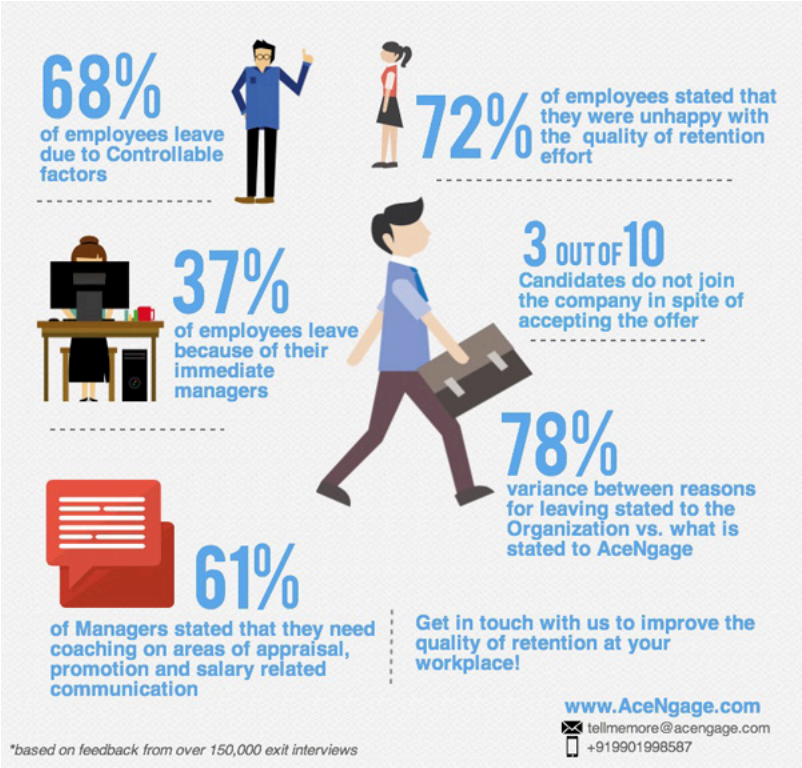
이진분류 실습 : 이직 예측

- 회사 인사팀에서는 여러분들에게 직원의 이직여부과 관련해서 분석을 요청하였습니다.
- 최근 이직율이 증가하는 것에 대해 우려를 갖고 있기에, 이직여부에 영향을 주는 요인에 대해 분석하여, 이직할 것으로 보이는 직원들에 대해 회사를 떠나지 않도록 인사 프로그램을 준비하려고 합니다.
- 어떤 직원이 이직할지 예측해 봅시다.



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
```

- 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) 데이터 로딩

```
# data data
path = "https://raw.githubusercontent.com/DA4BAM/dataset/master/Attrition_train_validation.CSV"
data = pd.read_csv(path)
data['Attrition'] = np.where(data['Attrition']=='Yes', 1, 0)
data.head(10)
```

	Attrition	Age	BusinessTravel	Department	DistanceFromHome	Education	EducationField
0	0	33	Travel_Rarely	Research & Development	7	3	Medical
1	0	35	Travel_Frequently	Research & Development	18	2	Life Sciences
2	0	42	Travel_Rarely	Research & Development	6	3	Medical
3	0	46	Travel_Rarely	Sales	2	3	Marketing
4	0	39	Travel_Frequently	Sales	20	3	Life Sciences
5	1	22	Travel_Frequently	Research & Development	4	1	Technical Degree
6	0	24	Travel_Rarely	Research & Development	21	2	Technical Degree
7	0	34	Travel_Rarely	Research & Development	8	3	Medical
8	0	30	Travel_Rarely	Research & Development	20	3	Other
9	0	26	Travel_Rarely	Research & Development	6	3	Life Sciences

10 rows × 26 columns

변수 명	내용	구분
Attrition	이직여부, Yes = 1 , No = 0	**Target**
Age	나이	숫자
BusinessTravel	출장 빈도(범주)	
Department	현 부서	
DistanceFromHome	집-직장 거리(마일)	숫자
Education	교육수준(범주)	1 Below College, 2 College, 3 Bachelor, 4 Master, 5 Doctor
EducationField	전공	
EmployeeNumber	사번	
EnvironmentSatisfaction	근무환경에 대한 만족도(범주)	1 Low, 2 Good, 3 Excellent, 4 Outstanding
Gender	성별	
JobInvolvement	직무 적극성(참여도)	1 Low, 2 Medium, 3 High, 4 Very High
JobRole	직무	
JobSatisfaction	직무 만족도	1 Low, 2 Medium, 3 High, 4 Very High
MaritalStatus	결혼상태	Single, Married, Divorced
MonthlyIncome	월급	숫자
NumCompaniesWorked	현재까지 근무한 회사 수	숫자
OverTime	야근여부	범주
PercentSalaryHike	전년대비 급여인상율(%)	숫자
RelationshipSatisfaction	동료와의 관계 만족도	1 Low, 2 Medium, 3 High, 4 Very High

변수 명	내용	구분
StockOptionLevel	스톡옵션 수준 0~3	범주
TotalWorkingYears	총 근무 연수	숫자
TrainingTimesLastYear	전년 교육훈련 횟수	숫자
WorkLifeBalance	워라밸. 일-삶 균형도	1 Bad, 2 Good, 3 Better, 4 Best
YearsAtCompany	현직장 근무 연수	숫자
YearsInCurrentRole	현직무 연수	숫자
YearsWithCurrManager	현 팀장과 근무한 연수	숫자

2.데이터 전처리

(1) 데이터 정리

```
target = 'Attrition'


# 불필요한 변수 제거
data.drop('EmployeeNumber', axis = 1, inplace = True)


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

(2) 가변수화

- 범주형 데이터이면서 값이 0,1 로 되어 있는 것이 아니라면, 가변수화를 수행해야 합니다.
- 대상이 되는 변수에 대해서 가변수화를 수행해주세요.

```
dum_cols = ['BusinessTravel','Department','Education','EducationField','EnvironmentSatisfaction','Gender',
            'JobRole', 'JobInvolvement', 'JobSatisfaction', 'MaritalStatus', 'OverTime', 'RelationshipSatisfaction',
            'StockOptionLevel','WorkLifeBalance' ]

x = pd.get_dummies(x, columns = dum_cols ,drop_first = True)
x.head()
```

	Age	DistanceFromHome	MonthlyIncome	NumCompaniesWorked	PercentSalaryHike	TotalWorkingYea
0	33	7	11691	0	11	
1	35	18	9362	2	11	
2	42	6	13348	9	13	
3	46	2	17048	8	23	
4	39	20	4127	2	18	

5 rows × 53 columns

(3) 데이터 분할

- train_test_split :
  - test_size : 0.# - 비율로 분할, 1보다 큰 자연수 - 갯수로 분할
  - train_size로 지정도 가능.

```
x_train, x_val, y_train, y_val = train_test_split(x, y, test_size = 200, random_state = 2022)
```

(4) 스케일링

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

```
y_train.value_counts() / len(y_train)
```

```
Attrition
0    0.839048
1    0.160952
Name: count, dtype: float64
```

4.모델링

(1) 모델1

- 다양한 구조의 모델 2개 이상을 설계하시오. (히든레이어, 노드 수 조절)

```
n = x_train.shape[1]
n
```

```
53
```

```
clear_session()
model1 = Sequential([Dense(32, input_shape=(n, ), activation='relu'),
                     Dense(16, activation='relu'),
                     Dense(8, activation='relu'),
                     Dense(4, activation='relu'),
                     Dense(1, activation='sigmoid')])
model1.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 32)                1728

 dense_1 (Dense)             (None, 16)                528

 dense_2 (Dense)             (None, 8)                 136

 dense_3 (Dense)             (None, 4)                 36

 dense_4 (Dense)             (None, 1)                 5

=================================================================
Total params: 2433 (9.50 KB)
Trainable params: 2433 (9.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model1.compile(optimizer=Adam(0.01), loss='binary_crossentropy')
hist = model1.fit(x_train, y_train, epochs=30, validation_split=0.2).history
```

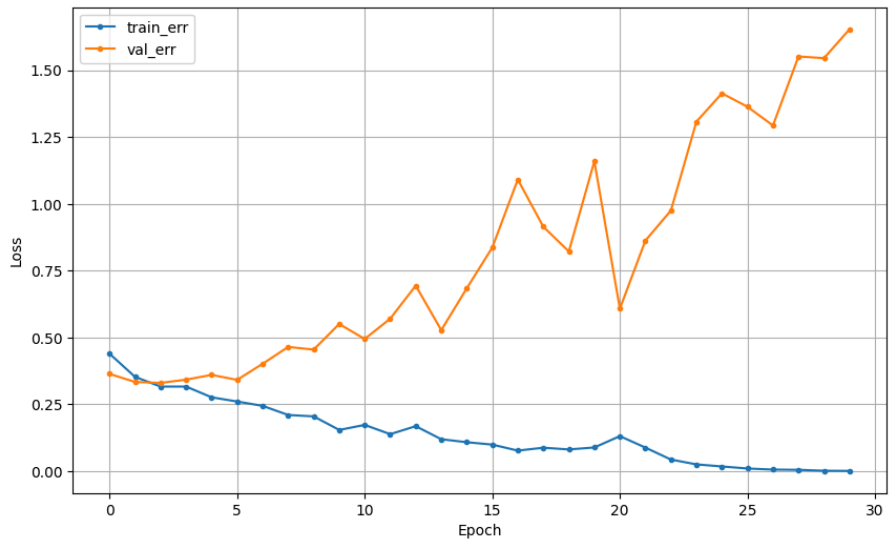
```
Epoch 2/30
27/27 [==============================] - 0s 4ms/step - loss: 0.3524 - val_loss: 0.3334
Epoch 3/30
27/27 [==============================] - 0s 5ms/step - loss: 0.3162 - val_loss: 0.3299
Epoch 4/30
27/27 [==============================] - 0s 6ms/step - loss: 0.3162 - val_loss: 0.3420
Epoch 5/30
27/27 [==============================] - 0s 5ms/step - loss: 0.2760 - val_loss: 0.3599
Epoch 6/30
27/27 [==============================] - 0s 5ms/step - loss: 0.2602 - val_loss: 0.3412
Epoch 7/30
27/27 [==============================] - 0s 6ms/step - loss: 0.2440 - val_loss: 0.4014
Epoch 8/30
27/27 [==============================] - 0s 5ms/step - loss: 0.2098 - val_loss: 0.4645
Epoch 9/30
27/27 [==============================] - 0s 5ms/step - loss: 0.2046 - val_loss: 0.4545
Epoch 10/30
27/27 [==============================] - 0s 5ms/step - loss: 0.1538 - val_loss: 0.5504
Epoch 11/30
27/27 [==============================] - 0s 5ms/step - loss: 0.1727 - val_loss: 0.4943
```

```
Epoch 13/30
27/27 [==============================] - 0s 6ms/step - loss: 0.1679 - val_loss: 0.6940
Epoch 14/30
27/27 [==============================] - 0s 6ms/step - loss: 0.1190 - val_loss: 0.5273
Epoch 15/30
27/27 [==============================] - 0s 8ms/step - loss: 0.1079 - val_loss: 0.6840
Epoch 16/30
27/27 [==============================] - 0s 8ms/step - loss: 0.0989 - val_loss: 0.8365
Epoch 17/30
27/27 [==============================] - 0s 7ms/step - loss: 0.0765 - val_loss: 1.0911
Epoch 18/30
27/27 [==============================] - 0s 10ms/step - loss: 0.0873 - val_loss: 0.9144
Epoch 19/30
27/27 [==============================] - 0s 9ms/step - loss: 0.0808 - val_loss: 0.8220
Epoch 20/30
27/27 [==============================] - 0s 7ms/step - loss: 0.0882 - val_loss: 1.1594
Epoch 21/30
27/27 [==============================] - 0s 9ms/step - loss: 0.1307 - val_loss: 0.6077
Epoch 22/30
27/27 [==============================] - 0s 9ms/step - loss: 0.0877 - val_loss: 0.8625
Epoch 23/30
27/27 [==============================] - 0s 9ms/step - loss: 0.0430 - val_loss: 0.9751
Epoch 24/30
27/27 [==============================] - 0s 8ms/step - loss: 0.0249 - val_loss: 1.3077
Epoch 25/30
27/27 [==============================] - 0s 10ms/step - loss: 0.0171 - val_loss: 1.4137
Epoch 26/30
27/27 [==============================] - 0s 9ms/step - loss: 0.0095 - val_loss: 1.3636
Epoch 27/30
27/27 [==============================] - 0s 6ms/step - loss: 0.0055 - val_loss: 1.2933
Epoch 28/30
27/27 [==============================] - 0s 5ms/step - loss: 0.0046 - val_loss: 1.5517
Epoch 29/30
27/27 [==============================] - 0s 5ms/step - loss: 0.0011 - val_loss: 1.5455
Epoch 30/30
27/27 [==============================] - 0s 5ms/step - loss: 5.0524e-04 - val_loss: 1.6535
```

```
pred = model1.predict(x_val)
pred = np.where(pred >= 0.55, 1 , 0)
```

```
7/7 [==============================] - 0s 3ms/step
```

```
dl_history_plot(hist)
```




```
# 과적합 또는 노드가 많아 그래프가 위와 같이 나온 것같음
```

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

```
[[156  13]
 [ 16  15]]
              precision    recall  f1-score   support

           0       0.91      0.92      0.91       169
           1       0.54      0.48      0.51        31

    accuracy                           0.85       200
   macro avg       0.72      0.70      0.71       200
weighted avg       0.85      0.85      0.85       200
```

(2) 모델2

- 다양한 구조를 설계하시오. (히든레이어, 노드 수 조절)

```
clear_session()
model2 = Sequential([Dense(16, input_shape=(n, ), activation='relu'),
                     Dense(8, activation='relu'),
                     Dense(1, activation='sigmoid')])
model2.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 16)                864

 dense_1 (Dense)             (None, 8)                 136

 dense_2 (Dense)             (None, 1)                 9

=================================================================
Total params: 1009 (3.94 KB)
Trainable params: 1009 (3.94 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

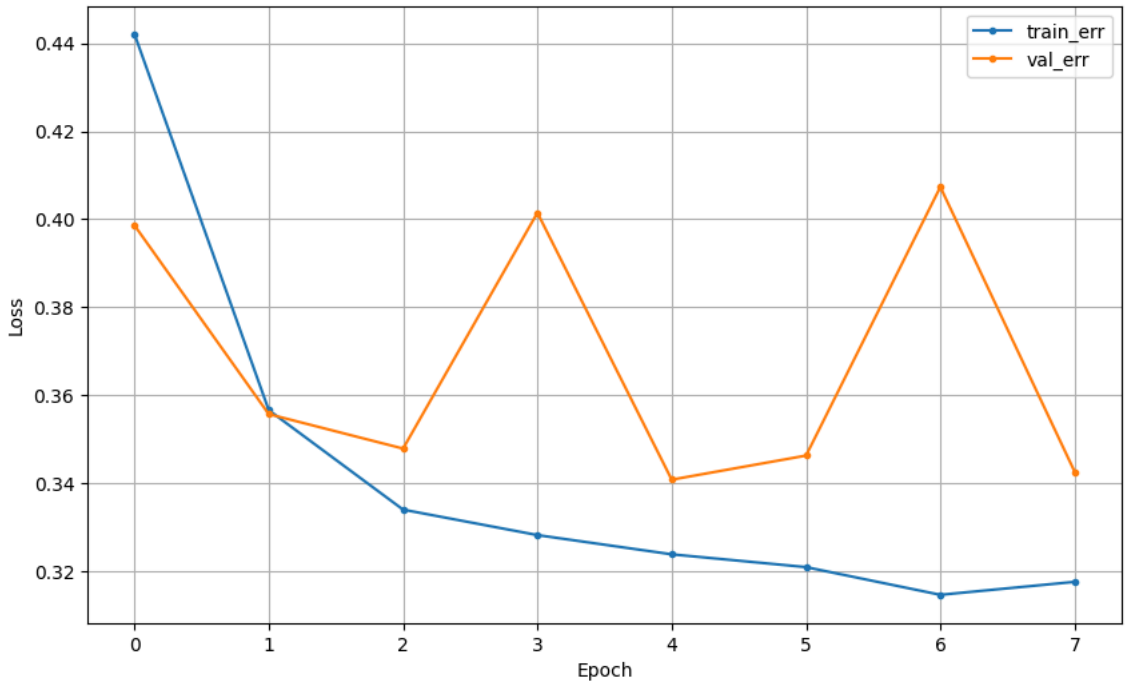
```
model2.compile(optimizer=Adam(0.1), loss='binary_crossentropy')
hist = model2.fit(x_train, y_train, epochs=8, validation_split=0.2).history
```

```
Epoch 1/8
27/27 [==============================] - 1s 11ms/step - loss: 0.4421 - val_loss: 0.3986
Epoch 2/8
27/27 [==============================] - 0s 4ms/step - loss: 0.3566 - val_loss: 0.3557
Epoch 3/8
27/27 [==============================] - 0s 4ms/step - loss: 0.3340 - val_loss: 0.3479
Epoch 4/8
27/27 [==============================] - 0s 5ms/step - loss: 0.3282 - val_loss: 0.4015
Epoch 5/8
27/27 [==============================] - 0s 4ms/step - loss: 0.3238 - val_loss: 0.3408
Epoch 6/8
27/27 [==============================] - 0s 4ms/step - loss: 0.3209 - val_loss: 0.3463
Epoch 7/8
27/27 [==============================] - 0s 5ms/step - loss: 0.3146 - val_loss: 0.4073
Epoch 8/8
27/27 [==============================] - 0s 5ms/step - loss: 0.3176 - val_loss: 0.3426
```

```
pred = model2.predict(x_val)
pred = np.where(pred >= 0.5, 1 , 0)
```

```
7/7 [==============================] - 0s 3ms/step
```

```
dl_history_plot(hist)
```




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

```
[[167   2]
 [ 19  12]]
              precision    recall  f1-score   support

           0       0.90      0.99      0.94       169
           1       0.86      0.39      0.53        31

    accuracy                           0.90       200
   macro avg       0.88      0.69      0.74       200
weighted avg       0.89      0.90      0.88       200
```

```
# 노드 수에 따라 epochs, lr 값 잘 정하기
```