와인 품질 예측하기

1.환경준비

(1) 라이브러리 로딩

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.metrics import *
from sklearn.preprocessing import MinMaxScaler

from keras.models import Sequential
from keras.layers import Dense
from keras.backend import clear_session
from keras.optimizers import Adam
from keras.utils import to_categorical
```

- 함수 만들기

```
# 학습곡선 함수
def dl_history_plot(history):
    plt.figure(figsize=(10,6))
    plt.plot(history['loss'], label='train_err', marker = '.')
    plt.plot(history['val_loss'], label='val_err', marker = '.')

    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend()
    plt.grid()
    plt.show()
```

(2) 데이터로딩

```
path = "https://raw.githubusercontent.com/DA4BAM/dataset/master/winequality-white.csv"
data = pd.read_csv(path)
data['quality'] = np.where(data['quality'] == 3, 4, np.where(data['quality'] == 9, 8, data['quality']))
data['quality'] = data['quality'] - 4
data.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40

Next steps: Generate code with data View recommended plots

- 범주가 5개 입니다.
  - 0 - 최하 ~ 4 - 최상

```
data['quality'].value_counts()
```

```
quality
2    2198
1    1457
3     880
0     183
4     180
Name: count, dtype: int64
```

2.데이터 준비

(1) 데이터 준비

- y에 대한 전처리 : 위에서 이미 0 ~ 4로 범주를 맞췄습니다.
- x, y 나누기

```
target = 'quality'
x = data.drop(target, axis = 1)
y = data.loc[:, target]
```

(2) 데이터 분할

```
x_train, x_val, y_train, y_val = train_test_split(x, y, test_size= .2, random_state = 2024)
```

(3) 스케일링

```
scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_val = scaler.transform(x_val)
```

3.모델링

최소 3개 이상의 모델을 생성하고 성능을 비교하시오.

```
n = x_train.shape[1] #num of columns
n
```

```
11
```

(1) 모델1

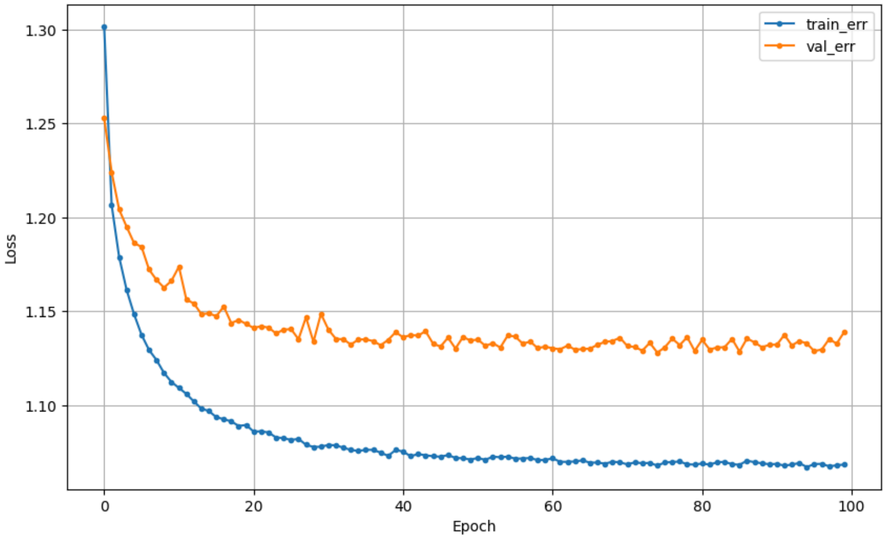
```
clear_session()
model1 = Sequential(Dense(5, input_shape=(n,) ,activation='softmax'))
model1.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 5)                 60        
                                                                 
=================================================================
Total params: 60 (240.00 Byte)
Trainable params: 60 (240.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model1.compile(optimizer=Adam(0.01), loss='sparse_categorical_crossentropy')
hist = model1.fit(x_train, y_train, epochs=100, validation_split=.2).history
```

```
Epoch 72/100
98/98 [==============================] - 0s 3ms/step - loss: 1.0696 - val_loss: 1.1309
Epoch 73/100
98/98 [==============================] - 0s 3ms/step - loss: 1.0691 - val_loss: 1.1291
Epoch 74/100
98/98 [==============================] - 0s 3ms/step - loss: 1.0693 - val_loss: 1.1332
Epoch 75/100
98/98 [==============================] - 0s 3ms/step - loss: 1.0679 - val_loss: 1.1280
Epoch 76/100
98/98 [==============================] - 0s 3ms/step - loss: 1.0695 - val_loss: 1.1308
Epoch 77/100
98/98 [==============================] - 0s 3ms/step - loss: 1.0697 - val_loss: 1.1355
Epoch 78/100
98/98 [==============================] - 0s 2ms/step - loss: 1.0700 - val_loss: 1.1320
Epoch 79/100
98/98 [==============================] - 0s 4ms/step - loss: 1.0685 - val_loss: 1.1362
Epoch 80/100
98/98 [==============================] - 0s 3ms/step - loss: 1.0684 - val_loss: 1.1289
Epoch 81/100
98/98 [==============================] - 0s 2ms/step - loss: 1.0690 - val_loss: 1.1348
Epoch 82/100
98/98 [==============================] - 0s 2ms/step - loss: 1.0683 - val_loss: 1.1297
Epoch 83/100
98/98 [==============================] - 0s 2ms/step - loss: 1.0696 - val_loss: 1.1307
Epoch 84/100
98/98 [==============================] - 0s 2ms/step - loss: 1.0698 - val_loss: 1.1309
Epoch 85/100
98/98 [==============================] - 0s 2ms/step - loss: 1.0686 - val_loss: 1.1353
Epoch 86/100
98/98 [==============================] - 0s 2ms/step - loss: 1.0683 - val_loss: 1.1285
Epoch 87/100
98/98 [==============================] - 0s 3ms/step - loss: 1.0704 - val_loss: 1.1355
Epoch 88/100
98/98 [==============================] - 0s 2ms/step - loss: 1.0697 - val_loss: 1.1335
Epoch 89/100
98/98 [==============================] - 0s 3ms/step - loss: 1.0691 - val_loss: 1.1308
Epoch 90/100
98/98 [==============================] - 0s 2ms/step - loss: 1.0686 - val_loss: 1.1322
Epoch 91/100
98/98 [==============================] - 0s 3ms/step - loss: 1.0688 - val_loss: 1.1322
Epoch 92/100
98/98 [==============================] - 0s 4ms/step - loss: 1.0678 - val_loss: 1.1374
Epoch 93/100
98/98 [==============================] - 0s 3ms/step - loss: 1.0684 - val_loss: 1.1318
Epoch 94/100
98/98 [==============================] - 0s 3ms/step - loss: 1.0692 - val_loss: 1.1342
Epoch 95/100
98/98 [==============================] - 0s 3ms/step - loss: 1.0671 - val_loss: 1.1329
Epoch 96/100
98/98 [==============================] - 0s 3ms/step - loss: 1.0687 - val_loss: 1.1288
Epoch 97/100
98/98 [==============================] - 0s 4ms/step - loss: 1.0688 - val_loss: 1.1297
Epoch 98/100
98/98 [==============================] - 0s 4ms/step - loss: 1.0674 - val_loss: 1.1352
Epoch 99/100
98/98 [==============================] - 0s 5ms/step - loss: 1.0680 - val_loss: 1.1328
Epoch 100/100
98/98 [==============================] - 0s 3ms/step - loss: 1.0684 - val_loss: 1.1389
```

```
dl_history_plot(hist)
```



```
pred = model1.predict(x_val)
pred = np.argmax(pred, axis=1)
```

```
31/31 [==============================] - 0s 2ms/step
```

```
print(confusion_matrix(y_val, pred))
print(classification_report(y_val, pred))
```

```
[[  2  20  11   0   0]
 [  1 164 126   8   0]
 [  2  90 297  46   0]
 [  0   0 121  55   1]
 [  0   0  16  20   0]]
              precision    recall  f1-score   support

           0       0.40      0.06      0.11        33
           1       0.60      0.55      0.57       299
           2       0.52      0.68      0.59       435
           3       0.43      0.31      0.36       177
           4       0.00      0.00      0.00        36

    accuracy                           0.53       980
   macro avg       0.39      0.32      0.33       980
weighted avg       0.50      0.53      0.51       980
```

코딩을 시작하거나 AI로 코드를 생성하세요.

(2) 모델2

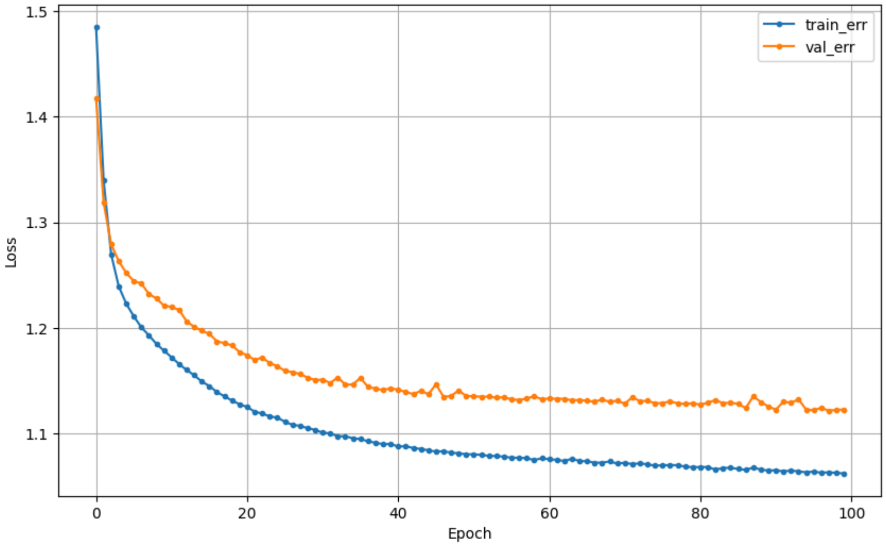
```
clear_session()
model2 = Sequential([Dense(16, input_shape=(n,) ,activation='relu'),
                     Dense(5, activation='relu'),
                     Dense(5, activation='softmax')])
model2.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 16)                192       
                                                                 
 dense_1 (Dense)             (None, 5)                 85        
 dense_2 (Dense)             (None, 5)                 30

=================================================================
Total params: 307 (1.20 KB)
Trainable params: 307 (1.20 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model2.compile(optimizer=Adam(0.001), loss='sparse_categorical_crossentropy')
hist = model2.fit(x_train, y_train, epochs=100, validation_split=.2).history
```

```
Epoch 72/100
98/98 [==============================] - 0s 3ms/step - loss: 1.0709 - val_loss: 1.1344
Epoch 73/100
98/98 [==============================] - 0s 3ms/step - loss: 1.0717 - val_loss: 1.1305
Epoch 74/100
98/98 [==============================] - 0s 2ms/step - loss: 1.0706 - val_loss: 1.1308
Epoch 75/100
98/98 [==============================] - 0s 3ms/step - loss: 1.0695 - val_loss: 1.1286
Epoch 76/100
98/98 [==============================] - 0s 3ms/step - loss: 1.0698 - val_loss: 1.1287
Epoch 77/100
98/98 [==============================] - 0s 3ms/step - loss: 1.0700 - val_loss: 1.1305
Epoch 78/100
98/98 [==============================] - 0s 3ms/step - loss: 1.0699 - val_loss: 1.1284
Epoch 79/100
98/98 [==============================] - 0s 3ms/step - loss: 1.0685 - val_loss: 1.1279
Epoch 80/100
98/98 [==============================] - 0s 3ms/step - loss: 1.0680 - val_loss: 1.1286
Epoch 81/100
98/98 [==============================] - 0s 3ms/step - loss: 1.0681 - val_loss: 1.1272
Epoch 82/100
98/98 [==============================] - 0s 3ms/step - loss: 1.0681 - val_loss: 1.1291
Epoch 83/100
98/98 [==============================] - 0s 3ms/step - loss: 1.0657 - val_loss: 1.1316
Epoch 84/100
98/98 [==============================] - 0s 3ms/step - loss: 1.0672 - val_loss: 1.1286
Epoch 85/100
98/98 [==============================] - 0s 3ms/step - loss: 1.0675 - val_loss: 1.1290
Epoch 86/100
98/98 [==============================] - 0s 3ms/step - loss: 1.0663 - val_loss: 1.1281
Epoch 87/100
98/98 [==============================] - 0s 3ms/step - loss: 1.0654 - val_loss: 1.1239
Epoch 88/100
98/98 [==============================] - 0s 2ms/step - loss: 1.0676 - val_loss: 1.1352
Epoch 89/100
98/98 [==============================] - 0s 4ms/step - loss: 1.0657 - val_loss: 1.1298
Epoch 90/100
98/98 [==============================] - 0s 4ms/step - loss: 1.0648 - val_loss: 1.1255
Epoch 91/100
98/98 [==============================] - 0s 4ms/step - loss: 1.0652 - val_loss: 1.1225
Epoch 92/100
98/98 [==============================] - 0s 4ms/step - loss: 1.0640 - val_loss: 1.1302
Epoch 93/100
98/98 [==============================] - 0s 4ms/step - loss: 1.0649 - val_loss: 1.1289
Epoch 94/100
98/98 [==============================] - 0s 5ms/step - loss: 1.0640 - val_loss: 1.1323
Epoch 95/100
98/98 [==============================] - 0s 5ms/step - loss: 1.0631 - val_loss: 1.1225
Epoch 96/100
98/98 [==============================] - 0s 3ms/step - loss: 1.0638 - val_loss: 1.1220
Epoch 97/100
98/98 [==============================] - 0s 3ms/step - loss: 1.0628 - val_loss: 1.1243
Epoch 98/100
98/98 [==============================] - 0s 2ms/step - loss: 1.0632 - val_loss: 1.1214
Epoch 99/100
98/98 [==============================] - 0s 2ms/step - loss: 1.0626 - val_loss: 1.1223
Epoch 100/100
98/98 [==============================] - 0s 3ms/step - loss: 1.0620 - val_loss: 1.1226
```

```
dl_history_plot(hist)
```



```
pred = model2.predict(x_val)
pred = np.argmax(pred, axis=1)
```

```
31/31 [==============================] - 0s 1ms/step
```

```
print(confusion_matrix(y_val, pred))
print(classification_report(y_val, pred))
```

```
[[  0  24   9   0   0]
 [  0 162 134   3   0]
 [  0  89 326  20   0]
 [  0   6 134  37   0]
 [  0   0  26  10   0]]
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        33
           1       0.58      0.54      0.56       299
           2       0.52      0.75      0.61       435
           3       0.53      0.21      0.30       177
           4       0.00      0.00      0.00        36

    accuracy                           0.54       980
   macro avg       0.32      0.30      0.29       980
weighted avg       0.50      0.54      0.50       980

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and be
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and be
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and be
  _warn_prf(average, modifier, msg_start, len(result))
```

(3) 모델3

```
y_ = to_categorical(y.values, 5)
```

```
x_train, x_val, y_train, y_val = train_test_split(x, y_, test_size = .2, random_state = 2024)
```

```
scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_val = scaler.transform(x_val)
```

```
n = x_train.shape[1] #num of columns
n
```

```
     11
```

```
# 메모리 정리
clear_session()

# Sequential
model3 = Sequential([Dense(10, input_shape=(n,) ,activation='relu'),
                     Dense(8, activation='relu'),
                     Dense(8, activation='relu'),
                     Dense(5, activation='softmax')])

# 모델요약
model3.summary()
```

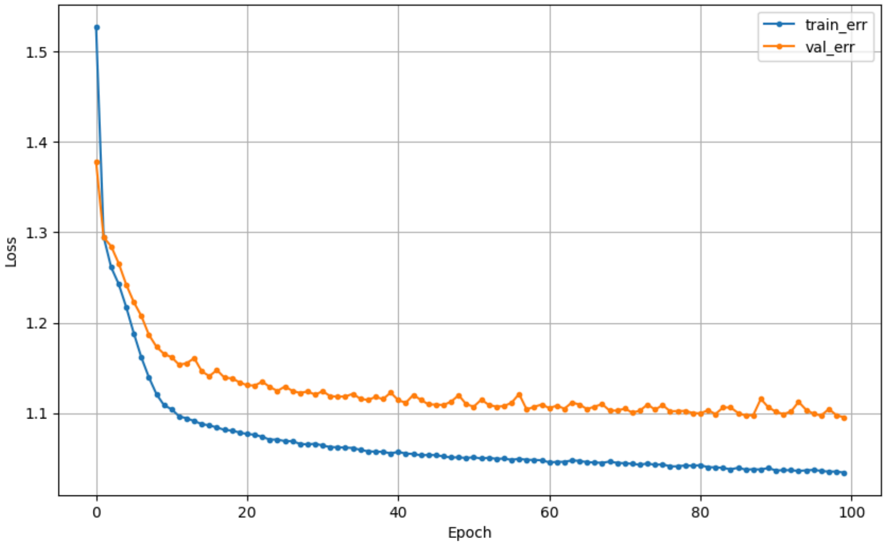
```
     Model: "sequential"
     _________________________________________________________________
      Layer (type)                Output Shape              Param #
     =================================================================
      dense (Dense)               (None, 10)                120

      dense_1 (Dense)             (None, 8)                 88

      dense_2 (Dense)             (None, 8)                 72

      dense_3 (Dense)             (None, 5)                 45

     =================================================================
     Total params: 325 (1.27 KB)
     Trainable params: 325 (1.27 KB)
     Non-trainable params: 0 (0.00 Byte)
     _________________________________________________________________
```

```
model3.compile(optimizer=Adam(0.001), loss='categorical_crossentropy')
```

```
hist = model3.fit(x_train, y_train, epochs=100, validation_split=0.2).history
```

```
98/98 [==============================] - 1s 6ms/step - loss: 1.0369 - val_loss: 1.0989
Epoch 93/100
98/98 [==============================] - 0s 4ms/step - loss: 1.0368 - val_loss: 1.1018
Epoch 94/100
98/98 [==============================] - 1s 5ms/step - loss: 1.0360 - val_loss: 1.1125
Epoch 95/100
98/98 [==============================] - 0s 5ms/step - loss: 1.0367 - val_loss: 1.1036
Epoch 96/100
98/98 [==============================] - 1s 5ms/step - loss: 1.0372 - val_loss: 1.0992
Epoch 97/100
98/98 [==============================] - 0s 4ms/step - loss: 1.0361 - val_loss: 1.0973
Epoch 98/100
98/98 [==============================] - 0s 4ms/step - loss: 1.0352 - val_loss: 1.1045
Epoch 99/100
98/98 [==============================] - 0s 3ms/step - loss: 1.0354 - val_loss: 1.0978
Epoch 100/100
98/98 [==============================] - 0s 3ms/step - loss: 1.0341 - val_loss: 1.0953
```

```
dl_history_plot(hist)
```



```
pred = model3.predict(x_val)
pred = pred.argmax(axis=1)
```

```
31/31 [==============================] - 0s 2ms/step
```

```
y_val = y_val.argmax(axis=1)
```

```
print(confusion_matrix(y_val, pred)) # 다중 분류에서 이 부분 잘 살펴야 함
print(classification_report(y_val, pred))
```

```
[[  1  23   7   2   0]
 [  1 177 113   8   0]
 [  0  92 290  53   0]
 [  0   1 111  65   0]
 [  0   0  19  17   0]]
              precision    recall  f1-score   support

           0       0.50      0.03      0.06        33
           1       0.60      0.59      0.60       299
           2       0.54      0.67      0.59       435
           3       0.45      0.37      0.40       177
           4       0.00      0.00      0.00        36

    accuracy                           0.54       980
   macro avg       0.42      0.33      0.33       980
weighted avg       0.52      0.54      0.52       980

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and be
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and be
```

```
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and be
  _warn_prf(average, modifier, msg_start, len(result))
```

코딩을 시작하거나 AI로 코드를 생성하세요.