데이터베이스응용 과제

20210858 정보통계학과 이지윤

1차과제

- **그래프 데이터베이스 주제 선정 이유**

  :학교폭력 그래프 데이터베이스

  학교폭력에 연관 있는 학생들만 별도로 관리하여 관계를 명확히 파악하기 위함.

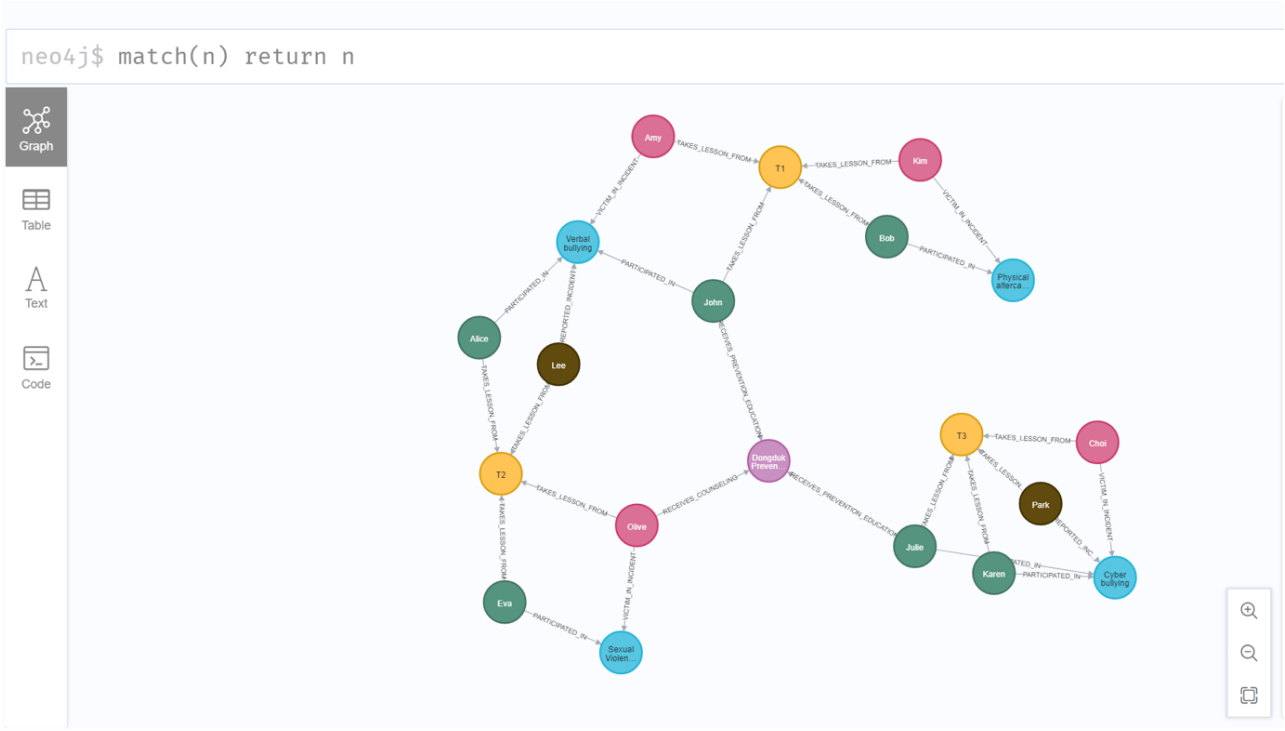
- **학교폭력 그래프 데이터베이스 소개**

  1. 학교폭력 발생 사건을 정의합니다.
  2. 발생된 학교폭력 사건과 연관된 학생들에 대하여 정의합니다.
  3. 연관된 학생들이 어느 선생님께 가르침을 받는지 정의합니다..
  4. 각 발생 사건에 대해 피해자와 가해자의 관계를 정의합니다.
  5. 학교폭력 예방센터에서 교육을 받은 학생을 정의합니다.

  총 20개의 노드를 가지며, 각 노드들이 다음의 릴레이션을 가지고 있습니다.
  아래에 설명하도록 하겠습니다.

  노드 총 20개
  가해자(Attacker_Student) – 6개
  피해자(victim_Student) – 4개
  목격자(Witness_Student) – 2개
  선생님(Teacher) – 3개
  학교 폭력 예방 센터(Prevention_School) – 1개
  학교 폭력 발생 사건(Incident) – 4개

  Relation
  피해자 -> 학교폭력사건 (VICTIM_IN_INCIDENT)
  가해자 -> 학교폭력사건 (PARTICIPATED_IN)
  목격자 -> 학교폭력사건 (REPORTED_INCIDENT)
  학생(피해자, 가해자, 목격자) -> 선생님 (TAKES_LESSON_FROM)
  일부 가해자 -> 학교폭력예방센터 (RECEIVES_PREVENTION_EDUCATION)
  일부 피해자 -> 학교폭력예방센터 (RECEIVES_COUNSELING)



Node labels

* (20)
Attacker_Student (6)
Victim_Student (4)
Witness_Student (2)
Teacher (3)
Prevention_School (1)
Incident (4)

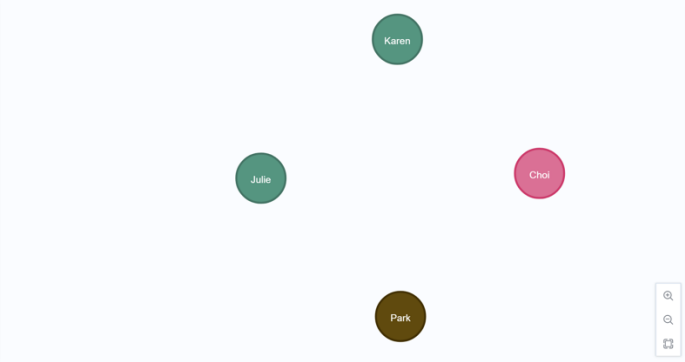
Relationship types

* (27)
TAKES_LESSON_FROM (12)
VICTIM_IN_INCIDENT (4)
PARTICIPATED_IN (6)
REPORTED_INCIDENT (2)
RECEIVES_PREVENTION_EDUCATION (2)
RECEIVES_COUNSELING (1)

- **학교폭력 데이터 베이스 질의 및 수행결과**

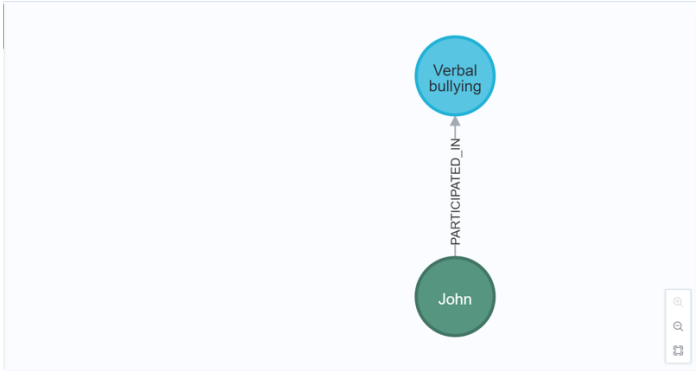
1. T3 과목의 수업을 듣는 학생 조회

```
MATCH (student)-[:TAKES_LESSON_FROM]->(teacher:Teacher {name: 'T3'})
RETURN student;
```



2. 가해자 'John'이 참여한 학교폭력 사건 조회

```
MATCH (john:Attacker_Student {name: 'John'})-[:PARTICIPATED_IN]-
>(incident:Incident)
RETURN john, incident;
```



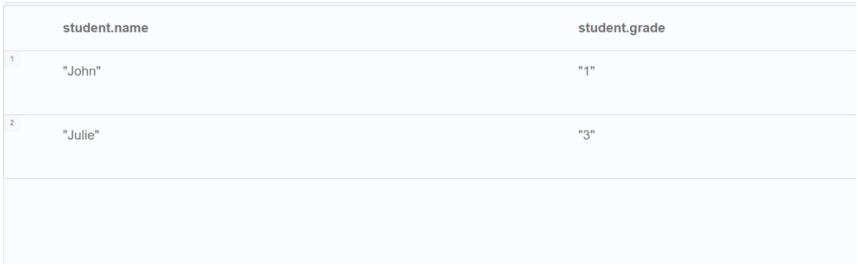
3. 피해자 'Amy'가 참여한 학교폭력 사건과 날짜 조회

```
MATCH (amy:Victim_Student {name: 'Amy'})-[:VICTIM_IN_INCIDENT]-
>(incident:Incident)
RETURN incident.description AS IncidentDescription, incident.date AS
IncidentDate;
```

	IncidentDescription	IncidentDate
1	"Verbal bullying"	"2023-04-15"

4. 예방센터에서 예방교육을 받은 학생 목록 조회

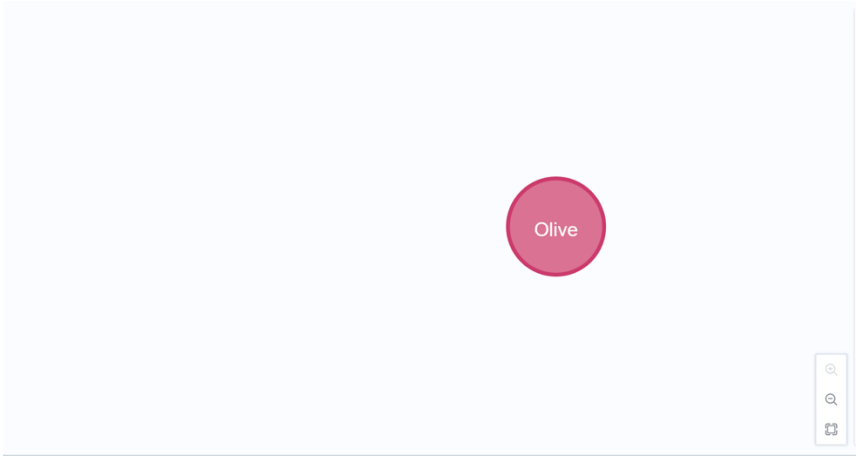
```
MATCH (student)-[:RECEIVES_PREVENTION_EDUCATION]->(:Prevention_School)
RETURN student.name, student.grade
```



	student.name	student.grade
1	"John"	"1"
2	"Julie"	"3"

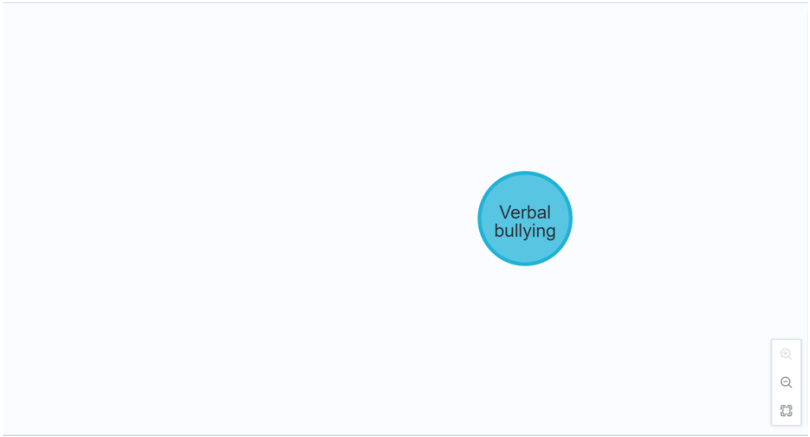
5. 학교폭력 예방센터에서 심리상담을 받는 학생 조회

```
MATCH (victim:Victim_Student)-[:RECEIVES_COUNSELING]->(:Prevention_School)
RETURN victim;
```



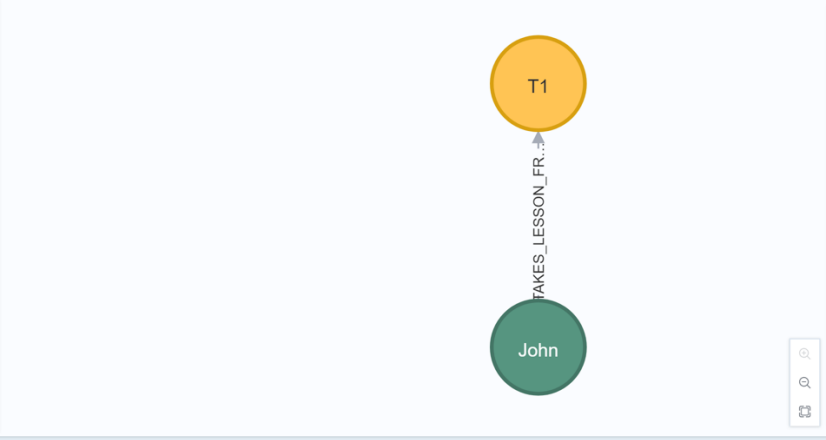
6. 2023-04-15 에 발생한 학교폭력 사건 조회

```
MATCH (incident:Incident)
WHERE incident.date = '2023-04-15'
RETURN incident
```



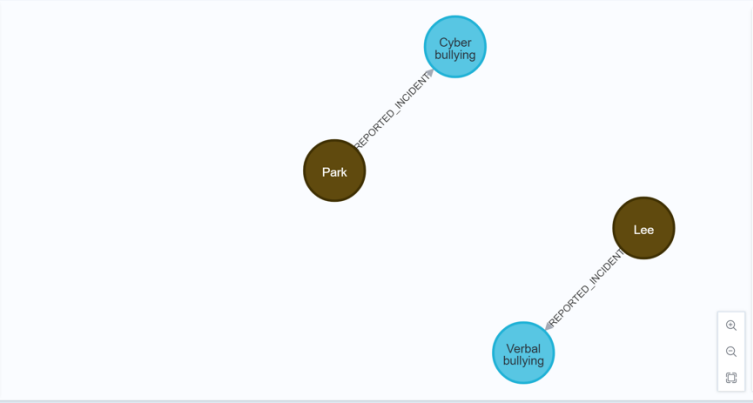
7. 학교폭력 예방센터에서 예방교육을 받은 학생 중 'Math'과목을 가르치는 선생님께 가르침을 받는 학생 목록 조회

```
MATCH (student)-[:RECEIVES_PREVENTION_EDUCATION]->(:Prevention_School)
MATCH (student)-[:TAKES_LESSON_FROM]->(teacher:Teacher {subject: 'Math'})
RETURN student, teacher;
```



8. 목격자가 존재하는 사건 조회

```
MATCH (incident:Incident)<-[:REPORTED_INCIDENT]-(witness:Witness_Student)
RETURN incident, witness;
```



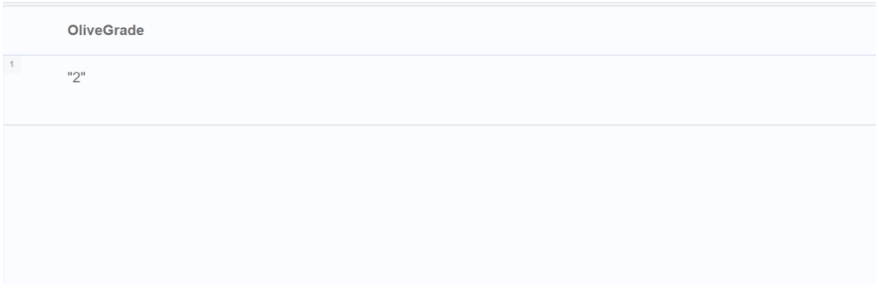
9. 피해 학생의 총 수 조회

```
MATCH (victim:Victim_Student)
RETURN COUNT(victim) AS TotalVictims;
```

	TotalVictims
1	4

10. Olive 학생의 grade 조회

```
MATCH (olive:Victim_Student {name: 'Olive'})
RETURN olive.grade AS OliveGrade;
```



	OliveGrade
1	"2"

2차과제

주제

- 파마 인디언 당뇨병 예측하기

데이터

- 샘플수 768
- Feature 8
  - Pregnant: 과거 임신 횟수
  - Plasma: 포도당 부하 검사 2 시간 후 공복 혈당 농도
  - Pressure: 확장기 혈압
  - Thickness: 삼두근 피부 주름 두께
  - Insulin: 혈청 인슐린
  - BMI: 체질량지수
  - Pedigree: 당뇨병 가족력
  - Age: 나이
- Class: 당뇨(1), 당뇨아님(0)

딥러닝 모델 구현

#구글 코랩 사용할 수 있는 환경 설정하기

```
from google.colab import drive
drive.mount('/content/gdrive')
```

#Kears 라이브러리를 사용하여 신경망 모델을 만들기 위한 코드

```
from keras.models import Sequential
from keras import layers
from keras import optimizers
```

#Sequential() : 모델 생성

#Dense() : 층 추가 (3 개 이상이면 딥러닝 모델)

마지막으로 출력된 층은 당뇨이다 아니다로 출력됨

```
model = Sequential()
model.add(layers.Dense(units=8, activation='relu'))
model.add(layers.Dense(units=16, activation='relu'))
model.add(layers.Dense(units=32, activation='relu'))
model.add(layers.Dense(units=64, activation='relu'))
model.add(layers.Dense(units=2, activation='softmax'))
```

loss : 손실(목적)함수 categorical_crossentropy 지정

#lr : 학습률=0.001 지정

```
model.compile(loss='categorical_crossentropy',
optimizer=optimizers.RMSprop(lr=0.001), metrics=['accuracy'])
```

#실행을 위한 코드 추가

```
import numpy as np
import pandas as pd
```

#데이터 불러오기

```
data_df =
pd.read_csv('/content/gdrive/MyDrive/ᄃᆡᄋᆡ ᄐᆞ ᄇᆡᄋᆡ ᄉᆞᅳᄋᆞᅳᆼᄋᆞᄑᆞᆼ/dia
betes.csv')
```

#예측 모델을 위한 데이터 분할

```
data_train = data_df.drop(['Outcome'], axis=1, inplace=False)
data_test = data_df
```

#기본적인 데이터 확인

```
data_test.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
data_train.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33

```
data_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 8 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
dtypes: float64(2), int64(6)
memory usage: 48.1 KB
```

#데이터 특성 처리

```
from tensorflow.keras.utils import to_categorical
data_train_np = np.zeros([data_train.shape[0], 8])
```

```
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
```

```
# 1. 특성 처리
data_train_np[:, 0:8] = StandardScaler().fit_transform(data_train)

# 2. 목표 변수 처리
data_train_np_y = to_categorical(data_df['Outcome'])

# 결과 확인
print("전처리된 데이터:")
print(data_train_np)
print("\n 전처리된 목표 변수:")
print(data_train_np_y)
```

```
전처리된 데이터:
[[ 0.63994726  0.84832379  0.14964075 ...  0.20401277  0.46849198
   1.4259954 ]
 [-0.84488505 -1.12339636 -0.16054575 ... -0.68442195 -0.36506078
  -0.19067191]
 [ 1.23388019  1.94372388 -0.26394125 ... -1.10325546  0.60439732
  -0.10558415]
 ...
 [ 0.3429808   0.00330087  0.14964075 ... -0.73518964 -0.68519336
  -0.27575966]
 [-0.84488505  0.1597866  -0.47073225 ... -0.24020459 -0.37110101
   1.17073215]
 [-0.84488505 -0.8730192   0.04624525 ... -0.20212881 -0.47378505
  -0.87137393]]

전처리된 목표 변수:
[[0. 1.]
 [1. 0.]
 [0. 1.]
 ...
 [1. 0.]
 [0. 1.]
 [1. 0.]]
```

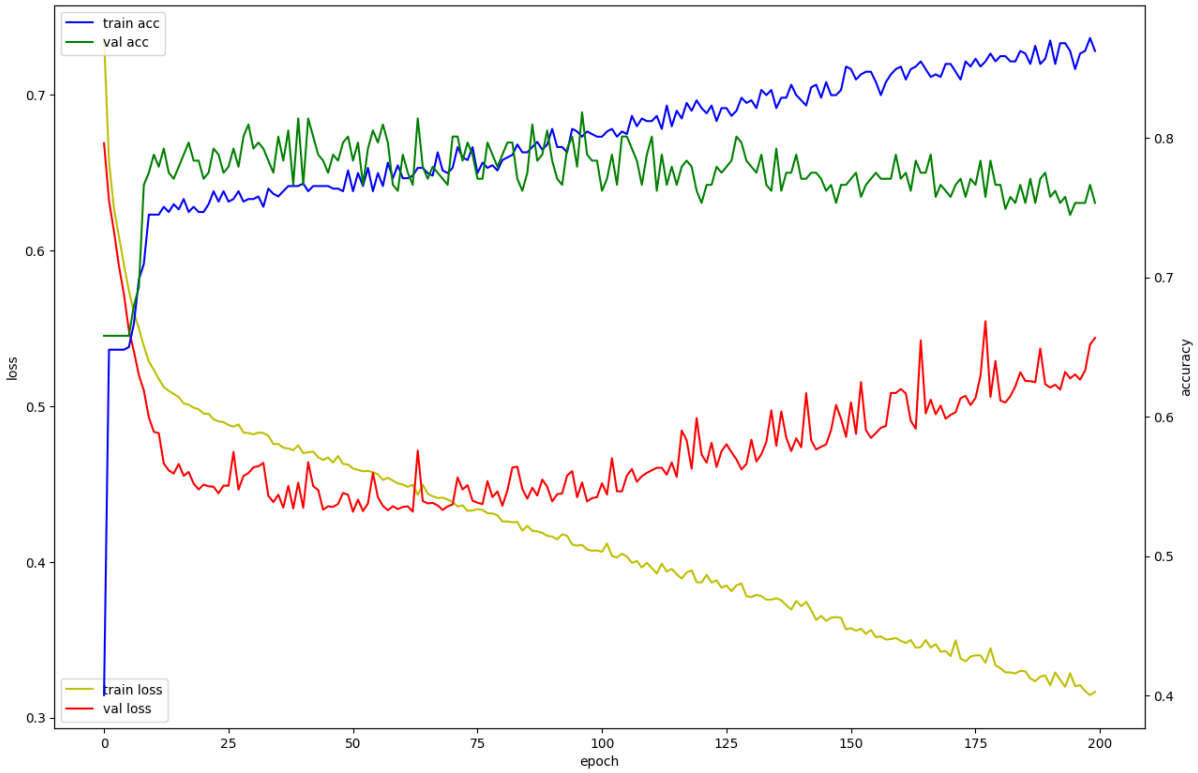
#model.fit() : 주어진 모델에 가중치를 조정하여 학습

```
hist = model.fit(data_train_np,data_train_np_y,epochs= 200,
batch_size=128, validation_split=0.3)
```

#loss 는 작을수록 accuracy 는 클수록 좋음

```
Epoch 1/200
5/5 [==============================] - 1s 71ms/step - loss: 0.7365 -
accuracy: 0.4004 - val_loss: 0.6689 - val_accuracy: 0.6580
Epoch 2/200
5/5 [==============================] - 0s 15ms/step - loss: 0.6570 -
accuracy: 0.6480 - val_loss: 0.6321 - val_accuracy: 0.6580
Epoch 3/200
5/5 [==============================] - 0s 11ms/step - loss: 0.6270 -
accuracy: 0.6480 - val_loss: 0.6124 - val_accuracy: 0.6580
Epoch 4/200
.
.
.
```

```
#모든 데이터 저장 -> 최종 모델 학습하는 그래프 -> 괜찮은지 보고 그래프 모델 뽑음
#학습 할수록 loss 값이 작아짐
#val_loss 갑자기 커지기도 .. 지나친 학습으로 일반화능력 떨어져서 잘 맞추지 못함 -
>해결 : 기계가 얼리스탑핑
import matplotlib.pyplot as plt

def drawHistory(hist):
  fig = plt.figure(figsize=(15,10)) # 10x15 크기의 figure 생성
  loss_ax = plt.gca()         # figure 의 기본 축 가져오기
  acc_ax = loss_ax.twinx()       # 축을 1 개 더 추가

  #손실 변화 양상을 그래프로 표시
  loss_ax.plot(hist.history['loss'], 'y', label='train loss')
  loss_ax.plot(hist.history['val_loss'], 'r', label='val loss')
  loss_ax.set_xlabel('epoch')
  loss_ax.set_ylabel('loss')
  loss_ax.legend(loc='lower left')

  #정확도 변화 양상을 그래프로 표시
  acc_ax.plot(hist.history['accuracy'], 'b', label='train acc')
  acc_ax.plot(hist.history['val_accuracy'], 'g', label='val acc')
  acc_ax.set_ylabel('accuracy')
  acc_ax.legend(loc='upper left')

  plt.show() #그래프를 화면에 그림

drawHistory(hist) #학습 경과를 그래프로 그리는 함수 호출
```




```
# 데이터 전처리
data_test_np = np.zeros((data_test.shape[0], 8))

# 과거 임신 횟수, 포도당 부하 검사 2 시간 후 공복 혈당 농도, 확장기 혈압 정규화
data_test_np[:, 0] = data_test['Pregnancies'] / 17  # 임신 횟수의 최댓값을 17 이라고 가정
data_test_np[:, 1] = data_test['Glucose'] / 199  # 포도당 부하 검사 값의 최댓값을 199 이라고 가정
data_test_np[:, 2] = data_test['BloodPressure'] / 122  # 확장기 혈압 값의 최댓값을 122 이라고 가정

# 삼두근 피부 주름 두께, 혈청 인슐린, 체질량지수, 당뇨병 가족력 정규화
data_test_np[:, 3] = data_test['SkinThickness'] / 99  # 삼두근 피부 주름 두께의 최댓값을 99 이라고 가정
data_test_np[:, 4] = data_test['Insulin'] / 846  # 혈청 인슐린 값의 최댓값을 846 이라고 가정
data_test_np[:, 5] = data_test['BMI'] / 67.1  # 체질량지수의 최댓값을 67.1 이라고 가정
```

```
data_test_np[:, 6] = data_test['DiabetesPedigreeFunction'] / 2.42  # 당뇨병 가족력의 최댓값을 2.42 라고 가정

# 나이 정규화
data_test_np[:, 7] = data_test['Age'] / 81  # 나이의 최댓값을 81 이라고 가정

# 목표 변수 처리
data_test_np_y = to_categorical(data_test['Outcome'])

# NaN 값 처리
data_test_np[np.isnan(data_test_np)] = 30 / 80

# 결과 확인
print("전처리된 테스트 데이터:")
print(data_test_np)
print("\n 전처리된 목표 변수:")
print(data_test_np_y)
```

#학습된 모델에 대한 예측 수행

```
o = model.predict(data_test_np)
print(o)
```

#출력값

```
[[0.2968571  0.70314294]
 [0.4818387  0.5181613 ]
 [0.514875   0.4851249 ]
 ...
 [0.49105287 0.508947  ]
 [0.34580123 0.65419865]
 [0.51724654 0.48275334]]
```

[당뇨병이 아닐 확률, 당뇨병일 확률]

```
o = np.argmax(o,-1)
print(o)
```

#출력값 1 이 당뇨병으로 예측된 값

```
[1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 0 1 0 1 1 0 1 1 1 0 0 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 0 1 0 1 0 0 1 0 0 1 1 1 1 1 1 1
1 1 0 0 0 1 1 0 1 1 0 0 0 1 1 0 1 0 1 1 1 1 1 0 1 0 0 0 0 1 1 1 0 1 0 1
1 0 0 1 1 0 1 1 1 1 1 0 0 1 1 0 1 1 0 1 0 1 1 1 0 1 1 0 1 1 0 1 0 1 1 0
1 0 0 0 1 1 0 0 1 0 0 1 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1
1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 0 0 1 1 0 1 1 1 1 1
0 0 1 0 1 1 1 1 1 1 1 0 1 1 0 0 1 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 1 0 0 1 1
1 1 0 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 0 0 0 1 0 1 1 1 1 1 1 1
0 0 1 0 1 1 0 1 1 0 0 0 0 0 1 1 0 1 1 0 1 1 1 1 1 1 0 1 1 0 1 1 1 0 1 1 1 1
1 0 1 1 1 0 1 0 0 1 0 0 0 1 1 0 1 1 1 0 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 0
1 1 1 1 1 0 1 0 1 0 1 1 1 0 0 1 1 0 0 1 1 1 1 0 1 1 1 1 1 1 0 0 1 0 1 1
0 1 1 0 1 1 1 1 1 1 1 0 1 1 1 0 1 0 0 1 0 0 1 1 0 1 0 0 0 1 1 1 0 1 0 0 0
1 1 1 1 0 1 1 1 1 1 1 0 0 0 1 1 1 0 0 1 0 1 0 0 0 1 1 0 1 1 0 1 1 1 0 1 0
0 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 0 0 1 1 1 0 1 1 1 0 1 1 1 1 0 1 1 0
1 1 1 1 0 1 1 0 0 1 1 0 1 0 1 0 1 0 0 0 0 1 1 0 0 0 0 1 0 1 0 1 1 1 1 0 0
1 1 1 1 0 0 1 1 1 1 1 1 1 0 1 0 0 0 0 1 1 1 0 1 1 0 0 1 0 0 0 0 1 0 1 1 0 0 0
0 0 1 0 1 1 0 1 1 0 1 1 0 1 0 1 0 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0
1 1 0 1 1 1 0 0 0 0 1 1 0 1 0 1 1 1 0 0 0 1 0 0 0 1 1 1 1 1 1 1 1 0 0 1
1 0 1 0 1 1 1 0 0 0 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 0 1 1 1 0 1 0 0 1 0
1 0 0 0 1 0 1 0 1 1 1 1 0 1 0 1 1 0 1 1 0 1 0 1 0 1 0 0 1 1 0 1 1 1 0 0
1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 0 1 1 0 1 0 1 1 1 0 1 0 0 1 1 1 1 0 1 1 0 1
1 1 1 1 1 1 1 1 0 0 1 1 1 0]
```