AIVLE School 미니프로젝트

통신 서비스 이용 고객의 이탈 여부 예측 문제

[미션 안내]

- 고객 관련 데이터를 분석한 후 고객의 서비스 중단 또는 경쟁업체로의 이탈 여부를 예측하는 머신러닝, 딥러닝 모델을 만들고 결과를 예측하세요.

[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.

1. scikit-learn 패키지는 머신러닝 교육을 위한 최고의 파이썬 패키지입니다.

scikit-learn를 별칭(alias) sk로 임포트하는 코드를 작성하고 실행하세요.

In [1]:
```
# 여기에 답안코드를 작성하세요.
!pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in c:\users\user\anaconda3\lib\site-packages (1.3.
0)
Requirement already satisfied: numpy>=1.17.3 in c:\users\user\anaconda3\lib\site-packages (fro
m scikit-learn) (1.24.3)
Requirement already satisfied: scipy>=1.5.0 in c:\users\user\anaconda3\lib\site-packages (from
scikit-learn) (1.11.1)
Requirement already satisfied: joblib>=1.1.1 in c:\users\user\anaconda3\lib\site-packages (fro
m scikit-learn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\user\anaconda3\lib\site-packag
es (from scikit-learn) (2.2.0)
```

In [2]:
```
import sklearn as sk
```

2. Pandas를 사용할 수 있도록 별칭(alias)을 pd로 해서 불러오세요.

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

3. 모델링을 위해 분석 및 처리할 데이터 파일을 읽어오려고 합니다.

Pandas함수로 데이터 파일을 읽어 데이터프레임 변수명 df에 할당하는 코드를 작성하세요.

- churn_data.csv 파일을 읽어 데이터 프레임 변수명 df에 할당하세요.

In [4]:
```
# 여기에 답안코드를 작성하세요.
df = pd.read_csv('churn_data.csv')
```

4. df에서 불필요한 customerID 컬럼을 삭제하고 df1에 저장하세요.

In [5]:
```
# 여기에 답안코드를 작성하세요.
df1 = df.drop('customerID', axis=1)
```

5. df1의 TotalCharges 컬럼의 타입을 float로 변경하세요.

- TotalCharge의 컬럼 타입을 확인하는 코드를 작성하세요.
- ' ' 값을 0으로 변환하고 컬럼 타입을 float로 변경하세요.
- 전처리 후 데이터를 df2에 저장하세요.

In [6]:
```
# 여기에 답안코드를 작성하세요.
df1['TotalCharges'].info()
df1['TotalCharges'] = df1['TotalCharges'].replace(' ', 0)
df1['TotalCharges'] = df1['TotalCharges'].astype(float)
df2 = df1.copy()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 7043 entries, 0 to 7042
Series name: TotalCharges
Non-Null Count  Dtype
--------------  -----
7043 non-null   object
dtypes: object(1)
memory usage: 55.2+ KB
```

6. df2에서 churn 컬럼의 데이터별 개수를 확인하는 코드를 작성하고

Yes, No를 각각 1, 0으로 변환한 후 df3에 저장하세요.

In [7]:
```
# 여기에 답안코드를 작성하세요.
print(df2['Churn'].value_counts())
df2['Churn'] = df2['Churn'].map({'Yes':1, 'No':0})
df3 = df2.copy()
```

```
Churn
No     5174
Yes    1869
Name: count, dtype: int64
```

7. df3의 모든 컬럼에 대해 결측치를 확인하는 코드를 작성하고 결측치를 처리하세요.

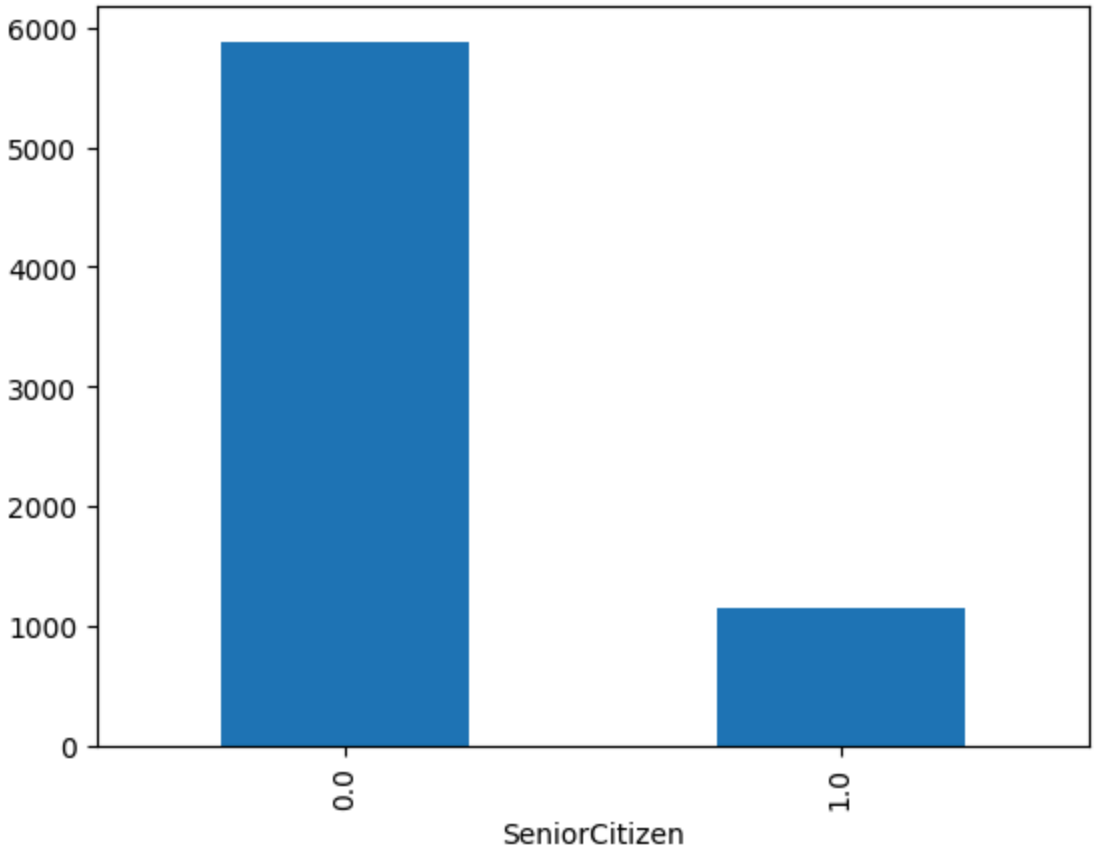
- 결측치가 40% 이상인 컬럼은 컬럼을 삭제하세요.
- 결측치가 40% 미만인 컬럼은 결측치가 있는 row를 삭제하세요.
- 전처리한 데이터를 df4에 저장하세요.

In [8]:
```
# 여기에 답안코드를 작성하세요.
fill40 = df3.isna().mean()*100
index_o40 = fill40[fill40>=40].index
index_u40 = fill40[fill40<=40].index
df3.drop(columns=index_o40, inplace=True)
df3.dropna(subset=index_u40, inplace=True)
df4 = df3.copy()
```

8. df4에서 SeniorCitizen 컬럼을 bar 차트로 확인해보고 불균형을 확인해보세요.

SeniorCitizen 컬럼은 불균형이 심하므로 삭제하세요.

In [9]:
```
# 여기에 답안코드를 작성하세요.
import matplotlib.pyplot as plt

df4['SeniorCitizen'].value_counts().plot(kind='bar')
plt.show()

df4.drop('SeniorCitizen', axis=1, inplace=True)
```

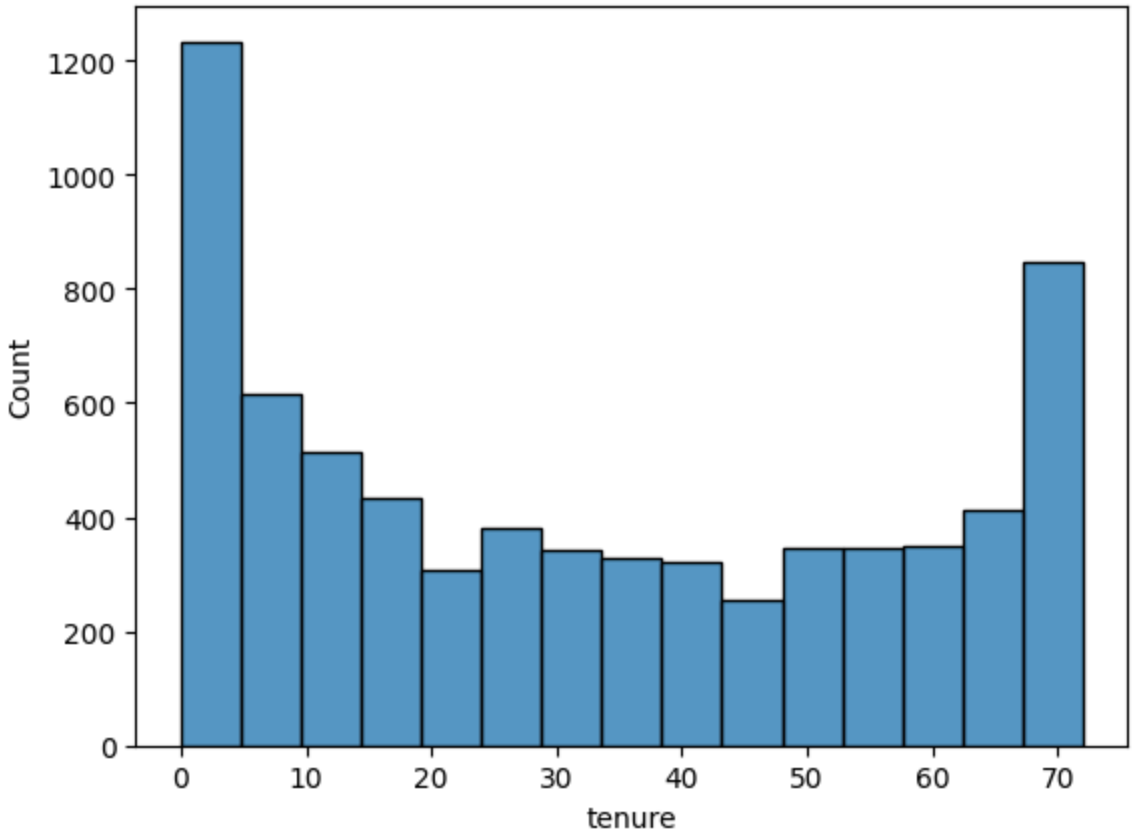


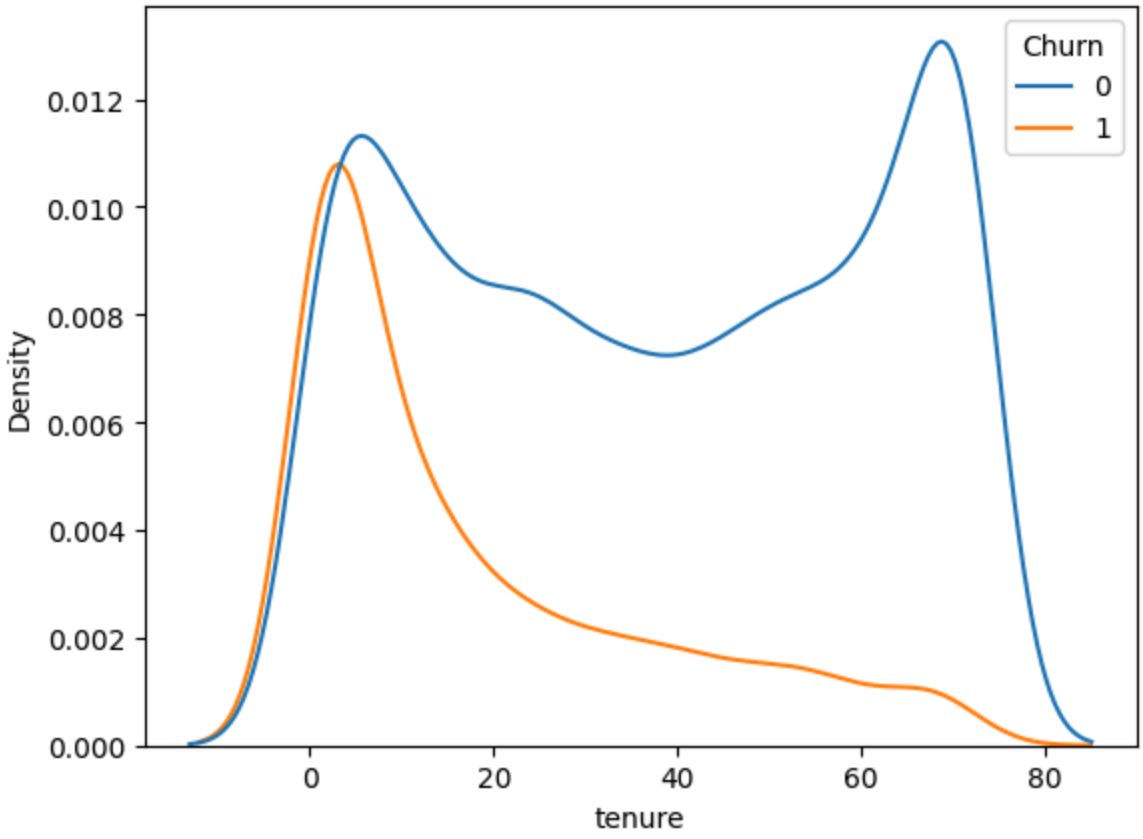
9. df4에서 다음의 가이드에 따라 데이터를 시각화 해보세요.

- tenure (서비스 사용기간)에 대해 히스토그램으로 시각화 하세요.
- tenure를 x 값으로 churn을 hue 값으로 사용하여 kdeplot으로 시각화 하고 '서비스 사용기간이 길어질 수록 이탈이 적다'에 대해 'O'인지 'X'인지 출력하세요.
- MultipleLines에 대해 countplot을 그리고 churn을 hue 값으로 사용하여 countplot으로 시각화 하고 'MultipleLines 서비스를 사용하는 고객이 약간 더 높은 이탈율을 보인다'에 대해 'O'인지 'X'인지 출력하세요.
- 'tenure','MonthlyCharges','TotalCharges' 컬럼간의 상관관계를 확인하여 heatmap으로 시각화하고 가장 높은 상관계수 값을 출력하세요.

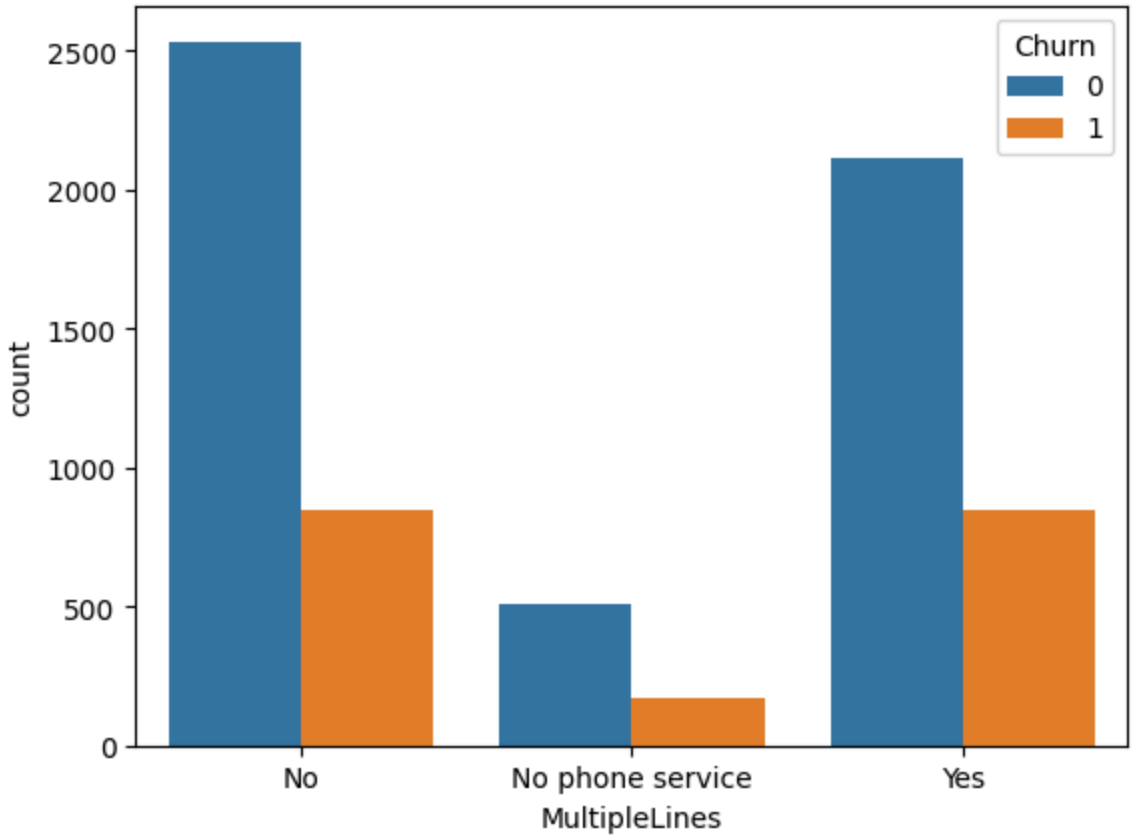
In [10]:
```
# 여기에 답안코드를 작성하세요.
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

sns.histplot(df4['tenure'])
plt.show()

sns.kdeplot(x='tenure', hue='Churn', data=df4)
#plt.legend(labels=['X', 'O'])
plt.show()
print('O')

sns.countplot(x='MultipleLines', hue='Churn', data=df4)
```

```
#plt.legend(labels=['O', 'X'])
plt.show()
print('O')

corr_cols = ['tenure', 'MonthlyCharges', 'TotalCharges']
corr_matrix = df4[corr_cols].corr()
sns.heatmap(corr_matrix, annot=True)

max_tri = np.triu(corr_matrix, k=1)
max_val = np.max(max_tri)
print(round(max_val, 2))
plt.show()
```



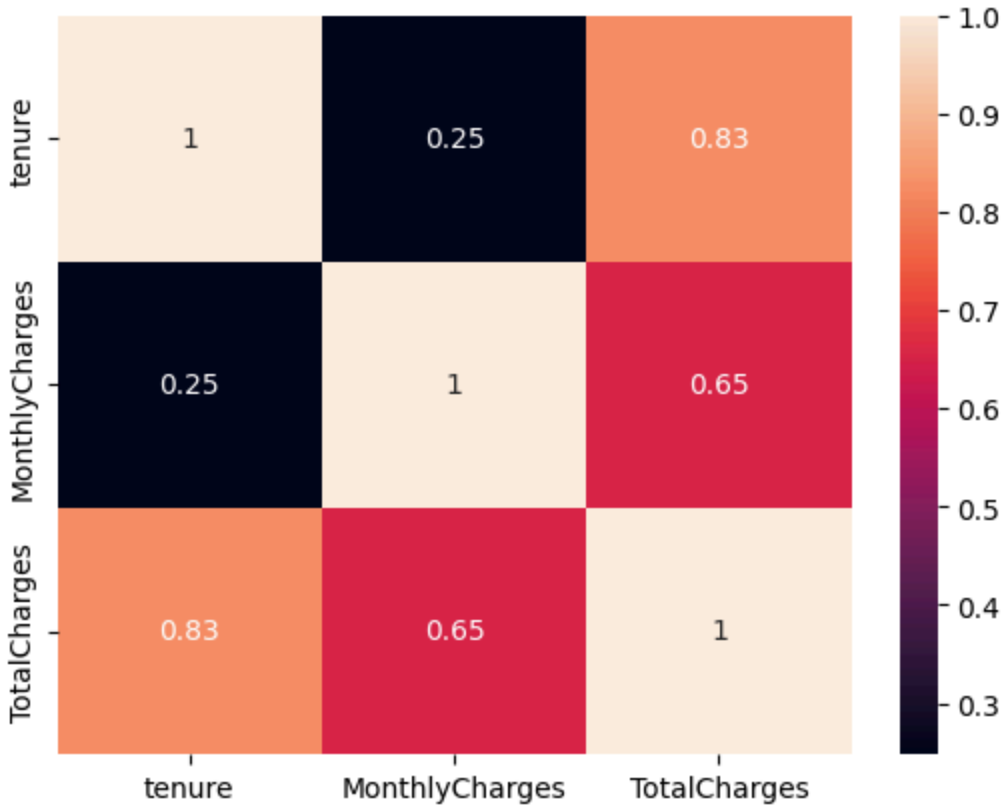

Churn
0
1
Density
0.012
0.010
0.008
0.006
0.004
0.002
0.000
0
20
40
60
80
tenure

0


Churn
0
1
count
2500
2000
1500
1000
500
0
No
No phone service
Yes
MultipleLines

0

0.83



10. df4에서 컬럼의 데이터 타입이 object인 컬럼들을 원-핫 인코딩 하세요.

- 컬럼의 데이터 타입이 object인 컬럼들을 object_cols 변수에 저장하세요.
- object_cols 변수의 컬럼들을 원-핫 인코딩하세요.
- 전처리된 데이터를 df5에 저장하세요.

In [11]:
```
# 여기에 답안코드를 작성하세요.
# object_cols  = ['gender', 'Partner', 'Dependents', 'PhoneService', 'MultipleLines', 'Interne
#                 'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilli

object_cols = df4.select_dtypes('object').columns.values
df5 = pd.get_dummies(df4, columns=object_cols, drop_first=True, dtype=int)
```

11. df5에 대해 Scikit-learn의 train_test_split 함수로 훈련, 검증 데이터를 분리하세요.

- 입력 : X, y (y에는 churn을 저장하고 X에는 churn을 제외한 나머지를 저장하세요)
- Train : Test 비율 = 8:2
- y Class 비율에 맞게 나누는 옵션을 추가하세요.
- random_state=42 로 설정하세요.
- 결과 : X_train, X_valid, y_train, y_valid에 저장하세요.

In [12]:
```
# 여기에 답안코드를 작성하세요.
from sklearn.model_selection import train_test_split

target = 'Churn'
X = df5.drop(target, axis=1)
y = df5.loc[:, target]

X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=.2, random_state=42)
```

12. MinMaxScaler 함수를 'scaler'로 정의하고 데이터를 정규화하세요.

In [13]:
```
# 여기에 답안코드를 작성하세요.
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X_train_s = scaler.fit_transform(X_train)
X_valid_s = scaler.transform(X_valid)
```

13. 고객 이탈 여부를 예측하는 머신러닝 모델을 만들려고 합니다.

아래 가이드에 따라 모델링하고 학습을 진행하세요.

- LogisticRegression 모델 정의하고 학습시키세요.
- KNN으로 모델을 정의하고 학습시키세요. (n_neighbors=5)
- Decision Tree로 모델을 정의하고 학습시키세요. (max_depth=10, random_state=42)
- RandomForest로 모델을 정의하고 학습시키세요. (n_estimators=3, random_state=42)
- XGBoost로 모델을 정의하고 학습시키세요. (n_estimators=3, random_state=42)
- Light GBM으로 모델을 정의하고 학습시키세요. (n_estimators=3, random_state=42)
- 각각 다른 셀에 답안코드를 작성하세요.

In [14]:
```
# 여기에 답안코드를 작성하세요.(LogisticRegression)
from sklearn.linear_model import LogisticRegression

model_lr = LogisticRegression()
model_lr.fit(X_train, y_train)
pred_lr = model_lr.predict(X_valid)
```

In [15]:
```
# 여기에 답안코드를 작성하세요.(KNN)
from sklearn.neighbors import KNeighborsClassifier

model_knn = KNeighborsClassifier()
model_knn.fit(X_train_s, y_train)
pred_knn = model_knn.predict(X_valid_s)
```

In [16]:
```
# 여기에 답안코드를 작성하세요.(Decision Tree)
from sklearn.tree import DecisionTreeClassifier
```

```
model_dt = DecisionTreeClassifier()
model_dt.fit(X_train, y_train)
pred_dt = model_dt.predict(X_valid)
```

In [17]:
```
# 여기에 답안코드를 작성하세요.(RandomForest)
from sklearn.ensemble import RandomForestClassifier

model_rf = RandomForestClassifier()
model_rf.fit(X_train, y_train)
pred_rf = model_rf.predict(X_valid)
```

In [18]:
```
# 여기에 답안코드를 작성하세요.(XgBoost)
from xgboost import XGBClassifier

model_xgb = XGBClassifier()
model_xgb.fit(X_train, y_train)
pred_xgb = model_xgb.predict(X_valid)
```

In [19]:
```
# 여기에 답안코드를 작성하세요.(lightgbm)
from lightgbm import LGBMClassifier

model_lgbm = LGBMClassifier()
model_lgbm.fit(X_train, y_train)
pred_lgbm = model_lgbm.predict(X_valid)
```

```
[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Info] Number of positive: 1518, number of negative: 4103
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000214
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 631
[LightGBM] [Info] Number of data points in the train set: 5621, number of used features: 27
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.270059 -> initscore=-0.994325
[LightGBM] [Info] Start training from score -0.994325
```

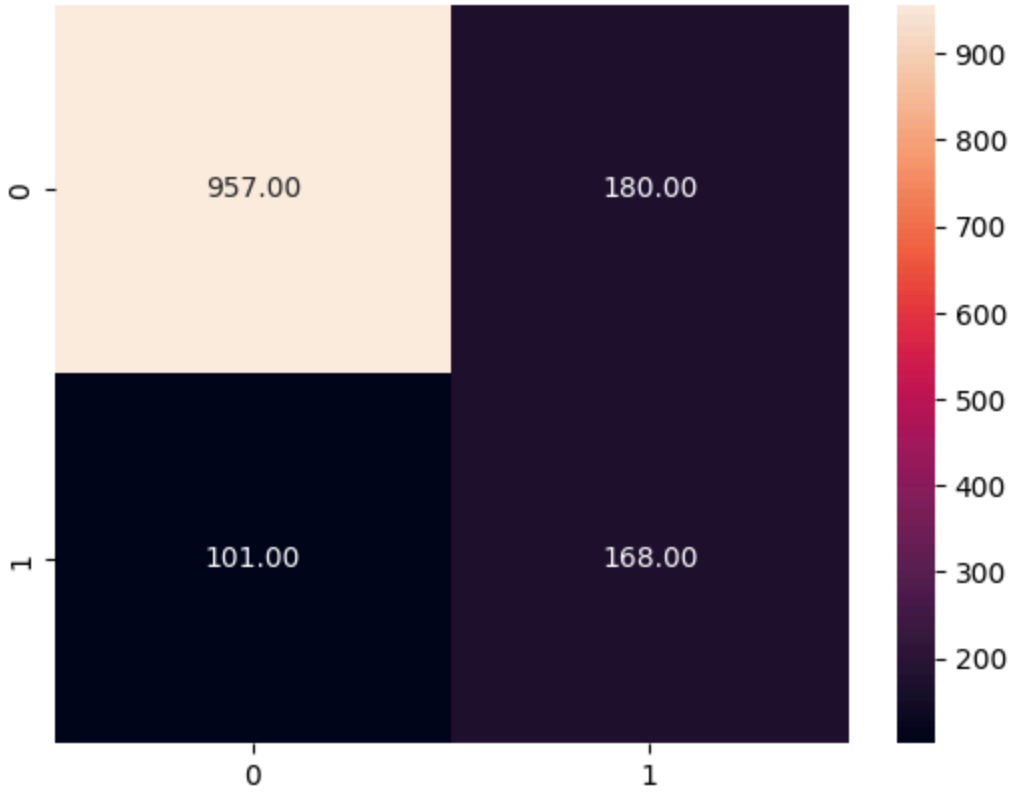
14. 바로 위 모델의 성능을 평가하려고 합니다.

y값을 예측하여 **confusion matrix**를 구하고 **heatmap** 그래프로 시각화하세요.

그리고 **Scikit-learn**의 **classification_report**를 활용하여 성능을 출력하세요.

In [20]:
```
# 여기에 답안코드를 작성하세요.
from sklearn.metrics import *

con_matrix = confusion_matrix(pred_rf, y_valid)
print(con_matrix)

sns.heatmap(con_matrix, annot=True, fmt='.2f')
plt.show()

print(classification_report(pred_rf, y_valid))
```

```
[[957 180]
 [101 168]]
```



	precision	recall	f1-score	support
0	0.90	0.84	0.87	1137
1	0.48	0.62	0.54	269
accuracy			0.80	1406
macro avg	0.69	0.73	0.71	1406
weighted avg	0.82	0.80	0.81	1406

다음 문항을 풀기 전에 아래 코드를 실행하세요.

In [21]:
```
!pip install tensorflow
```

```
Requirement already satisfied: tensorflow in c:\users\user\anaconda3\lib\site-packages (2.16.
1)
Requirement already satisfied: tensorflow-intel==2.16.1 in c:\users\user\anaconda3\lib\site-pa
ckages (from tensorflow) (2.16.1)
Requirement already satisfied: absl-py>=1.0.0 in c:\users\user\anaconda3\lib\site-packages (fr
om tensorflow-intel==2.16.1->tensorflow) (2.1.0)
Requirement already satisfied: astunparse>=1.6.0 in c:\users\user\anaconda3\lib\site-packages
(from tensorflow-intel==2.16.1->tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=23.5.26 in c:\users\user\anaconda3\lib\site-packag
es (from tensorflow-intel==2.16.1->tensorflow) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in c:\users\user\anaconda3
\lib\site-packages (from tensorflow-intel==2.16.1->tensorflow) (0.5.4)
Requirement already satisfied: google-pasta>=0.1.1 in c:\users\user\anaconda3\lib\site-package
s (from tensorflow-intel==2.16.1->tensorflow) (0.2.0)
Requirement already satisfied: h5py>=3.10.0 in c:\users\user\anaconda3\lib\site-packages (from
tensorflow-intel==2.16.1->tensorflow) (3.11.0)
Requirement already satisfied: libclang>=13.0.0 in c:\users\user\anaconda3\lib\site-packages
(from tensorflow-intel==2.16.1->tensorflow) (18.1.1)
Requirement already satisfied: ml-dtypes~=0.3.1 in c:\users\user\anaconda3\lib\site-packages
(from tensorflow-intel==2.16.1->tensorflow) (0.3.2)
Requirement already satisfied: opt-einsum>=2.3.2 in c:\users\user\anaconda3\lib\site-packages
(from tensorflow-intel==2.16.1->tensorflow) (3.3.0)
Requirement already satisfied: packaging in c:\users\user\anaconda3\lib\site-packages (from te
nsorflow-intel==2.16.1->tensorflow) (23.1)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<
5.0.0dev,>=3.20.3 in c:\users\user\anaconda3\lib\site-packages (from tensorflow-intel==2.16.1-
>tensorflow) (4.25.2)
Requirement already satisfied: requests<3,>=2.21.0 in c:\users\user\anaconda3\lib\site-package
s (from tensorflow-intel==2.16.1->tensorflow) (2.31.0)
Requirement already satisfied: setuptools in c:\users\user\anaconda3\lib\site-packages (from t
ensorflow-intel==2.16.1->tensorflow) (68.0.0)
Requirement already satisfied: six>=1.12.0 in c:\users\user\anaconda3\lib\site-packages (from
tensorflow-intel==2.16.1->tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in c:\users\user\anaconda3\lib\site-packages
(from tensorflow-intel==2.16.1->tensorflow) (2.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in c:\users\user\anaconda3\lib\site-pa
ckages (from tensorflow-intel==2.16.1->tensorflow) (4.11.0)
Requirement already satisfied: wrapt>=1.11.0 in c:\users\user\anaconda3\lib\site-packages (fro
m tensorflow-intel==2.16.1->tensorflow) (1.14.1)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in c:\users\user\anaconda3\lib\site-package
s (from tensorflow-intel==2.16.1->tensorflow) (1.62.2)
Requirement already satisfied: tensorboard<2.17,>=2.16 in c:\users\user\anaconda3\lib\site-pac
kages (from tensorflow-intel==2.16.1->tensorflow) (2.16.2)
Requirement already satisfied: keras>=3.0.0 in c:\users\user\anaconda3\lib\site-packages (from
tensorflow-intel==2.16.1->tensorflow) (3.2.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in c:\users\user\anaconda3
\lib\site-packages (from tensorflow-intel==2.16.1->tensorflow) (0.31.0)
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in c:\users\user\anaconda3\lib\site-packag
es (from tensorflow-intel==2.16.1->tensorflow) (1.24.3)
Requirement already satisfied: wheel<1.0,>=0.23.0 in c:\users\user\anaconda3\lib\site-packages
(from astunparse>=1.6.0->tensorflow-intel==2.16.1->tensorflow) (0.38.4)
Requirement already satisfied: rich in c:\users\user\anaconda3\lib\site-packages (from keras>=
3.0.0->tensorflow-intel==2.16.1->tensorflow) (13.7.0)
Requirement already satisfied: namex in c:\users\user\anaconda3\lib\site-packages (from keras>
=3.0.0->tensorflow-intel==2.16.1->tensorflow) (0.0.8)
Requirement already satisfied: optree in c:\users\user\anaconda3\lib\site-packages (from keras
>=3.0.0->tensorflow-intel==2.16.1->tensorflow) (0.11.0)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\user\anaconda3\lib\site-pa
ckages (from requests<3,>=2.21.0->tensorflow-intel==2.16.1->tensorflow) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\user\anaconda3\lib\site-packages (from
```

```
requests<3,>=2.21.0->tensorflow-intel==2.16.1->tensorflow) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\user\anaconda3\lib\site-packages
(from requests<3,>=2.21.0->tensorflow-intel==2.16.1->tensorflow) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\user\anaconda3\lib\site-packages
(from requests<3,>=2.21.0->tensorflow-intel==2.16.1->tensorflow) (2024.2.2)
Requirement already satisfied: markdown>=2.6.8 in c:\users\user\anaconda3\lib\site-packages (f
rom tensorboard<2.17,>=2.16->tensorflow-intel==2.16.1->tensorflow) (3.4.1)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in c:\users\user\anaconda
3\lib\site-packages (from tensorboard<2.17,>=2.16->tensorflow-intel==2.16.1->tensorflow) (0.7.
2)
Requirement already satisfied: werkzeug>=1.0.1 in c:\users\user\anaconda3\lib\site-packages (f
rom tensorboard<2.17,>=2.16->tensorflow-intel==2.16.1->tensorflow) (2.2.3)
Requirement already satisfied: MarkupSafe>=2.1.1 in c:\users\user\anaconda3\lib\site-packages
(from werkzeug>=1.0.1->tensorboard<2.17,>=2.16->tensorflow-intel==2.16.1->tensorflow) (2.1.1)
Requirement already satisfied: markdown-it-py>=2.2.0 in c:\users\user\anaconda3\lib\site-packa
ges (from rich->keras>=3.0.0->tensorflow-intel==2.16.1->tensorflow) (2.2.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in c:\users\user\anaconda3\lib\site-pac
kages (from rich->keras>=3.0.0->tensorflow-intel==2.16.1->tensorflow) (2.15.1)
Requirement already satisfied: mdurl~=0.1 in c:\users\user\anaconda3\lib\site-packages (from m
arkdown-it-py>=2.2.0->rich->keras>=3.0.0->tensorflow-intel==2.16.1->tensorflow) (0.1.0)
```

In [33]:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Dense, Activation, Dropout, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.utils import to_categorical
from keras.backend import clear_session
from keras.optimizers import Adam

tf.random.set_seed(1)
```

15. 고객 이탈여부를 예측하는 딥러닝 모델을 만들려고 합니다.

아래 가이드에 따라 모델링하고 학습을 진행하세요.

- Tensoflow framework를 사용하여 딥러닝 모델을 만드세요.
- 히든레이어(hidden layer) 2개이상으로 모델을 구성하세요.
- dropout 비율 0.2로 Dropout 레이어 1개를 추가해 주세요.
- 하이퍼파라미터 epochs: 30, batch_size: 16으로 설정해주세요.
- 각 에포크마다 loss와 metrics 평가하기 위한 데이터로 X_valid, y_valid 사용하세요.
- 학습정보는 history 변수에 저장해주세요

In [127…

```
# 여기에 답안코드를 작성하세요.
clear_session()
n = X_train.shape[1] # feature 갯수

model = Sequential([Dense(256, input_shape=(n, ), activation='relu'),
                    Dense(256, activation='relu'),
                    Dense(128, activation='relu'),
                    Dense(128, activation='relu'),
                    Dense(64, activation='relu'),
                    Dense(64, activation='relu'),
                    Dense(32, activation='relu'),
                    Dense(32, activation='relu'),
                    Dense(16, activation='relu'),
```

```
                    Dense(16, activation='relu'),
                    Dropout(0.2),
                    Dense(1, activation='sigmoid')
                   ])

model.summary()
model.compile(optimizer=Adam(0.001), loss='binary_crossentropy', metrics=['accuracy'])

es = EarlyStopping(monitor='val_loss', patience=5)
mcp = ModelCheckpoint('best_model.keras', monitor='val_loss', verbose=1, save_best_only=True)

history = model.fit(X_train, y_train, epochs=30, validation_data=(X_valid, y_valid), batch_siz
```

```
C:\Users\User\anaconda3\Lib\site-packages\keras\src\layers\core\dense.py:86: UserWarning: Do n
ot pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer
using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Pa
dense (Dense)	(None, 256)	
dense_1 (Dense)	(None, 256)	6
dense_2 (Dense)	(None, 128)	3
dense_3 (Dense)	(None, 128)	1
dense_4 (Dense)	(None, 64)	
dense_5 (Dense)	(None, 64)	
dense_6 (Dense)	(None, 32)	
dense_7 (Dense)	(None, 32)	
dense_8 (Dense)	(None, 16)	
dense_9 (Dense)	(None, 16)	
dropout (Dropout)	(None, 16)	
dense_10 (Dense)	(None, 1)	

Total params: 138,737 (541.94 KB)

Trainable params: 138,737 (541.94 KB)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/30
342/352 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7091 - loss: 0.9848
Epoch 1: val_loss improved from inf to 0.55989, saving model to best_model.keras
352/352 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.7099 - loss: 0.9758 - val_accuracy: 0.7
781 - val_loss: 0.5599
Epoch 2/30
349/352 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7673 - loss: 0.5511
Epoch 2: val_loss improved from 0.55989 to 0.50401, saving model to best_model.keras
352/352 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7673 - loss: 0.5510 - val_accuracy: 0.7
767 - val_loss: 0.5040
Epoch 3/30
339/352 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7702 - loss: 0.5379
Epoch 3: val_loss did not improve from 0.50401
352/352 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7702 - loss: 0.5377 - val_accuracy: 0.7
745 - val_loss: 0.5486
Epoch 4/30
342/352 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7733 - loss: 0.5274
Epoch 4: val_loss improved from 0.50401 to 0.49218, saving model to best_model.keras
352/352 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7733 - loss: 0.5270 - val_accuracy: 0.7
994 - val_loss: 0.4922
Epoch 5/30
343/352 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7739 - loss: 0.5125
Epoch 5: val_loss did not improve from 0.49218
352/352 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7739 - loss: 0.5123 - val_accuracy: 0.7
881 - val_loss: 0.4986
Epoch 6/30
348/352 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7785 - loss: 0.5010
Epoch 6: val_loss did not improve from 0.49218
352/352 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7786 - loss: 0.5010 - val_accuracy: 0.7
852 - val_loss: 0.5294
Epoch 7/30
342/352 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7791 - loss: 0.4920
Epoch 7: val_loss improved from 0.49218 to 0.48869, saving model to best_model.keras
352/352 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7790 - loss: 0.4919 - val_accuracy: 0.7
838 - val_loss: 0.4887
Epoch 8/30
341/352 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7756 - loss: 0.5001
Epoch 8: val_loss improved from 0.48869 to 0.48231, saving model to best_model.keras
352/352 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7755 - loss: 0.4999 - val_accuracy: 0.7
888 - val_loss: 0.4823
Epoch 9/30
344/352 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7826 - loss: 0.4816
Epoch 9: val_loss improved from 0.48231 to 0.47168, saving model to best_model.keras
352/352 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.7826 - loss: 0.4815 - val_accuracy: 0.7
959 - val_loss: 0.4717
Epoch 10/30
347/352 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7870 - loss: 0.4719
Epoch 10: val_loss did not improve from 0.47168
352/352 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.7869 - loss: 0.4720 - val_accuracy: 0.7
838 - val_loss: 0.4733
Epoch 11/30
342/352 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7840 - loss: 0.4732
Epoch 11: val_loss improved from 0.47168 to 0.47119, saving model to best_model.keras
352/352 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.7840 - loss: 0.4731 - val_accuracy: 0.7
916 - val_loss: 0.4712
Epoch 12/30
352/352 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7833 - loss: 0.4696
Epoch 12: val_loss improved from 0.47119 to 0.46460, saving model to best_model.keras
352/352 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.7833 - loss: 0.4696 - val_accuracy: 0.7
945 - val_loss: 0.4646
```

```
Epoch 13/30
345/352 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7851 - loss: 0.4666
Epoch 13: val_loss did not improve from 0.46460
352/352 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7851 - loss: 0.4667 - val_accuracy: 0.7
987 - val_loss: 0.4663
Epoch 14/30
345/352 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7802 - loss: 0.4703
Epoch 14: val_loss did not improve from 0.46460
352/352 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.7803 - loss: 0.4702 - val_accuracy: 0.8
009 - val_loss: 0.4769
Epoch 15/30
352/352 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7866 - loss: 0.4587
Epoch 15: val_loss did not improve from 0.46460
352/352 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.7866 - loss: 0.4587 - val_accuracy: 0.8
001 - val_loss: 0.4778
Epoch 16/30
351/352 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7849 - loss: 0.4622
Epoch 16: val_loss improved from 0.46460 to 0.45462, saving model to best_model.keras
352/352 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.7849 - loss: 0.4622 - val_accuracy: 0.7
952 - val_loss: 0.4546
Epoch 17/30
344/352 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7806 - loss: 0.4673
Epoch 17: val_loss did not improve from 0.45462
352/352 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.7806 - loss: 0.4672 - val_accuracy: 0.8
037 - val_loss: 0.4625
Epoch 18/30
352/352 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7890 - loss: 0.4677
Epoch 18: val_loss did not improve from 0.45462
352/352 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7890 - loss: 0.4676 - val_accuracy: 0.7
994 - val_loss: 0.4954
Epoch 19/30
351/352 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7833 - loss: 0.4670
Epoch 19: val_loss did not improve from 0.45462
352/352 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.7833 - loss: 0.4670 - val_accuracy: 0.7
909 - val_loss: 0.4783
Epoch 20/30
339/352 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7872 - loss: 0.4642
Epoch 20: val_loss improved from 0.45462 to 0.44720, saving model to best_model.keras
352/352 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.7872 - loss: 0.4640 - val_accuracy: 0.8
009 - val_loss: 0.4472
Epoch 21/30
341/352 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7892 - loss: 0.4541
Epoch 21: val_loss did not improve from 0.44720
352/352 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.7892 - loss: 0.4539 - val_accuracy: 0.8
001 - val_loss: 0.5128
Epoch 22/30
348/352 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7896 - loss: 0.4603
Epoch 22: val_loss improved from 0.44720 to 0.44048, saving model to best_model.keras
352/352 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.7895 - loss: 0.4603 - val_accuracy: 0.7
902 - val_loss: 0.4405
Epoch 23/30
350/352 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7841 - loss: 0.4536
Epoch 23: val_loss did not improve from 0.44048
352/352 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.7842 - loss: 0.4536 - val_accuracy: 0.8
051 - val_loss: 0.4519
Epoch 24/30
337/352 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7903 - loss: 0.4487
Epoch 24: val_loss improved from 0.44048 to 0.42708, saving model to best_model.keras
352/352 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.7903 - loss: 0.4485 - val_accuracy: 0.7
945 - val_loss: 0.4271
```

```
Epoch 25/30
351/352 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7890 - loss: 0.4429
Epoch 25: val_loss did not improve from 0.42708
352/352 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7890 - loss: 0.4429 - val_accuracy: 0.8
023 - val_loss: 0.4414
Epoch 26/30
350/352 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7833 - loss: 0.4508
Epoch 26: val_loss did not improve from 0.42708
352/352 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.7834 - loss: 0.4508 - val_accuracy: 0.7
937 - val_loss: 0.4528
Epoch 27/30
351/352 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7888 - loss: 0.4516
Epoch 27: val_loss did not improve from 0.42708
352/352 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.7888 - loss: 0.4516 - val_accuracy: 0.8
001 - val_loss: 0.4393
Epoch 28/30
337/352 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7889 - loss: 0.4471
Epoch 28: val_loss did not improve from 0.42708
352/352 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.7891 - loss: 0.4469 - val_accuracy: 0.8
058 - val_loss: 0.4533
Epoch 29/30
347/352 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7903 - loss: 0.4477
Epoch 29: val_loss did not improve from 0.42708
352/352 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.7903 - loss: 0.4477 - val_accuracy: 0.8
```

16. 위 딥러닝 모델의 성능을 평가하려고 합니다.

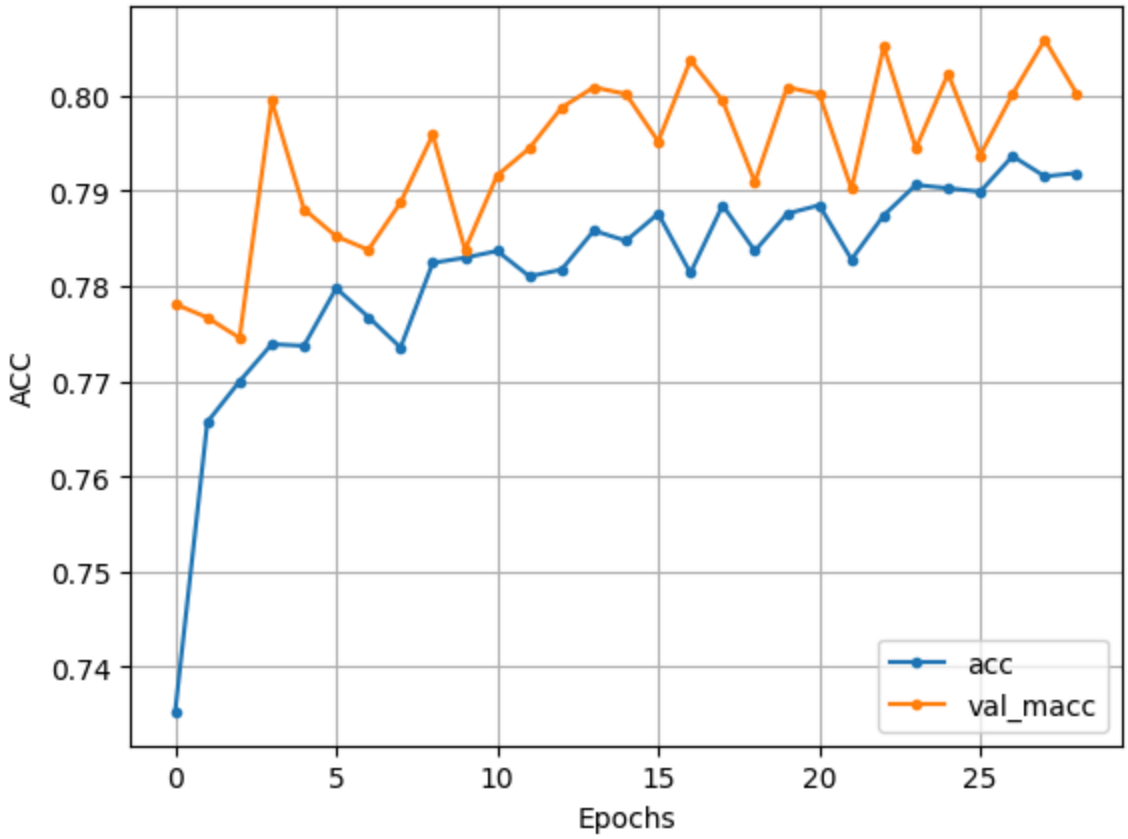
Matplotlib 라이브러리 활용해서 학습 accuracy와 검증 accuracy를 그래프로 표시하세요.

- 1개의 그래프에 학습 accuracy와 검증 accuracy 2가지를 모두 표시하세요.
- 위 2가지 각각의 범례를 'acc', 'val_macc'로 표시하세요.
- 그래프의 타이틀은 'Accuracy'로 표시하세요.
- X축에는 'Epochs'라고 표시하고 Y축에는 'Acc'라고 표시하세요.

In [128…

```python
# 여기에 답안코드를 작성하세요.
pred = model.predict(X_valid_s)
plt.plot(history['accuracy'], label='acc', marker='.')
plt.plot(history['val_accuracy'], label='val_macc', marker='.')
plt.xlabel('Epochs')
plt.ylabel('ACC')
plt.legend()
plt.grid()
```

```
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
```



In [129…
```
pred = model.predict(X_valid)

pred = np.where(pred>=0.5, 1, 0)

print(accuracy_score(y_valid, pred))
```

```
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
0.8001422475106685
```

In [ ]: