• 당뇨병 예측 딥러닝

UCI 머신 러닝 저장소(UCI Machine Learning Repository)에서 가져온 피마 인디언들의 당뇨병에 관한 데이터

1. 데이터 분석을 위해서 pandas 라이브러리 임포트하기

In [1]:
```
import pandas as pd
```

2. 준비된 데이터셋(빅데이터)을 pandas 기반으로 읽어오기

In [2]:
```
# 피마 인디언 당뇨병 데이터셋을 불러옵니다. 불러올 때 각 컬럼에 해당하는 이름을 지정합니다.
df = pd.read_csv('./dataset/pima-indians-diabetes.csv',
                 names = ["pregnant", "plasma", "pressure", "thickness", "insulin", "BMI", "pedigree", "age", "class"])

#df_ko = pd.read_csv('dataset/pima-indians-diabetes.csv',
#                    names = ["임신회수", "혈당", "혈압", "피부두께", "인슐린", "BMI", "가족력", "나이", "class"])
```

In [3]:
```
1 df
```

Out[3]:

	pregnant	plasma	pressure	thickness	insulin	BMI	pedigree	age	class
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
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

3. 데이터 분석하기

In [4]:
```
1 # 처음 5줄을 봅니다.
2 df.head(5)
```

Out[4]:

	pregnant	plasma	pressure	thickness	insulin	BMI	pedigree	age	class
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

In [5]:
```
# 데이터의 전반적인 정보를 확인해 봅니다.
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   pregnant   768 non-null    int64
 1   plasma     768 non-null    int64
 2   pressure   768 non-null    int64
 3   thickness  768 non-null    int64
 4   insulin    768 non-null    int64
 5   BMI        768 non-null    float64
 6   pedigree   768 non-null    float64
 7   age        768 non-null    int64
 8   class      768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [6]:
```
# 각 정보별 특징을 좀더 자세히 출력합니다.
# 정보별 샘플 수(count) , 평균(mean) , 표준편차(std), 최솟값(min), 백분위 수로 25%, 50%, 75%에 해당하는 값
# 그리고 최대값(max)이 정리되어 보임
df.describe()
```

Out[6]:

	pregnant	plasma	pressure	thickness	insulin	BMI	pedigree	age	class
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

In [7]:
```
# 데이터 중 임신횟수 정보와 클래스만을 출력해 봅니다.
df[['pregnant', 'class']]
```

Out[7]:

	pregnant	class
0	6	1
1	1	0
2	8	1
3	1	0
4	0	1
...	...	...
763	10	0
764	2	0
765	5	0
766	1	1
767	1	0

768 rows × 2 columns

많은 데이터를 단순히 나열하는 것은 한눈에 들어오지 않으므로 큰 의미가 없다 데이터를 잘 다루려면 데이터를 한 번 더 가공해야 함
< 데이터를 가공할 때의 주의점 >
우리가 무엇을 위해 작업을 하는지 그 목적을 잊어서는 안 됨
이 프로젝트의 목적 : 당뇨병 발병을 예측하는 것
그렇다면 모든 정보는 당뇨병 발병과 어떤 관계가 있는지를 중점에 놓아야 함

3.1. 임신횟수를 기반으로 당뇨병 발병 확률 분석하기

pandas에서 제공하는 groupby() 함수를 사용해 pregnant 정보를 기준으로 하는 새 그룹을 만듬
as_index=False는 pregnant 정보 좌측에 0, 1, 2 … 와 같은 새로운 인덱스(index)를 만듬
mean() 함수를 사용해 평균을 구하고,

sort_values() 함수를 써서 pregnant 컬럼을 ascending=True 으로 오름차순(ascending) 정렬하여
임신 횟수 당 당뇨병 발병 확률을 출력함

In [8]:
```
#print(df[['pregnant','class']].groupby(['pregnant'], as_index=False).mean().sort_values(by='pregnant', ascending=True))

pregnant_analysis = df[['pregnant','class']].groupby(['pregnant'], as_index=False).mean().sort_values(by='pregnant',
                                                                                                      ascending=True)
pregnant_analysis
```

Out[8]:

	pregnant	class
0	0	0.342342
1	1	0.214815
2	2	0.184466
3	3	0.360000
4	4	0.338235
5	5	0.368421
6	6	0.320000
7	7	0.555556
8	8	0.578947
9	9	0.642857
10	10	0.416667
11	11	0.636364
12	12	0.444444
13	13	0.500000
14	14	1.000000
15	15	1.000000
16	17	1.000000

3.2. 혈당을 기반으로 당뇨병 발병 확률 분석하기

In [9]:
```
#print(df[['plasma','class']].groupby(['plasma'], as_index=False).mean().sort_values(by='plasma', ascending=True))

plasma_analysis = df[['plasma','class']].groupby(['plasma'], as_index=False).mean().sort_values(by='plasma', ascending=True
plasma_analysis
```

Out[9]:

	plasma	class
0	0	0.40
1	44	0.00
2	56	0.00
3	57	0.00
4	61	0.00
...	...	...
131	195	1.00
132	196	1.00
133	197	0.75
134	198	1.00
135	199	1.00

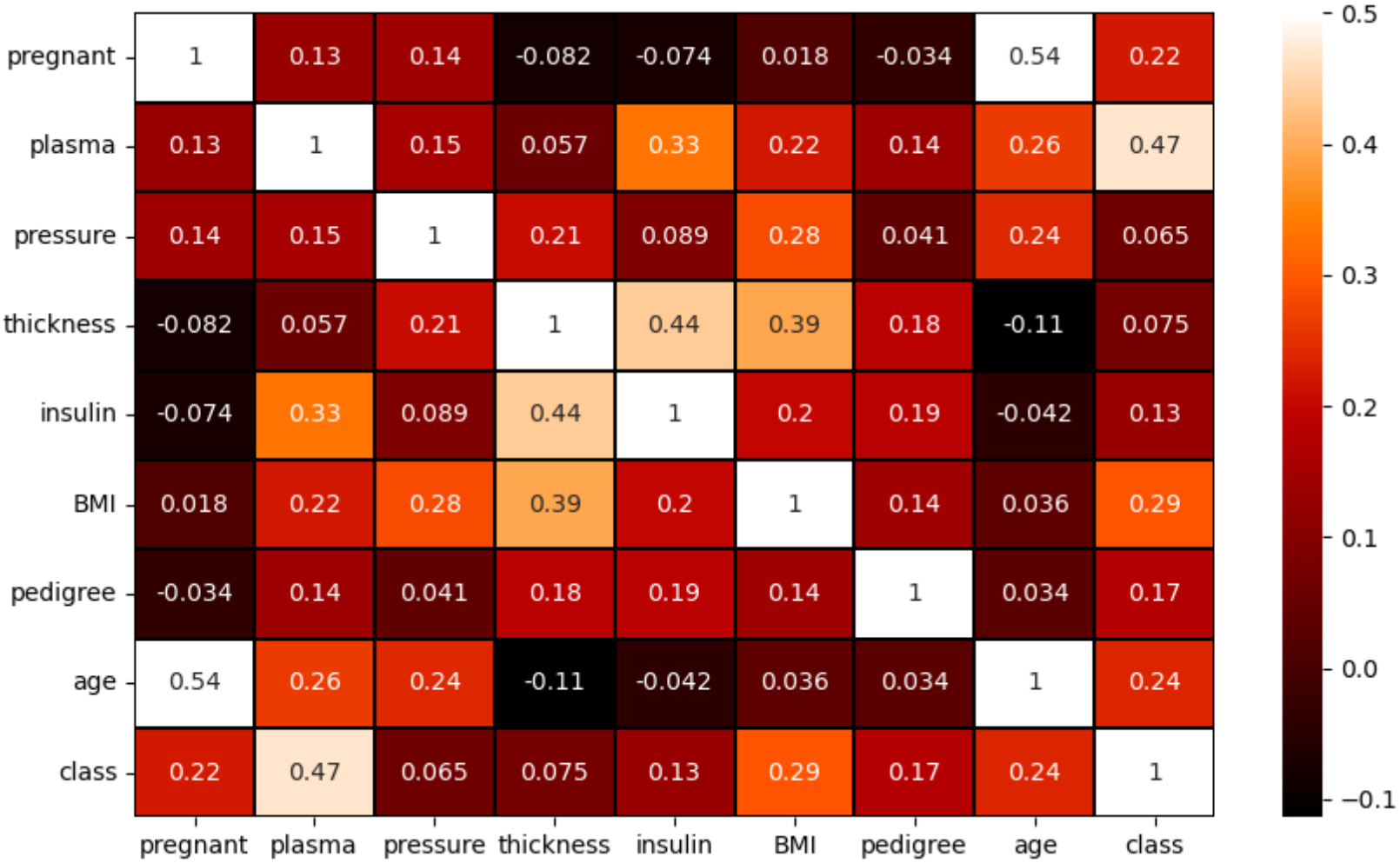
136 rows × 2 columns

3.3. 데이터의 상관관계를 그래프로 시각화하여 분석하기

- heatmap 그래프 (1) : cmap=colormap

In [10]:

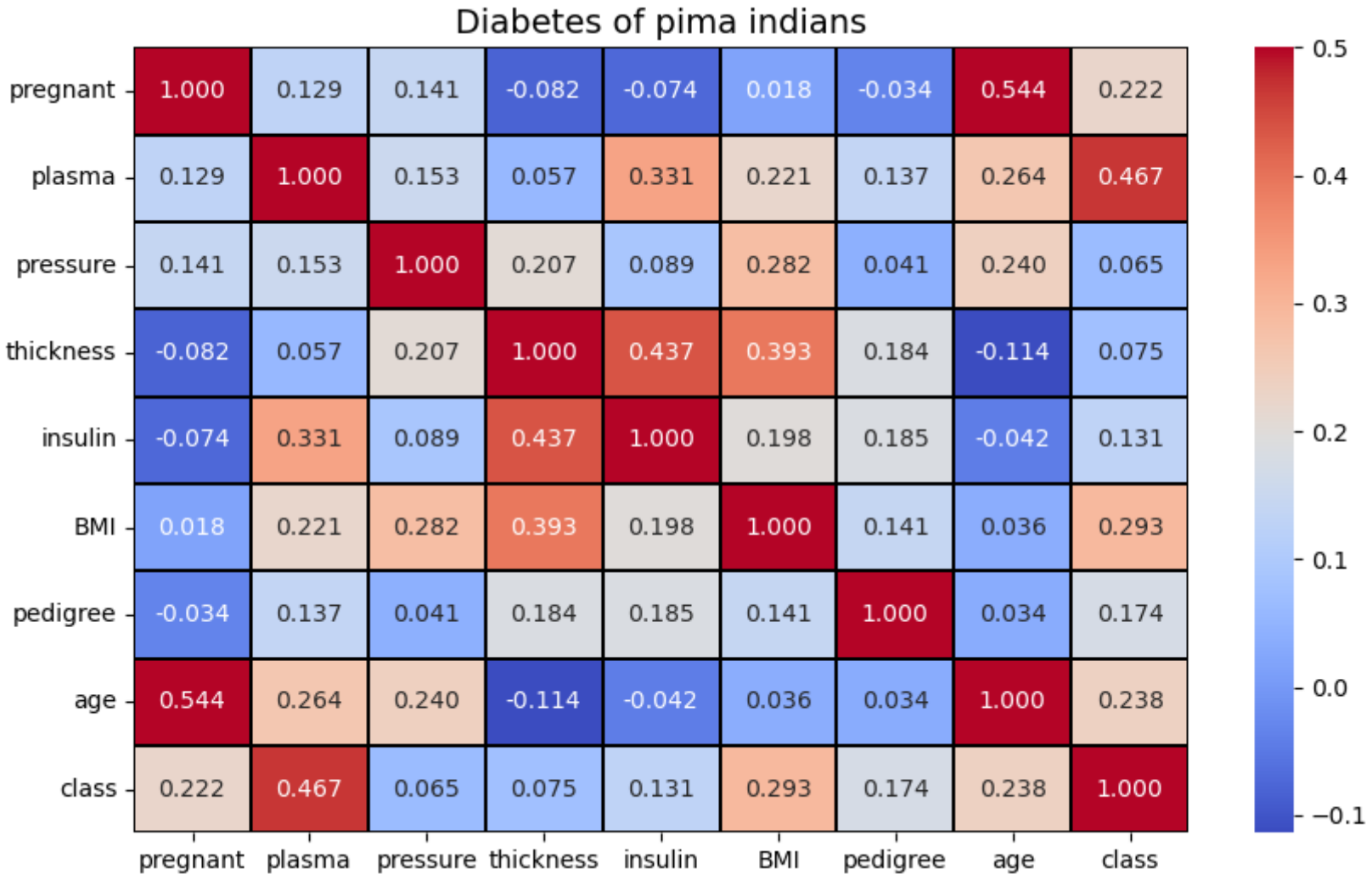
```
import matplotlib.pyplot as plt
import seaborn as sns
#%matplotlib qt5

colormap = plt.cm.gist_heat   #matplotlib.pyplot에서 제공하는 색상 구성을 사용할 수 있도록 셋팅
plt.figure(figsize=(10,6))   #그래프의 크기 셋팅

# seaborn에서 제공하는 heatmap : 두 항목씩 짝을 지은 뒤 각각 어떤 패턴으로 변화하는지를 관찰하는 함수
# 두 항목이 전혀 다른 패턴으로 변화하고 있으면 0을, 서로 비슷한 패턴으로 변할수록 1에 가까운 값을 출력함
# df.corr() 상관 분석을 제공
# vmax의 값을 0.5로 지정할 경우, 0.5이상부터 흰색으로 표시됨
# 그래프 위에 값이 출력되게 하려면 annot=True

sns.heatmap(df.corr(), cmap=colormap, linewidths=0.1,vmax=0.5,  linecolor='black', annot=True)
plt.show()
```



● heatmap 그래프 (2) : cmap='coolwarm'

In [11]:

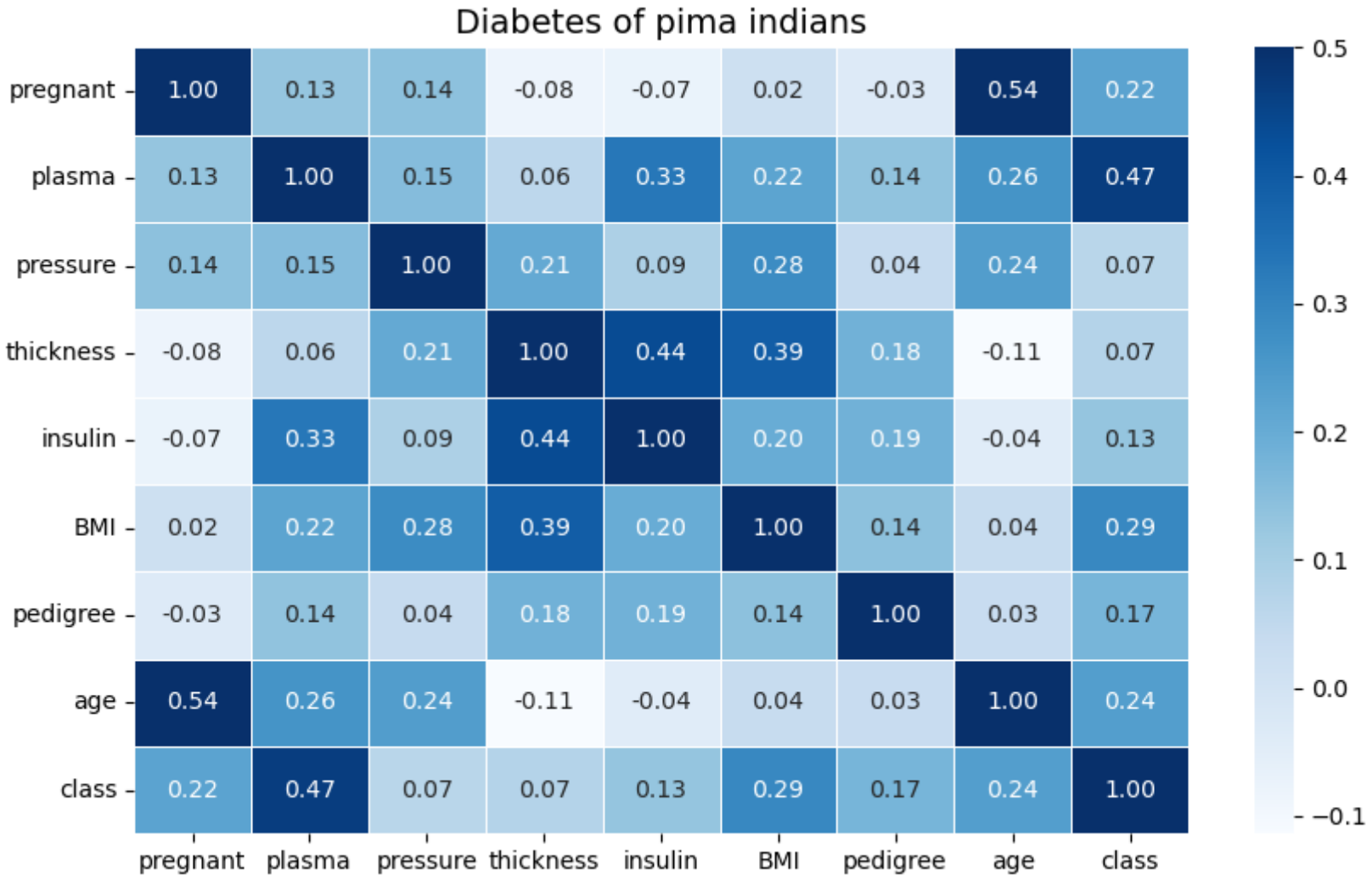
```
plt.figure(figsize=(10,6))
plt.title('Diabetes of pima indians', fontsize=14)
sns.heatmap(df.corr(), cmap='coolwarm', linewidths=0.2, vmax=0.5, linecolor='black', fmt = '.3f', annot=True)
plt.show()
```



- heatmap 그래프 (3) : cmap='Blues' / cmap='BrBG'

In [12]:

```
plt.figure(figsize=(10,6))   #그래프의 크기를 정합니다.
plt.title('Diabetes of pima indians', fontsize=14)
sns.heatmap(df.corr(), cmap='Blues',  linewidths=0.5, vmax=0.5, linecolor='white', fmt = '.2f', annot=True)
#sns.heatmap(df.corr(), cmap='BrBG',  linewidths=0.5, vmax=0.5, linecolor='white', fmt = '.2f', annot=True)

# 이미지 저장하기
plt.savefig('DL_RESULT/Diabetes.png')
plt.show()
```

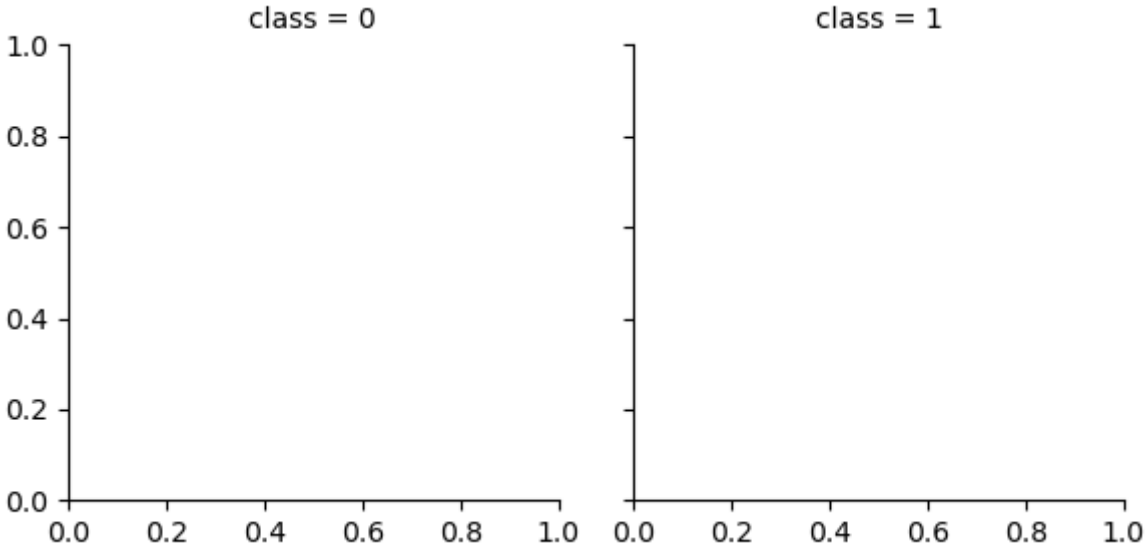


그래프를 통해 plasma (공복 혈당 농도)는 0.47의 수치로 class 항목과 가장 상관관계가 높다는 것을 알 수 있음
즉, plasma (공복 혈당 농도)가 당뇨병과 연관이 높다는 것을 분석을 통해 알게됨
이제 plasma와 class 항목만 따로 떼어 두 항목 간의 관계를 그래프로 다시 한번 확인해보자.

● FacetGrid 그래프

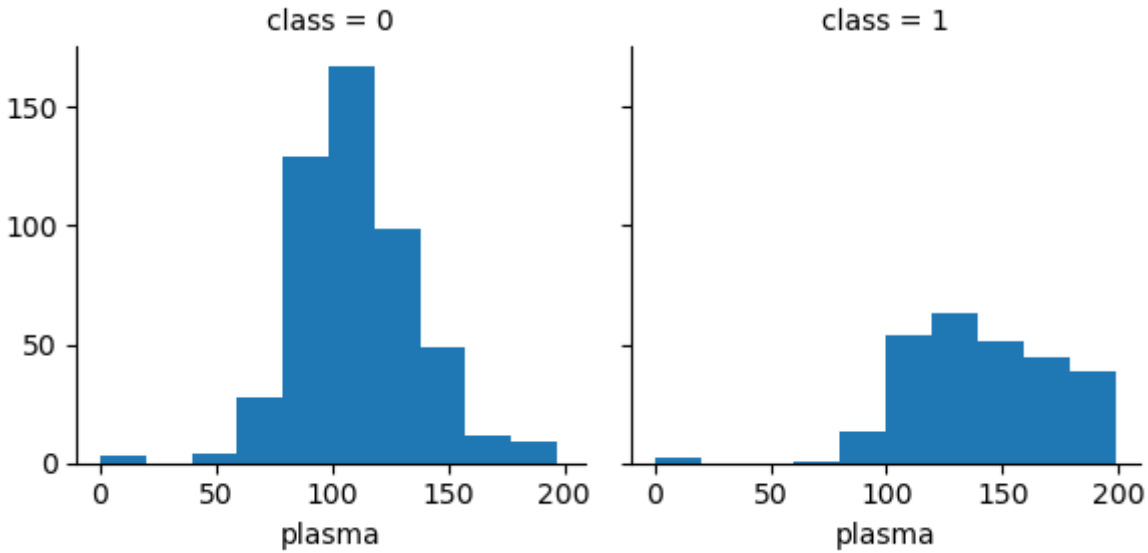
FacetGrid 패싯그리드 다양한 범주형 값을 가지는 데이터를 시각화하는데 좋은 방법
행, 열 방향으로 서로 다른 조건을 적용하여 여러 개의 서브 플롯 제작
각 서브 플롯에 적용할 그래프 종류를 map() 메서드를 이용하여 그리드 객체에 전달

In [13]:
```
# sns.FacetGrid(df, col='class') 에서 col='class' 사용하는 이유는
# 0과 1에 해당하는 2개의 column으로 분리하여 히스토그램으로 쌓는 구조로 그래프를 출력하기 위해서

grid = sns.FacetGrid(df, col='class')
grid
plt.show()
```



In [14]:

```
grid = sns.FacetGrid(df, col='class')
grid.map(plt.hist, 'plasma', bins=10)  # plasma -> x축, bins=10 -> 10개의 구간으로 분리하여 히스토그램 완성
plt.show()

# 당뇨병 환자의 경우 : class가 1에 해당하며 plasma 수치가 150이상인 경우가 많다는 것을 확인
```



4. 딥러닝에 필요한 라이브러리 임포트하기

In [15]:
```
# 딥러닝을 구동하는 데 필요한 케라스 함수를 불러옵니다.
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

from sklearn.model_selection import train_test_split

# 필요한 라이브러리를 불러옵니다.
import numpy as np
import tensorflow as tf

# 실행할 때마다 같은 결과를 출력하기 위해 설정하는 부분입니다.
seed=2
np.random.seed(seed)
tf.random.set_seed(seed)
```

우리가 random( ) 함수를 써서 임의의 숫자를 만들어 내는 것처럼 보여도 이는 컴퓨터 안에 미리 내장된 수많은
'랜덤 테이블' 중 하나를 불러내 그 표의 순서대로 숫자를 보여 주는 것
seed 값을 설정한다는 것은 그 랜덤 테이블 중에서 몇 번째 테이블을 불러와 사용할것인지를 정하는 것
seed 값이 같으면 똑같은 랜덤 값을 출력함!

넘파이 라이브러리를 사용하면서 텐서플로 기반으로 딥러닝을 구현할 때는 일정한 결과값을 얻기 위해
넘파이 seed 값과 텐서플로 seed 값을 모두 설정해야 함
최종 딥러닝 결과는 다양한 seed를 여러 번 실행하여 평균을 구하는 것이 가장 적절함

5. 준비된 데이터셋(빅데이터)을 numpy 기반으로 읽어오기

In [16]:
```
# 데이터를 불러 옵니다.
dataset = np.loadtxt("./dataset/pima-indians-diabetes.csv", delimiter=",")
X = dataset[:,0:8]
Y = dataset[:,8]
```

In [17]:
```
1 X
```

Out[17]:
```
array([[  6.   , 148.   ,  72.   , ...,  33.6  ,   0.627,  50.   ],
       [  1.   ,  85.   ,  66.   , ...,  26.6  ,   0.351,  31.   ],
       [  8.   , 183.   ,  64.   , ...,  23.3  ,   0.672,  32.   ],
       ...,
       [  5.   , 121.   ,  72.   , ...,  26.2  ,   0.245,  30.   ],
       [  1.   , 126.   ,  60.   , ...,  30.1  ,   0.349,  47.   ],
       [  1.   ,  93.   ,  70.   , ...,  30.4  ,   0.315,  23.   ]])
```

In [18]:
```
1 Y
```

Out[18]:
```
array([1., 0., 1., 0., 1., 0., 1., 0., 1., 1., 0., 1., 0., 1., 1., 1., 1.,
       1., 0., 1., 0., 0., 1., 1., 1., 1., 1., 0., 0., 0., 0., 1., 0., 0.,
       0., 0., 0., 1., 1., 1., 0., 0., 0., 1., 0., 1., 0., 0., 1., 0., 0.,
       0., 0., 1., 0., 0., 1., 0., 0., 0., 0., 1., 0., 0., 1., 0., 1., 0.,
       0., 0., 1., 0., 1., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 1.,
       0., 0., 0., 1., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 1., 1., 0.,
       0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 0., 0., 1., 1., 1., 0., 0.,
       0., 1., 0., 0., 0., 1., 1., 0., 0., 1., 1., 1., 1., 1., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
       0., 1., 1., 0., 0., 0., 1., 0., 0., 0., 0., 1., 1., 0., 0., 0., 0.,
       1., 1., 0., 0., 0., 1., 0., 1., 0., 1., 0., 0., 0., 0., 0., 1., 1.,
       1., 1., 1., 0., 0., 1., 1., 0., 1., 0., 1., 1., 1., 0., 0., 0., 0.,
       0., 0., 1., 1., 0., 1., 0., 0., 0., 1., 1., 1., 1., 0., 1., 1., 1.,
       1., 0., 0., 0., 0., 0., 1., 0., 0., 1., 1., 0., 0., 0., 1., 1., 1.,
       1., 0., 0., 0., 1., 1., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
       1., 0., 0., 0., 1., 0., 1., 0., 0., 1., 0., 1., 0., 0., 1., 1., 0.,
       0., 0., 0., 0., 1., 0., 0., 0., 1., 0., 0., 1., 1., 0., 0., 1., 0.,
       0., 0., 1., 1., 1., 0., 0., 1., 0., 1., 0., 1., 1., 0., 1., 0., 0.,
       1., 0., 1., 1., 0., 0., 1., 0., 1., 0., 0., 1., 0., 1., 0., 1., 1.,
       1., 0., 0., 1., 0., 1., 0., 0., 0., 1., 0., 0., 0., 0., 1., 1., 1.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 1., 1.,
       1., 0., 1., 1., 0., 0., 1., 0., 0., 1., 0., 0., 1., 1., 0., 0., 0.,
       0., 1., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 0., 0.,
       1., 0., 0., 1., 0., 0., 1., 0., 1., 1., 0., 1., 0., 1., 0., 1., 0.,
       1., 1., 0., 0., 0., 0., 1., 1., 0., 1., 0., 1., 0., 0., 0., 0., 1.,
       1., 0., 1., 0., 1., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 1., 0.,
       0., 1., 1., 1., 0., 0., 1., 0., 0., 1., 0., 0., 0., 1., 0., 0., 1.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.,
       1., 0., 0., 0., 1., 0., 0., 0., 1., 1., 0., 0., 0., 0., 0., 0., 0.,
       1., 0., 0., 0., 0., 1., 0., 0., 0., 1., 0., 0., 0., 1., 0., 0., 0.,
       1., 0., 0., 0., 0., 1., 1., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 1., 1., 1., 1., 0.,
       0., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
       1., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 1.,
       0., 1., 1., 0., 0., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 0.,
       1., 0., 0., 1., 0., 0., 0., 0., 1., 1., 0., 1., 0., 0., 0., 0., 1.,
       1., 0., 1., 0., 0., 0., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 1., 0., 0., 0., 0., 1., 0., 0., 1., 0., 0., 0., 1., 0., 0., 0.,
       1., 1., 1., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 1., 0., 1., 1.,
```

```
1., 1., 0., 1., 1., 0., 0., 0., 0., 0., 0., 0., 1., 1., 0., 1., 0.,
0., 1., 0., 1., 0., 0., 0., 0., 0., 1., 0., 1., 0., 1., 0., 1., 1.,
0., 0., 0., 0., 1., 1., 0., 0., 0., 1., 0., 1., 1., 0., 0., 1., 0.,
0., 1., 1., 0., 0., 1., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 1.,
1., 1., 0., 0., 0., 0., 0., 0., 1., 1., 0., 0., 1., 0., 0., 1., 0.,
1., 1., 1., 0., 0., 1., 1., 1., 0., 1., 0., 1., 0., 1., 0., 0., 0.,
0., 1., 0.])
```

6. 읽어온 데이터셋을 학습용과 테스트용으로 분리하기

In [19]:
```
#학습 데이터 70 %, 테스트 데이터셋 30% 로 설정하기
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=seed)
```

In [20]:
```
print( "학습 데이터 개수 : %d  ,  테스트 데이터 개수 %d  "  %(len(X_train), len(X_test)))
```

학습 데이터 개수 : 537 , 테스트 데이터 개수 231

In [21]:

```
print('● type(X_train) : ', type(X_train)) # train_test_split( ) 함수의 리턴 값은 numpy.ndarray 타입
print('● np.shape(X_train) : ',  np.shape(X_train)) # 학습 데이터 모양
print('● len(X_train) : ', len(X_train), '개') # 학습 데이터 개수
print(X_train)  #학습 데이터 속성 확인
print('-'*100)
print('● type(X_test) : ', type(X_test)) # train_test_split( ) 함수의 리턴 값은 numpy.ndarray 타입
print('● np.shape(X_test) : ',  np.shape(X_test)) # 테스트 데이터 모양
print('● len(X_test) : ', len(X_test), '개') # 테스트 데이터 개수
print(X_test)  #테스트 데이터 속성 확인
```

```
● type(X_train) :  <class 'numpy.ndarray'>
● np.shape(X_train) :  (537, 8)
● len(X_train) :  537 개
[[3.000e+00 8.400e+01 7.200e+01 ... 3.720e+01 2.670e-01 2.800e+01]
 [0.000e+00 1.180e+02 6.400e+01 ... 0.000e+00 1.731e+00 2.100e+01]
 [1.000e+01 9.200e+01 6.200e+01 ... 2.590e+01 1.670e-01 3.100e+01]
 ...
 [4.000e+00 1.250e+02 7.000e+01 ... 2.890e+01 1.144e+00 4.500e+01]
 [3.000e+00 1.160e+02 7.400e+01 ... 2.630e+01 1.070e-01 2.400e+01]
 [4.000e+00 1.100e+02 6.600e+01 ... 3.190e+01 4.710e-01 2.900e+01]]
----------------------------------------------------------------------------------------------------
● type(X_test) :  <class 'numpy.ndarray'>
● np.shape(X_test) :  (231, 8)
● len(X_test) :  231 개
[[2.00e+00 8.80e+01 7.40e+01 ... 2.90e+01 2.29e-01 2.20e+01]
 [2.00e+00 1.29e+02 8.40e+01 ... 2.80e+01 2.84e-01 2.70e+01]
 [0.00e+00 1.02e+02 7.80e+01 ... 3.45e+01 2.38e-01 2.40e+01]
 ...
 [0.00e+00 1.62e+02 7.60e+01 ... 5.32e+01 7.59e-01 2.50e+01]
```

7. 딥러닝 모델 설계하기

In [22]:
```
# 딥러닝 구조를 결정합니다(모델을 설정하는 부분)

model = Sequential() #keras에서 제공되는 기능

#1번째 층 : 입력 x는 8개, 출력은 12개, 활성화함수 relu
model.add(Dense(12, input_dim=8, activation='relu'))

#2번째 층 : 입력 x는 12개, 출력은 8개, 활성화함수 relu
model.add(Dense(8, activation='relu'))

#3번째 층 : 입력 x는 8개, 출력은 1개, 활성화함수 sigmoid
model.add(Dense(1, activation='sigmoid'))
```

In [23]:
```
model
```

Out[23]:
```
<keras.engine.sequential.Sequential at 0x2895a2b7a30>
```

8. 딥러닝 모델 컴파일 및 실행하기

keras의 Sequential()에서 제공하는 compile( ) 기능으로 loss, optimizer, metrics 설정

In [24]:
```
#옵션 중에서 loss='binary_crossentropy'는 시그모이드를 이용한 2진 분류에서 사용
model.compile(loss='binary_crossentropy', optimizer='adam',  metrics=['accuracy'])
```

keras의 Sequential()에서 제공하는 fit( ) 기능으로 x, y, 학습 반복횟수, 배치 사이즈 설정

In [25]:
```
#batch_size=1로 하면, 537개의 데이터를 1개씩 처리하므로 실행 결과에서 537/537 출력
#model.fit(X_train, Y_train, epochs=200, batch_size=1) # 학습 데이터셋 537 개로 학습

#batch_size=1로 하면, 537개의 데이터를 10개씩 처리하므로 실행 결과에서 54/54 출력
model.fit(X_train, Y_train, epochs=200, batch_size=10) # 학습 데이터셋 537 개로 학습
```

```
Epoch 177/200
54/54 [==============================] - 0s 1ms/step - loss: 0.5116 - accuracy: 0.7095
Epoch 178/200
54/54 [==============================] - 0s 2ms/step - loss: 0.5221 - accuracy: 0.7132
Epoch 179/200
54/54 [==============================] - 0s 1ms/step - loss: 0.5104 - accuracy: 0.7244
Epoch 180/200
54/54 [==============================] - 0s 2ms/step - loss: 0.5163 - accuracy: 0.7076
Epoch 181/200
54/54 [==============================] - 0s 1ms/step - loss: 0.5101 - accuracy: 0.7188
Epoch 182/200
54/54 [==============================] - 0s 1ms/step - loss: 0.5111 - accuracy: 0.7114
Epoch 183/200
54/54 [==============================] - 0s 1ms/step - loss: 0.5102 - accuracy: 0.7188
Epoch 184/200
54/54 [==============================] - 0s 1ms/step - loss: 0.5261 - accuracy: 0.7169

Epoch 185/200
54/54 [==============================] - 0s 2ms/step - loss: 0.5159 - accuracy: 0.7076
Epoch 186/200
54/54 [==============================] - 0s 2ms/step - loss: 0.5182 - accuracy: 0.7207
```

9. 테스트 데이터를 기반으로 딥러닝 평가하기

keras의 Sequential()에서 제공하는 model.evaluate( ) 기능 사용
리턴 값 : [loss 오차 , acc 정확도]

In [26]:
```
print('● 테스트 데이터 개수 : ', len(X_test), '개')
ev = model.evaluate(X_test, Y_test, batch_size=1) #디폴트 출력 내용 loss, accuracy 있음
print('● [ loss 오차 , accuracy 정확도] =' , ev)
```

```
● 테스트 데이터 개수 :  231 개
231/231 [==============================] - 0s 1ms/step - loss: 0.5982 - accuracy: 0.7100
● [ loss 오차 , accuracy 정확도] = [0.5982044339179993, 0.709956705570221]
```

In [27]:
```
# 30%에 해당하는 231개의 테스트 데이터셋으로 정확도 계산 결과를 출력합니다.
# batch_size=1로 하면, 231개의 데이터를 1개씩 처리하므로 실행 결과에서 231/231 출력
#print("₩n Test Accuracy : %.4f" % (model.evaluate(X_test, Y_test, batch_size=1)[1]))

# batch_size=1로 하면, 231개의 데이터를 10개씩 처리하므로 실행 결과에서 24/24 출력
print("₩n Test Accuracy : %.4f" % (model.evaluate(X_test, Y_test, batch_size=10)[1]))
```

```
24/24 [==============================] - 0s 1ms/step - loss: 0.5982 - accuracy: 0.7100

 Test Accuracy : 0.7100
```

<참고>
밀리 초, millisecond (ms, msec)는 10-3(1000분의 1)초에 해당하는 시간의 단위
마이크로 초, microsecond (us)는 10-6(100만분의 1)초에 해당하는 시간의 단위
피코 초, picosecond는 10-12(1조분의 1)초에 해당하는 시간의 단위
펨토 초, femtosecond는 10-15(1000조분의 1)초에 해당하는 시간의 단위
아토 초, attosecond는 10-18초에 해당하는 시간의 단위

10. 새로운 데이터를 대상으로 딥러닝 모델을 사용하여 예측하기

keras의 Sequential()에서 제공하는 predict() 기능 사용
predict()은 학습 및 테스트에 사용하지 않은 데이터를 사용하는 것이 바람직..

In [28]:
```
kim = np.array([[3, 78, 50, 32, 88, 31, 0.248, 26]])
park = np.array([[10, 115, 0, 0, 0, 35.3, 0.134, 29]])
choi = np.array([[2, 197, 70, 45, 543, 30.5, 0.158, 53]])

test_kim  = model.predict(kim)*100
test_park = model.predict(park)*100
test_choi = model.predict(choi)*100

print("Kim  당뇨병 가능성 예측 : %.2f" %test_kim, "%")
print("Park 당뇨병 가능성 예측 : %.2f" %test_park, "%")
print("Choi 당뇨병 가능성 예측 : %.2f" %test_choi, "%")
```

```
1/1 [==============================] - 0s 135ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 24ms/step
Kim  당뇨병 가능성 예측 : 32.54 %
Park 당뇨병 가능성 예측 : 89.56 %
Choi 당뇨병 가능성 예측 : 46.31 %
```

11. 딥러닝 모델 저장하기

In [29]:
```
model.save('DL_RESULT/Diabetes.h5')
```