딥러닝 모델 구현해 보기

학습 내용

- 정신 건강 설문 조사 데이터를 사용하여 개인의 우울증을 경험할 수 있는 요인을 탐색

- 첫번째 데이터 셋 : 자전거 공유 업체 시간대별 데이터
- 두번째 데이터 셋 : 타이타닉 데이터 셋(PyTorch)
- **세번째 데이터 셋 : 정신 건강 설문 조사 데이터**

목차

01. 사전 환경 설치

목차로 이동하기

GPU 버전 PyTorch 설치

```
# 가상환경 만들기
conda create --name gpuDL python=3.10

# Jupyter Notebook
conda install -c conda-forge notebook jupyter

# 추가 설치
pip install pandas scikit-learn

# PyTorch 설치하기
# 01 Anaconda 사용 시
conda install pytorch=2.5.1 torchvision torchaudio pytorch-
cuda=12.1 -c pytorch -c nvidia

# 02 pip 사용 시
pip3 install torch==2.5.1 torchvision torchaudio --index-url
https://download.pytorch.org/whl/cu121
```

In [1]:

```python
import torch
import sys
import numpy
import torch
```

```
print("Python version:", sys.version)
print("NumPy version:", numpy.__version__)
print("PyTorch version:", torch.__version__)

# CUDA 사용 가능 여부 확인
print(torch.cuda.is_available())

# 사용 가능한 GPU 장치 수 확인
print(torch.cuda.device_count())
```

```
Python version: 3.11.10 | packaged by Anaconda, Inc. | (main, Oct  3 2024, 07:22:
26) [MSC v.1929 64 bit (AMD64)]
NumPy version: 2.1.3
PyTorch version: 2.5.1+cpu
False
0
```

PyTorch 버전	권장 CUDA 버전
1.13.x	11.7
2.0.x	11.8
2.1.x --	12.1

In [2]:
```
import torch
import sklearn

print(torch.__version__)
print(sklearn.__version__)
```

```
2.5.1+cpu
1.5.2
```

02. 라이브러리 및 데이터 불러오기

목차로 이동하기

In [3]:
```
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
```

In [4]:
```
import os
os.getcwd()
```

Out[4]: 'd:\\github\\DeepLearning_Basic_Class'

In [5]:
```
# 시드 고정
torch.manual_seed(42)
np.random.seed(42)

# 1. 데이터 준비
# 상대 경로로 데이터 로드
```

```python
train = pd.read_csv('./datasets/health_mental_24/train.csv')
test = pd.read_csv('./datasets/health_mental_24/test.csv')
sub = pd.read_csv('./datasets/health_mental_24/sample_submission.csv')

# 데이터 shape 확인
print("훈련 데이터 shape:", train.shape)
print("테스트 데이터 shape:", test.shape)

# 데이터 정보 확인
print("\n훈련 데이터 정보:")
print(train.info())

print("\n테스트 데이터 정보:")
print(test.info())
```

```
훈련 데이터 shape: (140700, 20)
테스트 데이터 shape: (93800, 19)

훈련 데이터 정보:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 140700 entries, 0 to 140699
Data columns (total 20 columns):
 #   Column                                 Non-Null Count   Dtype  
---  ------                                 --------------   -----  
 0   id                                     140700 non-null  int64  
 1   Name                                   140700 non-null  object 
 2   Gender                                 140700 non-null  object 
 3   Age                                    140700 non-null  float64
 4   City                                   140700 non-null  object 
 5   Working Professional or Student        140700 non-null  object 
 6   Profession                             104070 non-null  object 
 7   Academic Pressure                      27897 non-null   float64
 8   Work Pressure                          112782 non-null  float64
 9   CGPA                                   27898 non-null   float64
 10  Study Satisfaction                     27897 non-null   float64
 11  Job Satisfaction                       112790 non-null  float64
 12  Sleep Duration                         140700 non-null  object 
 13  Dietary Habits                         140696 non-null  object 
 14  Degree                                 140698 non-null  object 
 15  Have you ever had suicidal thoughts ?  140700 non-null  object 
 16  Work/Study Hours                       140700 non-null  float64
 17  Financial Stress                       140696 non-null  float64
 18  Family History of Mental Illness       140700 non-null  object 
 19  Depression                             140700 non-null  int64  
dtypes: float64(8), int64(2), object(10)
memory usage: 21.5+ MB
None

테스트 데이터 정보:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 93800 entries, 0 to 93799
Data columns (total 19 columns):
 #   Column                                 Non-Null Count  Dtype  
---  ------                                 --------------  -----  
 0   id                                     93800 non-null  int64  
 1   Name                                   93800 non-null  object 
 2   Gender                                 93800 non-null  object 
 3   Age                                    93800 non-null  float64
 4   City                                   93800 non-null  object 
 5   Working Professional or Student        93800 non-null  object 
 6   Profession                             69168 non-null  object 
 7   Academic Pressure                      18767 non-null  float64
 8   Work Pressure                          75022 non-null  float64
 9   CGPA                                   18766 non-null  float64
 10  Study Satisfaction                     18767 non-null  float64
 11  Job Satisfaction                       75026 non-null  float64
 12  Sleep Duration                         93800 non-null  object 
 13  Dietary Habits                         93795 non-null  object 
 14  Degree                                 93798 non-null  object 
 15  Have you ever had suicidal thoughts ?  93800 non-null  object 
 16  Work/Study Hours                       93800 non-null  float64
 17  Financial Stress                       93800 non-null  float64
 18  Family History of Mental Illness       93800 non-null  object 
dtypes: float64(8), int64(1), object(10)
```

```
memory usage: 13.6+ MB
None
```

In [6]:
```python
train.head()
```

Out[6]:

	id	Name	Gender	Age	City	Working Professional or Student	Profession	Academic Pressure	Pre
0	0	Aaradhya	Female	49.0	Ludhiana	Working Professional	Chef	NaN	
1	1	Vivan	Male	26.0	Varanasi	Working Professional	Teacher	NaN	
2	2	Yuvraj	Male	33.0	Visakhapatnam	Student	NaN	5.0	
3	3	Yuvraj	Male	22.0	Mumbai	Working Professional	Teacher	NaN	
4	4	Rhea	Female	30.0	Kanpur	Working Professional	Business Analyst	NaN	

In [7]:
```python
test.head(3)
```

Out[7]:

	id	Name	Gender	Age	City	Working Professional or Student	Profession	Academic Pressure
0	140700	Shivam	Male	53.0	Visakhapatnam	Working Professional	Judge	NaN
1	140701	Sanya	Female	58.0	Kolkata	Working Professional	Educational Consultant	NaN
2	140702	Yash	Male	53.0	Jaipur	Working Professional	Teacher	NaN

In [8]:
```python
sub.head()
```

Out[8]:

	id	Depression
0	140700	0
1	140701	0
2	140702	0
3	140703	0
4	140704	0

In [9]:

```
train.columns
```

Out[9]:

```
Index(['id', 'Name', 'Gender', 'Age', 'City',
       'Working Professional or Student', 'Profession', 'Academic Pressure',
       'Work Pressure', 'CGPA', 'Study Satisfaction', 'Job Satisfaction',
       'Sleep Duration', 'Dietary Habits', 'Degree',
       'Have you ever had suicidal thoughts ?', 'Work/Study Hours',
       'Financial Stress', 'Family History of Mental Illness', 'Depression'],
      dtype='object')
```

In [40]:

```
import numpy as np
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder, StandardScaler

# 필요한 피처 선택
features = [
    'Age', 'Work/Study Hours', 'Financial Stress',  # 숫자형 변수
    'Gender', 'Working Professional or Student', 'City',
    'Sleep Duration', 'Dietary Habits', 'Degree',  # 범주형 변수
    'Academic Pressure', 'Work Pressure', 'CGPA',
    'Study Satisfaction', 'Job Satisfaction'  # 결측치 있는 숫자형 변수
]
target = 'Depression'
```

In [42]:

```
# 전처리 함수 정의
def preprocess_data(df, gubun='train'):
    # 데이터프레임 복사
    data = df.copy()

    # 범주형 변수 인코딩
    categorical_features = [
        'Gender',
        'Working Professional or Student',
        'City',
        'Sleep Duration',
        'Dietary Habits',
        'Degree'
    ]

    # 라벨 인코더 초기화
    le = LabelEncoder()

    # 범주형 변수 인코딩
    for col in categorical_features:
        # NaN 값을 문자열로 변환 후 인코딩
        data[col] = data[col].fillna('Unknown')
```

```python
        data[col] = le.fit_transform(data[col].astype(str))

    # 숫자형 변수 분리
    numeric_features = [
        'Age', 'Work/Study Hours', 'Financial Stress',
        'Academic Pressure', 'Work Pressure', 'CGPA',
        'Study Satisfaction', 'Job Satisfaction'
    ]

    # 결측값 처리
    imputer = SimpleImputer(strategy='median')

    # 숫자형 변수 결측값 대체
    numeric_data = imputer.fit_transform(data[numeric_features])

    # 스케일러로 숫자형 변수 표준화
    scaler = StandardScaler()
    scaled_numeric_data = scaler.fit_transform(numeric_data)

    # 범주형 변수와 숫자형 변수 결합
    X = np.column_stack([
        scaled_numeric_data,  # 표준화된 숫자형 변수
        data[categorical_features].values  # 인코딩된 범주형 변수
    ])

    if gubun=="train":
        # 타겟 변수 추출
        y = data[target].values
        return X, y
    else:
        return X

# 전처리 적용
X, y = preprocess_data(train)
```

In [43]:

```python
# 스케일링
scaler = StandardScaler()
X = scaler.fit_transform(X)

# 데이터 분할
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# NumPy to PyTorch Tensor 변환
# X_train이라는 NumPy 배열을 PyTorch의 FloatTensor로 변환.
# FloatTensor는 32비트 부동 소수점 숫자로 구성된 텐서를 생성.
# 이 변환은 PyTorch 모델에서 데이터를 처리할 수 있도록 준비하는 단계
X_train = torch.FloatTensor(X_train)
X_test = torch.FloatTensor(X_test)
y_train = torch.FloatTensor(y_train).unsqueeze(1)
y_test = torch.FloatTensor(y_test).unsqueeze(1)
```

03. 신경망 모델 정의

목차로 이동하기

- 딥러닝의 이해를 위해 일부 특징(변수)만 지정하였음.
- 이미지를 사용할 때는 지정된 이미지 전체를 입력 데이터로 사용하는 경우가 대부분.

In [32]:
```
# 2. 신경망 모델 정의
model = nn.Sequential(
    nn.Linear(14, 32),
    nn.ReLU(),
    nn.Dropout(0.3),
    nn.Linear(32, 16),
    nn.ReLU(),
    nn.Dropout(0.3),
    nn.Linear(16, 1),
    nn.Sigmoid()
)

# 3. 모델 학습
# 손실 함수와 옵티마이저 정의
criterion = nn.BCELoss()  # 이진 분류 손실 함수
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

In [35]:
```
# 학습 진행
epochs = 1000
for epoch in range(epochs):
    # 순전파
    outputs = model(X_train)
    loss = criterion(outputs, y_train)

    # 정확도 계산
    with torch.no_grad():
        # 이진 분류의 경우
        predicted = (outputs > 0.5).float()
        accuracy = (predicted == y_train).float().mean()


    # 역전파
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    # 20번마다 손실과 정확도 출력
    if (epoch + 1) % 20 == 0:
        print(f'Epoch [{epoch+1}/{epochs}], '
              f'Loss: {loss.item():.4f}, '
              f'Accuracy: {accuracy.item():.4f}')
```

```
Epoch [20/1000], Loss: 0.1903, Accuracy: 0.9216
Epoch [40/1000], Loss: 0.1895, Accuracy: 0.9219
Epoch [60/1000], Loss: 0.1891, Accuracy: 0.9217
Epoch [80/1000], Loss: 0.1887, Accuracy: 0.9216
Epoch [100/1000], Loss: 0.1885, Accuracy: 0.9218
Epoch [120/1000], Loss: 0.1883, Accuracy: 0.9219
Epoch [140/1000], Loss: 0.1882, Accuracy: 0.9218
Epoch [160/1000], Loss: 0.1880, Accuracy: 0.9219
Epoch [180/1000], Loss: 0.1879, Accuracy: 0.9219
Epoch [200/1000], Loss: 0.1878, Accuracy: 0.9221
Epoch [220/1000], Loss: 0.1877, Accuracy: 0.9222
Epoch [240/1000], Loss: 0.1875, Accuracy: 0.9222
Epoch [260/1000], Loss: 0.1874, Accuracy: 0.9223
Epoch [280/1000], Loss: 0.1873, Accuracy: 0.9223
Epoch [300/1000], Loss: 0.1872, Accuracy: 0.9224
Epoch [320/1000], Loss: 0.1872, Accuracy: 0.9224
Epoch [340/1000], Loss: 0.1871, Accuracy: 0.9224
Epoch [360/1000], Loss: 0.1870, Accuracy: 0.9224
Epoch [380/1000], Loss: 0.1869, Accuracy: 0.9225
Epoch [400/1000], Loss: 0.1868, Accuracy: 0.9224
Epoch [420/1000], Loss: 0.1868, Accuracy: 0.9224
Epoch [440/1000], Loss: 0.1867, Accuracy: 0.9225
Epoch [460/1000], Loss: 0.1866, Accuracy: 0.9225
Epoch [480/1000], Loss: 0.1865, Accuracy: 0.9225
Epoch [500/1000], Loss: 0.1865, Accuracy: 0.9225
Epoch [520/1000], Loss: 0.1864, Accuracy: 0.9225
Epoch [540/1000], Loss: 0.1863, Accuracy: 0.9226
Epoch [560/1000], Loss: 0.1862, Accuracy: 0.9226
Epoch [580/1000], Loss: 0.1862, Accuracy: 0.9226
Epoch [600/1000], Loss: 0.1861, Accuracy: 0.9226
Epoch [620/1000], Loss: 0.1860, Accuracy: 0.9226
Epoch [640/1000], Loss: 0.1860, Accuracy: 0.9226
Epoch [660/1000], Loss: 0.1859, Accuracy: 0.9228
Epoch [680/1000], Loss: 0.1859, Accuracy: 0.9228
Epoch [700/1000], Loss: 0.1858, Accuracy: 0.9228
Epoch [720/1000], Loss: 0.1858, Accuracy: 0.9228
Epoch [740/1000], Loss: 0.1857, Accuracy: 0.9229
Epoch [760/1000], Loss: 0.1857, Accuracy: 0.9228
Epoch [780/1000], Loss: 0.1856, Accuracy: 0.9228
Epoch [800/1000], Loss: 0.1856, Accuracy: 0.9229
Epoch [820/1000], Loss: 0.1855, Accuracy: 0.9230
Epoch [840/1000], Loss: 0.1855, Accuracy: 0.9230
Epoch [860/1000], Loss: 0.1854, Accuracy: 0.9231
Epoch [880/1000], Loss: 0.1854, Accuracy: 0.9231
Epoch [900/1000], Loss: 0.1853, Accuracy: 0.9231
Epoch [920/1000], Loss: 0.1853, Accuracy: 0.9232
Epoch [940/1000], Loss: 0.1853, Accuracy: 0.9231
Epoch [960/1000], Loss: 0.1852, Accuracy: 0.9231
Epoch [980/1000], Loss: 0.1852, Accuracy: 0.9231
Epoch [1000/1000], Loss: 0.1851, Accuracy: 0.9231
```

In [37]:

```python
# 4. 모델 평가
model.eval()  # 평가 모드
with torch.no_grad():
    test_outputs = model(X_test)
    predicted = (test_outputs > 0.5).float()
    accuracy = (predicted == y_test).float().mean()
    print(f'Test Accuracy: {accuracy.item():.4f}')
```

```
Test Accuracy: 0.9231
```

04. 새로운 데이터로 예측

목차로 이동하기

In [38]:
```
# 필요한 피처 선택
features = [
    'Age', 'Work/Study Hours', 'Financial Stress',  # 숫자형 변수
    'Gender', 'Working Professional or Student', 'City',
    'Sleep Duration', 'Dietary Habits', 'Degree',  # 범주형 변수
    'Academic Pressure', 'Work Pressure', 'CGPA',
    'Study Satisfaction', 'Job Satisfaction'  # 결측치 있는 숫자형 변수
]

# 선택한 피처로 test 데이터 준비
X_predict = preprocess_data(test, gubun='test')

# 결측값 처리
X_predict = scaler.transform(X_predict)

# 스케일링
X_predict = scaler.transform(X_predict)

# PyTorch Tensor로 변환
X_predict = torch.FloatTensor(X_predict)

# 예측 수행
model.eval()
with torch.no_grad():
    pred = model(X_predict)
    pred = (pred > 0.5).float()

# submission 파일에 예측값 저장
sub['Depression'] = pred.numpy()

sub.to_csv('./datasets/health_mental_24/sub03.csv', index=False)
print("예측 완료 및 sub 파일 저장.")
```

```
예측 완료 및 sub 파일 저장.
```

In [ ]:
```
```