

```

import pandas as pd
from google.colab import drive
drive.mount("/content/gdrive", force_remount=True)
filepath = "/content/gdrive/My Drive/csv/"
df_train = pd.read_csv(filepath + 'train.csv')
df_test = pd.read_csv(filepath + 'test.csv')
print(df_train.head(3))
print(df_test.head(3))

```

처음 블록에는 구글 드라이브에 csv파일을 넣어놓고 불러오는 코드와 이 파일들의 첫 3개의 값들을 확인 하기 위해 head() 메소드를 이용해서 출력했다.

```

[ ] from torch.utils.data import Dataset
# Define a custom dataset for Titanic data
class SpaceShipTitanicDataset(Dataset):
    def __init__(self, data, labels=None, mode='train'): #생성자 함수
        self.data = data
        self.labels = labels
        self.mode = mode
    def __len__(self): #data의 길이 1000이면 0~999까지
        return len(self.data)
    def __getitem__(self, idx):
        if self.mode == 'train':
            x = self.data[idx]
            y = self.labels[idx]
            return x, y
        else: # test모듈은 x가 없기 때문에 x를 만들어서 반환
            x = self.data[idx]
            return x

```

이 블록에서는 스페이스 쉽 타이타닉의 데이터 셋 클래스를 만들었다.

예전에 했던 타이타닉의 데이터셋을 그대로 가져왔는데 __getitem 메소드로 x,y값을 가져오고 train과 test를 나눠서 test 데이터에는 y값이 없기 때문에 x값만 따로 가져올수 있게 만들었다.

```

import numpy as np
import torch
from torch import nn
import torch.nn.functional as F
def get_data_from_df(df):
    x = np.zeros((len(df), 4))
    # HomePlanet
    df["HomePlanet"] = df["HomePlanet"].fillna("Earth") # fillna는 비어있는 값을 어떤 값으로 채울것인지 해주는 함수
    x[:, 0] = df["HomePlanet"].map( {"Earth": 0, "Europa": 1, "Mars": 2} ).astype(int)
    # CryoSleep
    df["CryoSleep"] = df["CryoSleep"].fillna(True)
    x[:, 1] = df["CryoSleep"].astype(int)
    # VIP
    df["VIP"] = df["VIP"].fillna(False)
    x[:, 2] = df["VIP"].astype(int)
    # AGE
    df["Age"] = df["Age"].fillna(df["Age"].mean())
    x[:, 3] = (df["Age"] - df["Age"].mean()) / df["Age"].std() # 정규화 진행하면 평균=1, 표준편차 0이됨.
    return x

```

중요도로 따지면 이부분이 전처리영역으로 데이터에서 어떤 피처를 기준으로 Transported의 True, False를 판별할지를 정하는 영역이라고 할 수있다. 글쓴이는 여러개의 피처중 Homeplanet과 CryoSleep, VIP, Age 4개의 피처를 선택했다. 먼저 HomePlanet을 선택한 이유는 Homeplanet은 출발하는 사람이 사는 행성을 나타내는 피처인데 피처에서 대표되는 대표값이 3개 뿐이라 True, False를 구분하기에 적절하다고 판단했고 Earth의 수가 제일 많

아서 NaN값들은 전부 Earth로 채우고 숫자가 많은 순서대로 0, 1, 2 순서대로 넣었다. 여기서 Earth, Europa, Mars 순이었다.

두번째 피쳐는 CryoSleep을 선택했다. 이건 냉동수면을 나타내는 피쳐였는데 선택하게 된 계기는 뭔가 냉동수면을 하면 건강한 상태로 Transported가 될거같다는 생각에 넣었다. 그리고 NaN값은 True가 False보다 많아서 True로 선택해서 채워줬다.

세번째 피쳐로는 VIP를 골랐는데 뭔가 VIP이면 이동을 잘할거라 생각했다. 하지만 표를 분석해보니 VIP이면 오히려 이동을 잘 못하였고 아닌 사람들의 비율이 훨씬 많았다! 그래서 NaN값을 False로 채웠다.

네번째 피쳐로는 Age를 골랐고 정규화를 해서 나이가 평균에 가까울수록 많이 transported를 했을거라고 생각하고 고르게 되었고 그래서 NaN값은 평균으로 골랐다.

```
[ ] x_train = get_data_from_df(df_train)
    y_train = df_train["Transported"].values.reshape(-1, 1)
    x_test = get_data_from_df(df_test)

from torch.utils.data import DataLoader
train_dataset = SpaceShipTitanicDataset(data=x_train, labels=y_train, mode='train')
test_dataset = SpaceShipTitanicDataset(data=x_test, mode='test')
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True) # batch_size dataset하나당 몇개씩 받을것인지, shuffle은 순서를 섞을
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)

[ ] import torch.nn as nn
    # Define the neural network architecture
    class TitanicNet(nn.Module):
        def __init__(self, input_size, hidden_size, output_size):
            super(TitanicNet, self).__init__()
            self.fc1 = nn.Linear(input_size, hidden_size)
            self.relu = nn.ReLU()
            self.fc2 = nn.Linear(hidden_size, output_size)
            self.sigmoid = nn.Sigmoid()
        def forward(self, x):
            out = self.fc1(x)
            out = self.relu(out)
            out = self.fc2(out)
            out = self.sigmoid(out)
            return out
```

train일때는 x는 전처리 후 리 가져오고 y는 transported값만 가져와서 아래의 TitanicNet로 돌려서 예상과 비슷한지 비교했다. 그리고 test는 x값만 존재하니 x_test로 전처리 후 그대로 가져왔다.

Dataset은 이제 위에 만들었던 데이터셋을 조종할수 있는 SpaceShipTitanicDataset class를 이용해서 추후에 우리가 만든 네트워크로 값들을 이용해서 Transported값을 예측하기 위함이다! 그리고 DataLoader에서 batch_size를 설정해서 한번에 돌릴 데이터값의 양을 정했고 더 규칙이 없는 상황을 만들기 위해 Shuffle을 했다!

네트워크 레이어는 두개를 넣었고 원래의 타이타닉의 네트워크를 그대로 가져왔다!

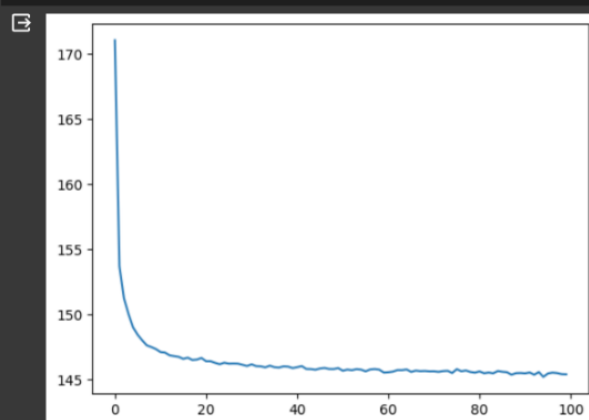
```
[ ] import torch
import torch.optim as optim
# Initialize the model
input_size = len(x_train[0])
hidden_size = 64
output_size = 1 # class 개수
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = TitanicNet(input_size, hidden_size, output_size)
model = model.to(device)
# Define the loss function and optimizer
criterion = nn.BCELoss() # binary cross entropy
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

여기서 loss function은 True, False를 고르는 이진분류 문제이기 때문에 BCELoss(Binary Cross Entropy)를 사용했다! 그리고 Optimizer는 제일 많이 사용하는 Adam optimizer로 선택했고 Learning rate는 10의 -3을 지정했다.

```
▶ # Train the model
num_epochs = 100
total_correct, total_samples = 0, 0
losses = []
model.train()
for epoch in range(num_epochs):
    epoch_loss = 0
    for inputs, labels in train_loader:
        inputs = inputs.float().to(device)
        labels = labels.float().to(device)
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        epoch_loss += loss.item()
        predicted = torch.round(outputs) # 0.5 이상이면 살았다 0.5 미만이면 죽었다.
        total_correct += (predicted == labels).sum().item()
        total_samples += labels.size(0)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step() # 업데이트 아담방식으로
    losses.append(epoch_loss)
accuracy = total_correct / total_samples
print(f"Epoch : {epoch: 2d}, Accuracy: {accuracy: .6f}")
```

여기서는 우리가 원하는 파라미터를 도출해내기 위한 실행문이고 Epoch(반복횟수)는 100번 정도로 해서 해봤다. 이렇게 loss값을 계산해서 정확도를 계산해보니 73정도가 나왔다.

```
▶ import matplotlib.pyplot as plt
losses = np.array(losses)
plt.plot(losses)
plt.show()
```




이렇게 loss graph를 그려보면 로스 값이 올바른 모양, 즉 $-\log$ 형태로 아름답게(?) 나오는것을 확인할 수 있다.

```
[ ] # Evaluate the model
total_pred = []
model.eval()
with torch.no_grad(): # 그래디언트 계산 안한다.
    for inputs in test_loader:
        inputs = inputs.float().to(device)
        outputs = model(inputs)
        predicted = torch.round(outputs)
        total_pred += predicted.bool().tolist()

[ ] total_pred = np.array(total_pred).flatten()
# Save the predictions to a CSV file
df_test['Transported'] = total_pred
df_test[['PassengerId', 'Transported']].to_csv('submission.csv', index=False)
```

이렇게 train데이터를 test해봤기때문에 이제 제출한 test셋을 만들어서 제출했다!

Submissions

Submission and Description		Public Score ^①
<div><div>All</div><div>Successful</div><div>Errors</div></div> <div> submission (7).csv Complete · 8d ago</div>		0.73906

최종적으로 73퍼센트가 나오는걸로 마무리했다!

소감: 직접 전처리해서 코드를 짜는걸 해보니까 직접 전처리를 하려고 표의 값을 분석하는 곳에서 코딩에서는 느낄수 없는 새로운 경험을 해볼수 있었다. 데이터분석에서 규칙을 찾는 과정이 마치 코딩에서 알고리즘을 구하는 과정과 유사해서 흥미로웠다. 정확히 알지 못하지만 맛보기로 만들어보니까 나름 할만했다!