TEST7

```
import pandas as pd
from sklearn.model_selection import train_test_split

# CSV 파일에서 데이터 불러오기
data1 = pd.read_csv('C:/Users/Hesong/test7/train_data_7.csv')

# 'DAY'는 수치형 데이터이므로 따로 선택
numeric_feature = ['TPRSC_CAPA', 'DAY']
X_numeric = data1[numeric_feature]

# 'INGR_NM'을 원핫인코딩하여 선택
categorical_features = ['INGR_NM', 'ANTBT_NM', 'antibiotics']
X_categorical = pd.get_dummies(data1[categorical_features], columns=categorical_features)

# 두 데이터프레임을 합쳐 입력 데이터 생성
X = pd.concat([X_categorical, X_numeric], axis=1)

# "RSLT_CONT" 값을 숫자로 변환
y = data1['RSLT_CONT'].map({'R': 1, 'S': 0})

# 학습 데이터와 테스트 데이터 분할
X_train7, X_test7, y_train7, y_test7 = train_test_split(X, y, test_size=0.3, stratify=y, random_state=42)

# 나눈 데이터의 크기 출력
print("학습 데이터:", X_train7.shape)
print("테스트 데이터:", X_test7.shape)
print("학습 레이블:", y_train7.shape)
print("테스트 레이블:", y_test7.shape)

# 학습 데이터와 테스트 데이터를 CSV 파일로 저장
X_train7.to_csv('C:/Users/Hesong/test7/X_train7.csv', index=False)
X_test7.to_csv('C:/Users/Hesong/test7/X_test7.csv', index=False)
y_train7.to_csv('C:/Users/Hesong/test7/y_train7.csv', index=False, header=['RSLT_CONT'])
y_test7.to_csv('C:/Users/Hesong/test7/y_test7.csv', index=False, header=['RSLT_CONT'])
```

```
학습 데이터: (419302, 133)
테스트 데이터: (179701, 133)
학습 레이블: (419302,)
테스트 레이블: (179701,)
```

```
import pandas as pd

# CSV 파일로부터 데이터 불러오기
X_train1 = pd.read_csv('C:/Users/Hesong/test7/X_train7.csv')
X_test1 = pd.read_csv('C:/Users/Hesong/test7/X_test7.csv')
y_train1 = pd.read_csv('C:/Users/Hesong/test7/y_train7.csv')['RSLT_CONT']
y_test1 = pd.read_csv('C:/Users/Hesong/test7/y_test7.csv')['RSLT_CONT']
```

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import roc_auc_score, accuracy_score, precision_score, recall_score, f1_score

# 모델 생성
model = Sequential()

# 입력층과 첫 번째 은닉층
model.add(Dense(units=8, activation='relu', input_dim=X_train7.shape[1]))

# 두 번째 은닉층
model.add(Dense(units=4, activation='relu'))

# 출력층
model.add(Dense(units=1, activation='sigmoid'))

# 모델 컴파일
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# 모델 요약 정보 출력
model.summary()

# 모델 훈련
model.fit(X_train7, y_train7, epochs=30, batch_size=32, validation_data=(X_test7, y_test7))

# 모델 평가
```

```
loss, accuracy = model.evaluate(X_test7, y_test7)

# 모델 예측
y_pred = model.predict(X_test7)
y_pred_binary = (y_pred > 0.5).astype(int)  # 확률을 이진 클래스로 변환

# 결과 출력
print(y_pred)

# AUROC 계산
roc_auc = roc_auc_score(y_test7, y_pred)

# 평가 지표 계산
accuracy = accuracy_score(y_test7, y_pred_binary)
precision = precision_score(y_test7, y_pred_binary, average='weighted')
recall = recall_score(y_test7, y_pred_binary, average='weighted')
f1 = f1_score(y_test7, y_pred_binary, average='weighted')

# 결과 출력
print("AUROC:", roc_auc)
print("accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 8)                 1072

 dense_1 (Dense)             (None, 4)                 36

 dense_2 (Dense)             (None, 1)                 5

=================================================================
Total params: 1113 (4.35 KB)
Trainable params: 1113 (4.35 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/30
13104/13104 [==============================] - 24s 2ms/step - loss: 0.7588 - accuracy: 0.6986 - val_loss: 1.1856 - val_accuracy: 0.7076
Epoch 2/30
13104/13104 [==============================] - 21s 2ms/step - loss: 0.6422 - accuracy: 0.7157 - val_loss: 0.5301 - val_accuracy: 0.7390
Epoch 3/30
13104/13104 [==============================] - 21s 2ms/step - loss: 0.5803 - accuracy: 0.7178 - val_loss: 0.5479 - val_accuracy: 0.7278
Epoch 4/30
13104/13104 [==============================] - 24s 2ms/step - loss: 0.5596 - accuracy: 0.7252 - val_loss: 0.5428 - val_accuracy: 0.7321
Epoch 5/30
13104/13104 [==============================] - 26s 2ms/step - loss: 0.5527 - accuracy: 0.7270 - val_loss: 0.5449 - val_accuracy: 0.7283
Epoch 6/30
13104/13104 [==============================] - 24s 2ms/step - loss: 0.5470 - accuracy: 0.7292 - val_loss: 0.5303 - val_accuracy: 0.7377
Epoch 7/30
13104/13104 [==============================] - 24s 2ms/step - loss: 0.5427 - accuracy: 0.7308 - val_loss: 0.5403 - val_accuracy: 0.7314
Epoch 8/30
13104/13104 [==============================] - 24s 2ms/step - loss: 0.5406 - accuracy: 0.7316 - val_loss: 0.5406 - val_accuracy: 0.7296
Epoch 9/30
13104/13104 [==============================] - 22s 2ms/step - loss: 0.5395 - accuracy: 0.7314 - val_loss: 0.5419 - val_accuracy: 0.7191
Epoch 10/30
13104/13104 [==============================] - 25s 2ms/step - loss: 0.5372 - accuracy: 0.7321 - val_loss: 0.5238 - val_accuracy: 0.7413
Epoch 11/30
13104/13104 [==============================] - 22s 2ms/step - loss: 0.5364 - accuracy: 0.7329 - val_loss: 0.5319 - val_accuracy: 0.7331
Epoch 12/30
13104/13104 [==============================] - 23s 2ms/step - loss: 0.5357 - accuracy: 0.7331 - val_loss: 0.5504 - val_accuracy: 0.7317
Epoch 13/30
13104/13104 [==============================] - 23s 2ms/step - loss: 0.5334 - accuracy: 0.7345 - val_loss: 0.5241 - val_accuracy: 0.7405
Epoch 14/30
13104/13104 [==============================] - 26s 2ms/step - loss: 0.5334 - accuracy: 0.7335 - val_loss: 0.5252 - val_accuracy: 0.7384
Epoch 15/30
13104/13104 [==============================] - 20s 2ms/step - loss: 0.5327 - accuracy: 0.7343 - val_loss: 0.5261 - val_accuracy: 0.7418
Epoch 16/30
13104/13104 [==============================] - 25s 2ms/step - loss: 0.5319 - accuracy: 0.7339 - val_loss: 0.5477 - val_accuracy: 0.7255
Epoch 17/30
13104/13104 [==============================] - 21s 2ms/step - loss: 0.5320 - accuracy: 0.7342 - val_loss: 0.5345 - val_accuracy: 0.7281
Epoch 18/30
13104/13104 [==============================] - 20s 2ms/step - loss: 0.5322 - accuracy: 0.7350 - val_loss: 0.5424 - val_accuracy: 0.7317
Epoch 19/30
13104/13104 [==============================] - 23s 2ms/step - loss: 0.5306 - accuracy: 0.7348 - val_loss: 0.5304 - val_accuracy: 0.7367
Epoch 20/30
13104/13104 [==============================] - 21s 2ms/step - loss: 0.5303 - accuracy: 0.7358 - val_loss: 0.5367 - val_accuracy: 0.7364
Epoch 21/30
13104/13104 [==============================] - 21s 2ms/step - loss: 0.5294 - accuracy: 0.7354 - val_loss: 0.5304 - val_accuracy: 0.7308
Epoch 22/30
```

TEST8

```
import pandas as pd
from sklearn.model_selection import train_test_split

# CSV 파일에서 데이터 불러오기
data8 = pd.read_csv('C:/Users/Hesong/test8/train_data_8.csv')

# 'DAY'는 수치형 데이터이므로 따로 선택
numeric_feature = ['TPRSC_CAPA', 'DAY']
X_numeric = data8[numeric_feature]

# 'INGR_NM'을 원핫인코딩하여 선택
categorical_features = ['INGR_NM', 'ANTBT_NM', 'antibiotics']
X_categorical = pd.get_dummies(data8[categorical_features], columns=categorical_features)

# 두 데이터프레임을 합쳐 입력 데이터 생성
X = pd.concat([X_categorical, X_numeric], axis=1)

# "RSLT_CONT" 값을 숫자로 변환
y = data8['RSLT_CONT'].map({'R': 1, 'S': 0})

# 학습 데이터와 테스트 데이터 분할
X_train8, X_test8, y_train8, y_test8 = train_test_split(X, y, test_size=0.3, stratify=y, random_state=42)

# 나눈 데이터의 크기 출력
print("학습 데이터:", X_train8.shape)
print("테스트 데이터:", X_test8.shape)
print("학습 레이블:", y_train8.shape)
print("테스트 레이블:", y_test8.shape)

# 학습 데이터와 테스트 데이터를 CSV 파일로 저장
X_train8.to_csv('C:/Users/Hesong/test8/X_train8.csv', index=False)
X_test8.to_csv('C:/Users/Hesong/test8/X_test8.csv', index=False)
y_train8.to_csv('C:/Users/Hesong/test8/y_train8.csv', index=False, header=['RSLT_CONT'])
y_test8.to_csv('C:/Users/Hesong/test8/y_test8.csv', index=False, header=['RSLT_CONT'])
```

```
학습 데이터: (419302, 133)
테스트 데이터: (179701, 133)
학습 레이블: (419302,)
테스트 레이블: (179701,)
```

```
import pandas as pd

# CSV 파일로부터 데이터 불러오기
X_train8 = pd.read_csv('C:/Users/Hesong/test8/X_train8.csv')
X_test8 = pd.read_csv('C:/Users/Hesong/test8/X_test8.csv')
y_train8 = pd.read_csv('C:/Users/Hesong/test8/y_train8.csv')['RSLT_CONT']
y_test8 = pd.read_csv('C:/Users/Hesong/test8/y_test8.csv')['RSLT_CONT']
```

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import roc_auc_score, accuracy_score, precision_score, recall_score, f1_score

# 모델 생성
model = Sequential()

# 입력층과 첫 번째 은닉층
model.add(Dense(units=8, activation='relu', input_dim=X_train8.shape[1]))

# 두 번째 은닉층
model.add(Dense(units=4, activation='relu'))

# 출력층
model.add(Dense(units=1, activation='sigmoid'))

# 모델 컴파일
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# 모델 요약 정보 출력
model.summary()

# 모델 훈련
model.fit(X_train8, y_train8, epochs=30, batch_size=32, validation_data=(X_test8, y_test8))
```

```
# 모델 평가
loss, accuracy = model.evaluate(X_test8, y_test8)

# 모델 예측
y_pred = model.predict(X_test8)
y_pred_binary = (y_pred > 0.5).astype(int)  # 확률을 이진 클래스로 변환

# 결과 출력
print(y_pred)

# AUROC 계산
roc_auc = roc_auc_score(y_test8, y_pred)

# 평가 지표 계산
accuracy = accuracy_score(y_test8, y_pred_binary)
precision = precision_score(y_test8, y_pred_binary, average='weighted')
recall = recall_score(y_test8, y_pred_binary, average='weighted')
f1 = f1_score(y_test8, y_pred_binary, average='weighted')

# 결과 출력
print("AUROC:", roc_auc)
print("accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 8)                 1072

 dense_4 (Dense)             (None, 4)                 36

 dense_5 (Dense)             (None, 1)                 5

=================================================================
Total params: 1113 (4.35 KB)
Trainable params: 1113 (4.35 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/30
13104/13104 [==============================] - 38s 3ms/step - loss: 0.6771 - accuracy: 0.7056 - val_loss: 0.5452 - val_accuracy: 0.7325
Epoch 2/30
13104/13104 [==============================] - 36s 3ms/step - loss: 0.5841 - accuracy: 0.7212 - val_loss: 0.5306 - val_accuracy: 0.7398
Epoch 3/30
13104/13104 [==============================] - 36s 3ms/step - loss: 0.5570 - accuracy: 0.7242 - val_loss: 0.5449 - val_accuracy: 0.7121
Epoch 4/30
13104/13104 [==============================] - 27s 2ms/step - loss: 0.5491 - accuracy: 0.7231 - val_loss: 0.5496 - val_accuracy: 0.7282
Epoch 5/30
13104/13104 [==============================] - 21s 2ms/step - loss: 0.5453 - accuracy: 0.7254 - val_loss: 0.5371 - val_accuracy: 0.7270
Epoch 6/30
13104/13104 [==============================] - 21s 2ms/step - loss: 0.5421 - accuracy: 0.7278 - val_loss: 0.5598 - val_accuracy: 0.7249
Epoch 7/30
13104/13104 [==============================] - 21s 2ms/step - loss: 0.5405 - accuracy: 0.7283 - val_loss: 0.5328 - val_accuracy: 0.7283
Epoch 8/30
13104/13104 [==============================] - 21s 2ms/step - loss: 0.5391 - accuracy: 0.7289 - val_loss: 0.5501 - val_accuracy: 0.7329
Epoch 9/30
13104/13104 [==============================] - 18s 1ms/step - loss: 0.5388 - accuracy: 0.7293 - val_loss: 0.5344 - val_accuracy: 0.7323
Epoch 10/30
13104/13104 [==============================] - 17s 1ms/step - loss: 0.5372 - accuracy: 0.7298 - val_loss: 0.5697 - val_accuracy: 0.6943
Epoch 11/30
13104/13104 [==============================] - 20s 2ms/step - loss: 0.5359 - accuracy: 0.7306 - val_loss: 0.5274 - val_accuracy: 0.7332
Epoch 12/30
13104/13104 [==============================] - 21s 2ms/step - loss: 0.5346 - accuracy: 0.7317 - val_loss: 0.5392 - val_accuracy: 0.7385
Epoch 13/30
13104/13104 [==============================] - 18s 1ms/step - loss: 0.5341 - accuracy: 0.7316 - val_loss: 0.5347 - val_accuracy: 0.7399
Epoch 14/30
13104/13104 [==============================] - 19s 1ms/step - loss: 0.5339 - accuracy: 0.7323 - val_loss: 0.5388 - val_accuracy: 0.7357
Epoch 15/30
13104/13104 [==============================] - 22s 2ms/step - loss: 0.5325 - accuracy: 0.7327 - val_loss: 0.5583 - val_accuracy: 0.7006
Epoch 16/30
13104/13104 [==============================] - 37s 3ms/step - loss: 0.5335 - accuracy: 0.7325 - val_loss: 0.5451 - val_accuracy: 0.7268
Epoch 17/30
13104/13104 [==============================] - 38s 3ms/step - loss: 0.5324 - accuracy: 0.7328 - val_loss: 0.5622 - val_accuracy: 0.7277
Epoch 18/30
13104/13104 [==============================] - 39s 3ms/step - loss: 0.5318 - accuracy: 0.7332 - val_loss: 0.5312 - val_accuracy: 0.7305
Epoch 19/30
13104/13104 [==============================] - 39s 3ms/step - loss: 0.5308 - accuracy: 0.7344 - val_loss: 0.5289 - val_accuracy: 0.7380
Epoch 20/30
13104/13104 [==============================] - 38s 3ms/step - loss: 0.5308 - accuracy: 0.7338 - val_loss: 0.5256 - val_accuracy: 0.7330
Epoch 21/30
13104/13104 [==============================] - 38s 3ms/step - loss: 0.5310 - accuracy: 0.7339 - val_loss: 0.5268 - val_accuracy: 0.7311
Epoch 22/30
```

TEST9

```
import pandas as pd
from sklearn.model_selection import train_test_split

# CSV 파일에서 데이터 불러오기
data9 = pd.read_csv('C:/Users/Hesong/test9/train_data_9.csv')

# 'DAY','TPRSC_CAPA'는 수치형 데이터이므로 따로 선택
numeric_feature = ['TPRSC_CAPA', 'DAY']
X_numeric = data9[numeric_feature]

# 'INGR_NM', 'ANTBT_NM', 'antibiotics'을 원핫인코딩하여 선택
categorical_features = ['INGR_NM', 'ANTBT_NM', 'antibiotics']
X_categorical = pd.get_dummies(data9[categorical_features], columns=categorical_features)

# 두 데이터프레임을 합쳐 입력 데이터 생성
X = pd.concat([X_categorical, X_numeric], axis=1)

# "RSLT_CONT" 값을 숫자로 변환
y = data9['RSLT_CONT'].map({'R': 1, 'S': 0})

# 학습 데이터와 테스트 데이터 분할
X_train9, X_test9, y_train9, y_test9 = train_test_split(X, y, test_size=0.3, stratify=y, random_state=42)

# 나눈 데이터의 크기 출력
print("학습 데이터:", X_train9.shape)
print("테스트 데이터:", X_test9.shape)
print("학습 레이블:", y_train9.shape)
print("테스트 레이블:", y_test9.shape)

# 학습 데이터와 테스트 데이터를 CSV 파일로 저장
X_train9.to_csv('C:/Users/Hesong/test9/X_train9.csv', index=False)
X_test9.to_csv('C:/Users/Hesong/test9/X_test9.csv', index=False)
y_train9.to_csv('C:/Users/Hesong/test9/y_train9.csv', index=False, header=['RSLT_CONT'])
y_test9.to_csv('C:/Users/Hesong/test9/y_test9.csv', index=False, header=['RSLT_CONT'])
```

```
학습 데이터: (419302, 133)
테스트 데이터: (179701, 133)
학습 레이블: (419302,)
테스트 레이블: (179701,)
```

```
import pandas as pd

# CSV 파일로부터 데이터 불러오기
X_train9 = pd.read_csv('C:/Users/Hesong/test9/X_train9.csv')
X_test9 = pd.read_csv('C:/Users/Hesong/test9/X_test9.csv')
y_train9 = pd.read_csv('C:/Users/Hesong/test9/y_train9.csv')['RSLT_CONT']
y_test9 = pd.read_csv('C:/Users/Hesong/test9/y_test9.csv')['RSLT_CONT']
```

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import roc_auc_score, accuracy_score, precision_score, recall_score, f1_score

# 모델 생성
model = Sequential()

# 입력층과 첫 번째 은닉층
model.add(Dense(units=8, activation='relu', input_dim=X_train9.shape[1]))

# 두 번째 은닉층
model.add(Dense(units=4, activation='relu'))

# 출력층
model.add(Dense(units=1, activation='sigmoid'))

# 모델 컴파일
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# 모델 요약 정보 출력
model.summary()

# 모델 훈련
model.fit(X_train9, y_train9, epochs=30, batch_size=32, validation_data=(X_test9, y_test9))
```

```
# 모델 평가
loss, accuracy = model.evaluate(X_test9, y_test9)

# 모델 예측
y_pred = model.predict(X_test9)
y_pred_binary = (y_pred > 0.5).astype(int)  # 확률을 이진 클래스로 변환

# 결과 출력
print(y_pred)

# AUROC 계산
roc_auc = roc_auc_score(y_test9, y_pred)

# 평가 지표 계산
accuracy = accuracy_score(y_test9, y_pred_binary)
precision = precision_score(y_test9, y_pred_binary, average='weighted')
recall = recall_score(y_test9, y_pred_binary, average='weighted')
f1 = f1_score(y_test9, y_pred_binary, average='weighted')

# 결과 출력
print("AUROC:", roc_auc)
print("accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 8)                 1072

 dense_7 (Dense)             (None, 4)                 36

 dense_8 (Dense)             (None, 1)                 5

=================================================================
Total params: 1113 (4.35 KB)
Trainable params: 1113 (4.35 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/30
13104/13104 [==============================] - 35s 3ms/step - loss: 0.9396 - accuracy: 0.6701 - val_loss: 0.6132 - val_accuracy: 0.6727
Epoch 2/30
13104/13104 [==============================] - 33s 3ms/step - loss: 0.5590 - accuracy: 0.7224 - val_loss: 0.5698 - val_accuracy: 0.7024
Epoch 3/30
13104/13104 [==============================] - 33s 3ms/step - loss: 0.5463 - accuracy: 0.7296 - val_loss: 0.5264 - val_accuracy: 0.7412
Epoch 4/30
13104/13104 [==============================] - 33s 2ms/step - loss: 0.5429 - accuracy: 0.7301 - val_loss: 0.5352 - val_accuracy: 0.7315
Epoch 5/30
13104/13104 [==============================] - 31s 2ms/step - loss: 0.5420 - accuracy: 0.7300 - val_loss: 0.5286 - val_accuracy: 0.7398
Epoch 6/30
13104/13104 [==============================] - 32s 2ms/step - loss: 0.5390 - accuracy: 0.7313 - val_loss: 0.5405 - val_accuracy: 0.7272
Epoch 7/30
13104/13104 [==============================] - 32s 2ms/step - loss: 0.5378 - accuracy: 0.7315 - val_loss: 0.5477 - val_accuracy: 0.7292
Epoch 8/30
13104/13104 [==============================] - 32s 2ms/step - loss: 0.5355 - accuracy: 0.7330 - val_loss: 0.5353 - val_accuracy: 0.7310
Epoch 9/30
13104/13104 [==============================] - 31s 2ms/step - loss: 0.5362 - accuracy: 0.7324 - val_loss: 0.5453 - val_accuracy: 0.7298
Epoch 10/30
13104/13104 [==============================] - 33s 3ms/step - loss: 0.5347 - accuracy: 0.7330 - val_loss: 0.5356 - val_accuracy: 0.7366
Epoch 11/30
13104/13104 [==============================] - 33s 2ms/step - loss: 0.5328 - accuracy: 0.7342 - val_loss: 0.5525 - val_accuracy: 0.7114
Epoch 12/30
13104/13104 [==============================] - 32s 2ms/step - loss: 0.5331 - accuracy: 0.7336 - val_loss: 0.5224 - val_accuracy: 0.7414
Epoch 13/30
13104/13104 [==============================] - 35s 3ms/step - loss: 0.5324 - accuracy: 0.7343 - val_loss: 0.5206 - val_accuracy: 0.7428
Epoch 14/30
13104/13104 [==============================] - 35s 3ms/step - loss: 0.5317 - accuracy: 0.7344 - val_loss: 0.5301 - val_accuracy: 0.7378
Epoch 15/30
13104/13104 [==============================] - 34s 3ms/step - loss: 0.5301 - accuracy: 0.7355 - val_loss: 0.5293 - val_accuracy: 0.7386
Epoch 16/30
13104/13104 [==============================] - 36s 3ms/step - loss: 0.5307 - accuracy: 0.7349 - val_loss: 0.5528 - val_accuracy: 0.7295
Epoch 17/30
13104/13104 [==============================] - 34s 3ms/step - loss: 0.5304 - accuracy: 0.7351 - val_loss: 0.5190 - val_accuracy: 0.7432
Epoch 18/30
13104/13104 [==============================] - 35s 3ms/step - loss: 0.5296 - accuracy: 0.7353 - val_loss: 0.5235 - val_accuracy: 0.7371
Epoch 19/30
13104/13104 [==============================] - 36s 3ms/step - loss: 0.5296 - accuracy: 0.7355 - val_loss: 0.5408 - val_accuracy: 0.7355
Epoch 20/30
13104/13104 [==============================] - 35s 3ms/step - loss: 0.5298 - accuracy: 0.7359 - val_loss: 0.5382 - val_accuracy: 0.7345
Epoch 21/30
13104/13104 [==============================] - 35s 3ms/step - loss: 0.5291 - accuracy: 0.7356 - val_loss: 0.5277 - val_accuracy: 0.7340
Epoch 22/30
```

TEST10

```
import pandas as pd
from sklearn.model_selection import train_test_split

# CSV 파일에서 데이터 불러오기
data10 = pd.read_csv('C:/Users/Hesong/test10/train_data_10.csv')

# 'DAY'는 수치형 데이터이므로 따로 선택
numeric_feature = ['TPRSC_CAPA', 'DAY']
X_numeric = data10[numeric_feature]

# 'INGR_NM'을 원핫인코딩하여 선택
categorical_features = ['INGR_NM', 'ANTBT_NM', 'antibiotics']
X_categorical = pd.get_dummies(data10[categorical_features], columns=categorical_features)

# 두 데이터프레임을 합쳐 입력 데이터 생성
X = pd.concat([X_categorical, X_numeric], axis=1)

# "RSLT_CONT" 값을 숫자로 변환
y = data10['RSLT_CONT'].map({'R': 1, 'S': 0})

# 학습 데이터와 테스트 데이터 분할
X_train10, X_test10, y_train10, y_test10 = train_test_split(X, y, test_size=0.3, stratify=y, random_state=42)

# 나눈 데이터의 크기 출력
print("학습 데이터:", X_train10.shape)
print("테스트 데이터:", X_test10.shape)
print("학습 레이블:", y_train10.shape)
print("테스트 레이블:", y_test10.shape)

# 학습 데이터와 테스트 데이터를 CSV 파일로 저장
X_train10.to_csv('C:/Users/Hesong/test10/X_train10.csv', index=False)
X_test10.to_csv('C:/Users/Hesong/test10/X_test10.csv', index=False)
y_train10.to_csv('C:/Users/Hesong/test10/y_train10.csv', index=False, header=['RSLT_CONT'])
y_test10.to_csv('C:/Users/Hesong/test10/y_test10.csv', index=False, header=['RSLT_CONT'])
```

```
학습 데이터: (419302, 133)
테스트 데이터: (179701, 133)
학습 레이블: (419302,)
테스트 레이블: (179701,)
```

```
import pandas as pd

# CSV 파일로부터 데이터 불러오기
X_train10 = pd.read_csv('C:/Users/Hesong/test10/X_train10.csv')
X_test10 = pd.read_csv('C:/Users/Hesong/test10/X_test10.csv')
y_train10 = pd.read_csv('C:/Users/Hesong/test10/y_train10.csv')['RSLT_CONT']
y_test10 = pd.read_csv('C:/Users/Hesong/test10/y_test10.csv')['RSLT_CONT']
```

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import roc_auc_score, accuracy_score, precision_score, recall_score, f1_score

# 모델 생성
model = Sequential()

# 입력층과 첫 번째 은닉층
model.add(Dense(units=8, activation='relu', input_dim=X_train10.shape[1]))

# 두 번째 은닉층
model.add(Dense(units=4, activation='relu'))

# 출력층
model.add(Dense(units=1, activation='sigmoid'))

# 모델 컴파일
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# 모델 요약 정보 출력
model.summary()

# 모델 훈련
model.fit(X_train10, y_train10, epochs=30, batch_size=32, validation_data=(X_test10, y_test10))
```

```
# 모델 평가
loss, accuracy = model.evaluate(X_test10, y_test10)

# 모델 예측
y_pred = model.predict(X_test10)
y_pred_binary = (y_pred > 0.5).astype(int)  # 확률을 이진 클래스로 변환

# 결과 출력
print(y_pred)

# AUROC 계산
roc_auc = roc_auc_score(y_test10, y_pred)

# 평가 지표 계산
accuracy = accuracy_score(y_test10, y_pred_binary)
precision = precision_score(y_test10, y_pred_binary, average='weighted')
recall = recall_score(y_test10, y_pred_binary, average='weighted')
f1 = f1_score(y_test10, y_pred_binary, average='weighted')

# 결과 출력
print("AUROC:", roc_auc)
print("accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_9 (Dense)             (None, 8)                 1072

 dense_10 (Dense)            (None, 4)                 36

 dense_11 (Dense)            (None, 1)                 5

=================================================================
Total params: 1113 (4.35 KB)
Trainable params: 1113 (4.35 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/30
13104/13104 [==============================] - 38s 3ms/step - loss: 0.9940 - accuracy: 0.6888 - val_loss: 0.7015 - val_accuracy: 0.6621
Epoch 2/30
13104/13104 [==============================] - 35s 3ms/step - loss: 0.9311 - accuracy: 0.6978 - val_loss: 0.5428 - val_accuracy: 0.7389
Epoch 3/30
13104/13104 [==============================] - 35s 3ms/step - loss: 0.8324 - accuracy: 0.7039 - val_loss: 1.2761 - val_accuracy: 0.5983
Epoch 4/30
13104/13104 [==============================] - 36s 3ms/step - loss: 0.8078 - accuracy: 0.7052 - val_loss: 0.5863 - val_accuracy: 0.6926
Epoch 5/30
13104/13104 [==============================] - 35s 3ms/step - loss: 0.7658 - accuracy: 0.7076 - val_loss: 1.0063 - val_accuracy: 0.7188
Epoch 6/30
13104/13104 [==============================] - 35s 3ms/step - loss: 0.7246 - accuracy: 0.7102 - val_loss: 0.6311 - val_accuracy: 0.7312
Epoch 7/30
13104/13104 [==============================] - 36s 3ms/step - loss: 0.6796 - accuracy: 0.7138 - val_loss: 0.8103 - val_accuracy: 0.6502
Epoch 8/30
13104/13104 [==============================] - 34s 3ms/step - loss: 0.6400 - accuracy: 0.7168 - val_loss: 0.5717 - val_accuracy: 0.7323
Epoch 9/30
13104/13104 [==============================] - 33s 2ms/step - loss: 0.5987 - accuracy: 0.7204 - val_loss: 0.7054 - val_accuracy: 0.7234
Epoch 10/30
13104/13104 [==============================] - 32s 2ms/step - loss: 0.5779 - accuracy: 0.7242 - val_loss: 0.5218 - val_accuracy: 0.7418
Epoch 11/30
13104/13104 [==============================] - 32s 2ms/step - loss: 0.5596 - accuracy: 0.7266 - val_loss: 0.5257 - val_accuracy: 0.7384
Epoch 12/30
13104/13104 [==============================] - 32s 2ms/step - loss: 0.5483 - accuracy: 0.7293 - val_loss: 0.5278 - val_accuracy: 0.7379
Epoch 13/30
13104/13104 [==============================] - 33s 2ms/step - loss: 0.5432 - accuracy: 0.7302 - val_loss: 0.5280 - val_accuracy: 0.7368
Epoch 14/30
13104/13104 [==============================] - 33s 2ms/step - loss: 0.5393 - accuracy: 0.7322 - val_loss: 0.5304 - val_accuracy: 0.7372
Epoch 15/30
13104/13104 [==============================] - 33s 2ms/step - loss: 0.5363 - accuracy: 0.7329 - val_loss: 0.5401 - val_accuracy: 0.7310
Epoch 16/30
13104/13104 [==============================] - 33s 2ms/step - loss: 0.5337 - accuracy: 0.7345 - val_loss: 0.5316 - val_accuracy: 0.7353
Epoch 17/30
13104/13104 [==============================] - 32s 2ms/step - loss: 0.5326 - accuracy: 0.7343 - val_loss: 0.5238 - val_accuracy: 0.7411
Epoch 18/30
13104/13104 [==============================] - 32s 2ms/step - loss: 0.5325 - accuracy: 0.7345 - val_loss: 0.5268 - val_accuracy: 0.7380
Epoch 19/30
13104/13104 [==============================] - 33s 2ms/step - loss: 0.5310 - accuracy: 0.7345 - val_loss: 0.5248 - val_accuracy: 0.7391
Epoch 20/30
13104/13104 [==============================] - 32s 2ms/step - loss: 0.5299 - accuracy: 0.7353 - val_loss: 0.5209 - val_accuracy: 0.7416
Epoch 21/30
13104/13104 [==============================] - 33s 2ms/step - loss: 0.5292 - accuracy: 0.7360 - val_loss: 0.5605 - val_accuracy: 0.7076
Epoch 22/30
```