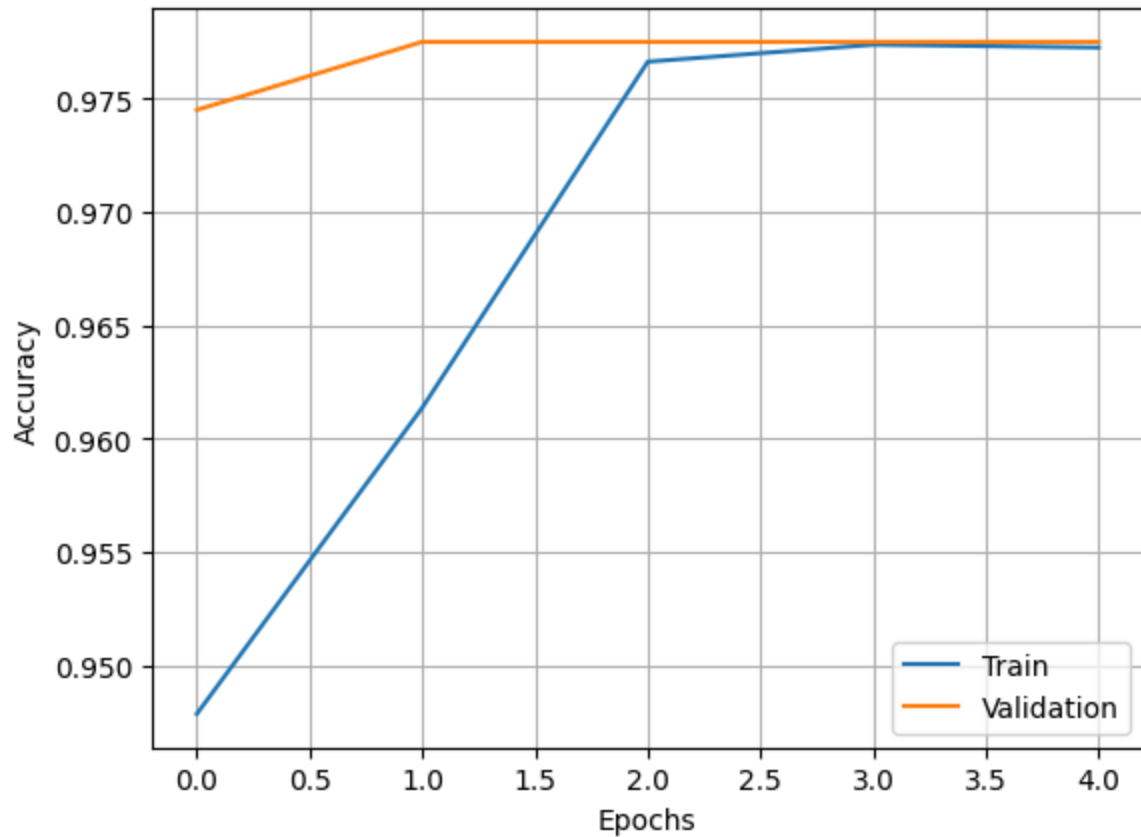
```

9-3. 모델 성능을 평가해서 그래프로 표현하세요. 학습 정확도와 검증정확도를 그래프로 표시하고 xlabel에는 Epochs, ylabel에는 Accuracy, 범례에는 Train과 Validation으로 표시하세요..

```

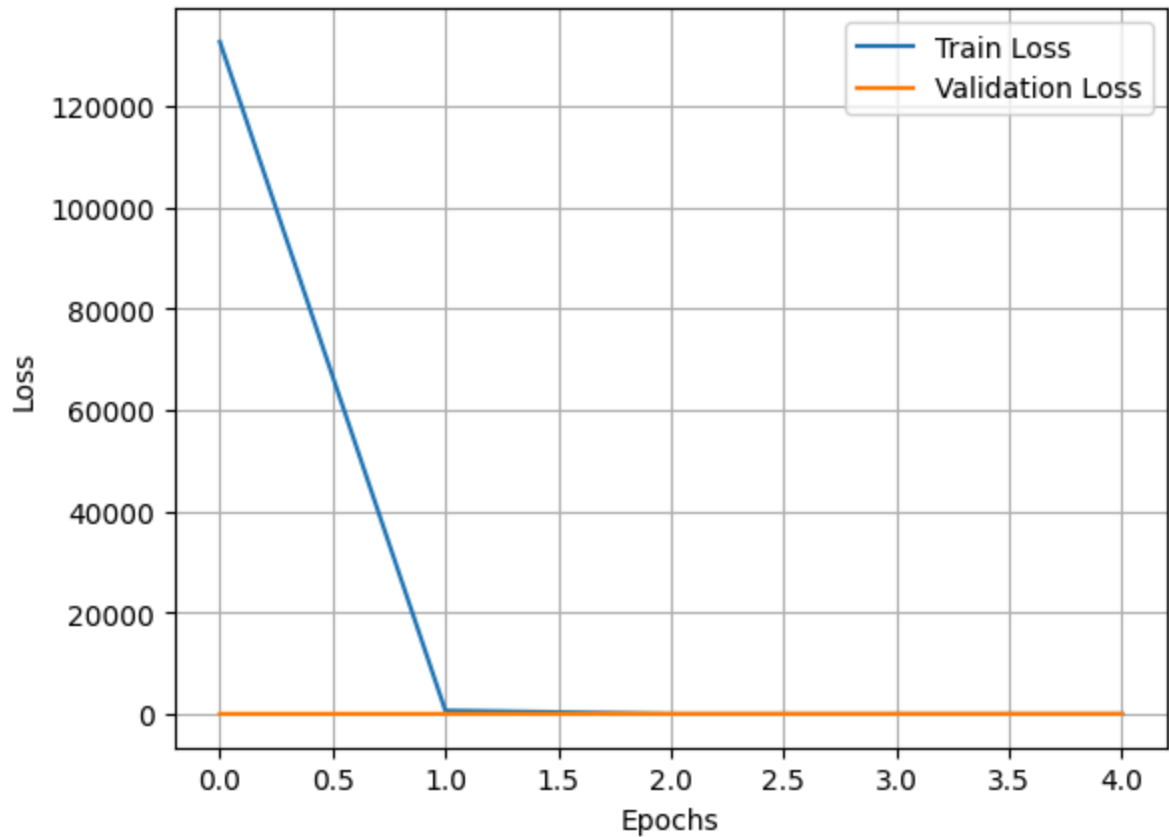
In [44]: # 여기에 답안코드를 작성하세요.
plt.plot(history.history['acc'], label='Train')
plt.plot(history.history['val_acc'], label='Validation')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid()
plt.show()

```



9-4. 모델 성능을 평가해서 그래프로 표현하세요. 학습 손실과 검증 손실을 그래프로 표시하고 xlabel에는 Epochs, ylabel에는 Loss, 범례에는 Train Loss와 Validation Loss로 표시하세요.

```
In [45]: # 여기에 답안코드를 작성하세요.
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid()
plt.show()
```



9-5. y값을 예측하여 y_test_pred에 저장하고 정확도를 출력하세요..

```
In [46]: # 여기에 답안코드를 작성하세요.
y_test_pred = model.predict(x_test)
y_test_pred = np.argmax(y_test_pred, axis=1)
y_test = np.argmax(y_test_oh, axis=1)

print(accuracy_score(y_test, y_test_pred))

63/63 ————— 0s 2ms/step
0.9775
```

In []: