Python Final Project : part1

# 버섯 모양에 따른 유독성 판단과
# 각각의 요소들이 유독성과 가지는 관계를 선형적 관계로 나타내기.

20241685 안현정, 20241693 이서희

## Summary of Questions and results

**연구 질문**

버섯의 독성 여부 판단하기

: 버섯의 특징만을 가지고 독성여부를 판단할 수 있을 지에 대한 의문점이 생겼다. 그래서 이를 가지고 독성을 판단하는 분류 프로그램을 만든다.

버섯 종류 판정하기

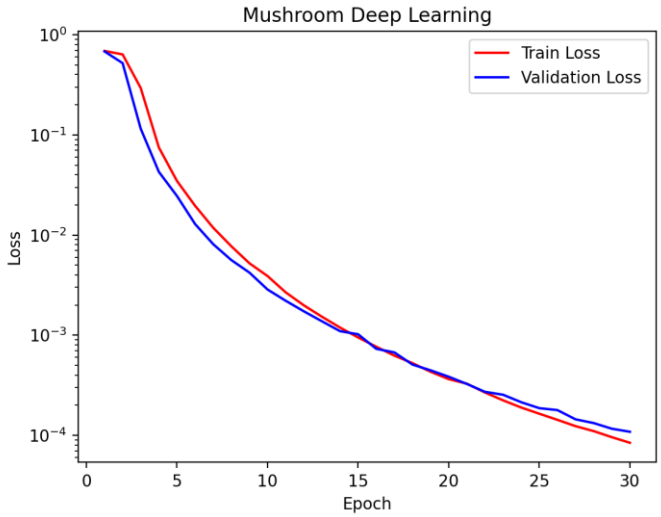
: 버섯 특징 데이터를 가지고 전부 다는 아니더라도, 특정 몇 종류의 판정이 가능한 분류 프로그램을 만든다.

(수정사항) 몇몇 특징과 독성을 가지는 것의 여부의 상관관계를 파악하기

: 버섯의 모양별로 버섯의 독성 여부에 얼만큼의 상관관계를 가지는 지 파악하고 이를 그래프로 나타내어 직관적인 판단이 가능한 지표를 만든다.

수정을 진행한 이유는, 고작 몇 개 종류의 판정으로는 실용성이 매우 떨어지고 이 프로젝트를 하는 목적과도 맞지 않기 때문에, 수업시간에 배웠던 그래프를 이용해서 활동을 진행하는 연구 질문으로 수정하게 되었다.

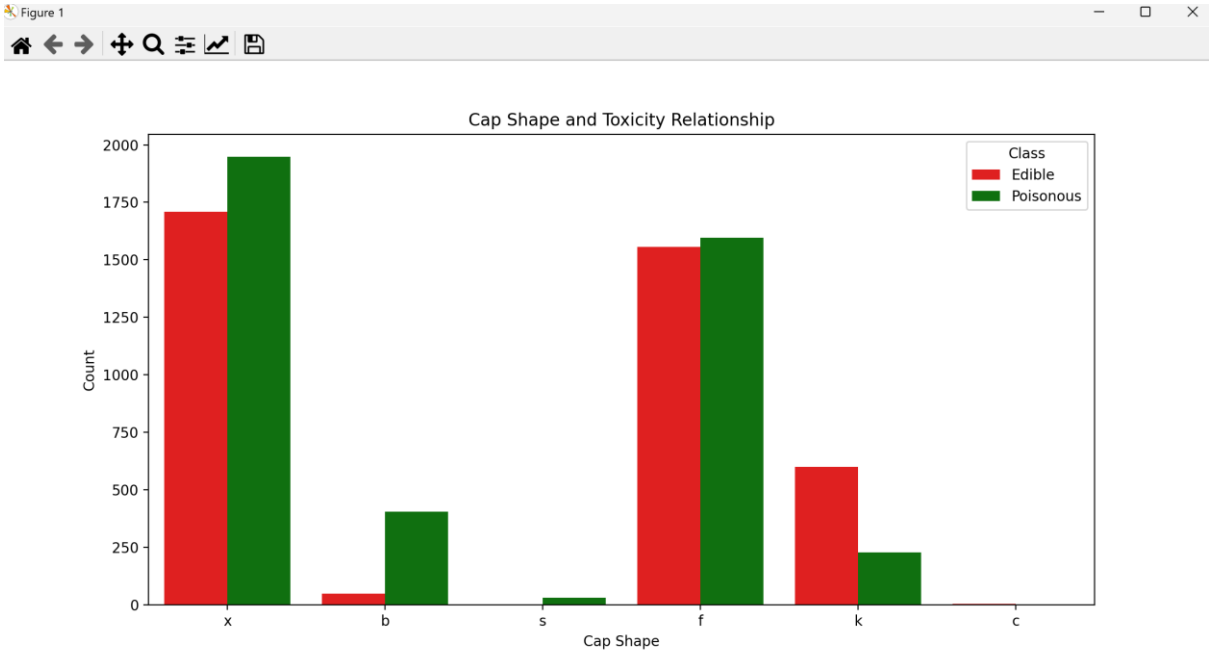
**결과**

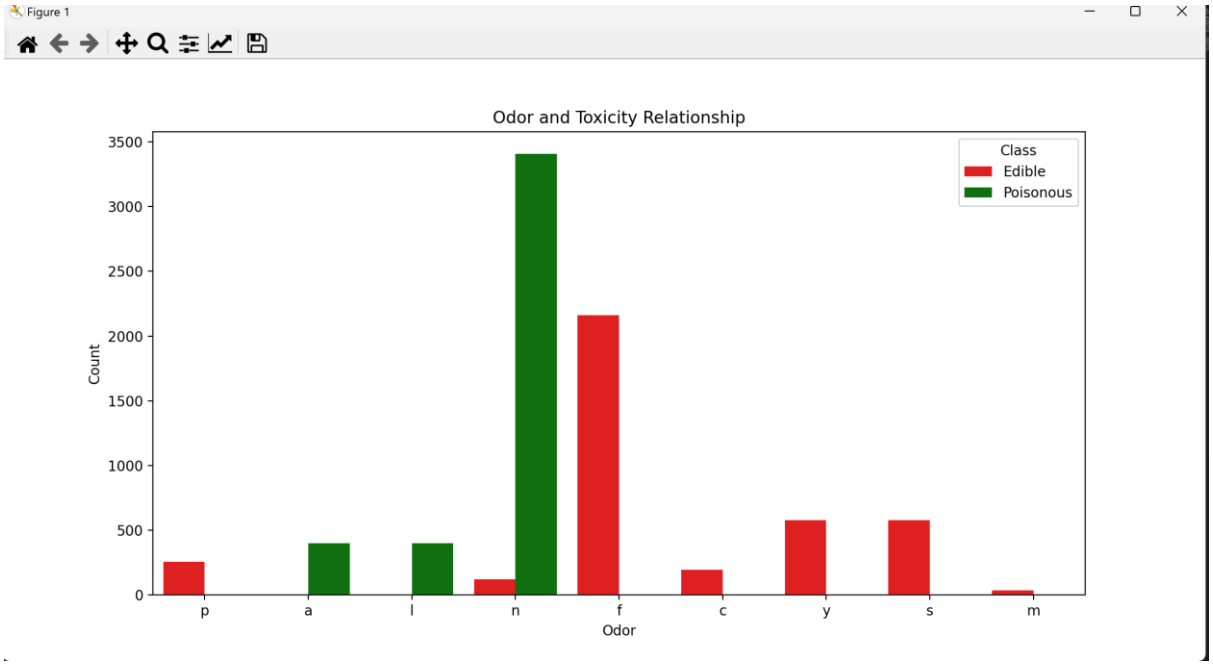



```
Epoch 0: Train Loss: 0.6899, Validation Loss: 0.6822
Epoch 1: Train Loss: 0.6369, Validation Loss: 0.5212
Epoch 2: Train Loss: 0.2938, Validation Loss: 0.1153
Epoch 3: Train Loss: 0.0750, Validation Loss: 0.0428
Epoch 4: Train Loss: 0.0347, Validation Loss: 0.0246
Epoch 5: Train Loss: 0.0196, Validation Loss: 0.0129
Epoch 6: Train Loss: 0.0118, Validation Loss: 0.0081
Epoch 7: Train Loss: 0.0077, Validation Loss: 0.0056
Epoch 8: Train Loss: 0.0052, Validation Loss: 0.0042
Epoch 9: Train Loss: 0.0039, Validation Loss: 0.0029
Epoch 10: Train Loss: 0.0027, Validation Loss: 0.0022
Epoch 11: Train Loss: 0.0020, Validation Loss: 0.0017
Epoch 12: Train Loss: 0.0015, Validation Loss: 0.0014
Epoch 13: Train Loss: 0.0012, Validation Loss: 0.0011
Epoch 14: Train Loss: 0.0009, Validation Loss: 0.0010
Epoch 15: Train Loss: 0.0008, Validation Loss: 0.0007
Epoch 16: Train Loss: 0.0006, Validation Loss: 0.0007
Epoch 17: Train Loss: 0.0005, Validation Loss: 0.0005
Epoch 18: Train Loss: 0.0004, Validation Loss: 0.0004
Epoch 19: Train Loss: 0.0004, Validation Loss: 0.0004
Epoch 20: Train Loss: 0.0003, Validation Loss: 0.0003
Epoch 21: Train Loss: 0.0003, Validation Loss: 0.0003
Epoch 22: Train Loss: 0.0002, Validation Loss: 0.0003
Epoch 23: Train Loss: 0.0002, Validation Loss: 0.0002
Epoch 24: Train Loss: 0.0002, Validation Loss: 0.0002
Epoch 25: Train Loss: 0.0001, Validation Loss: 0.0002
Epoch 26: Train Loss: 0.0001, Validation Loss: 0.0001
Epoch 27: Train Loss: 0.0001, Validation Loss: 0.0001
Epoch 28: Train Loss: 0.0001, Validation Loss: 0.0001
Epoch 29: Train Loss: 0.0001, Validation Loss: 0.0001
```

계속해서 train 과 test값의 오차가 줄어들도록 학습시킨 결과로 이런 값이 나왔다.

또다른 도전목표와 관련하여 특정 칼럼과 독성 유무의 상관관계를 그래프로 표현하는 도전목표의 결과값은 이와 같다.




1) cap-shape - Toxic
2) Odor - Toxic

## Motivation

1. 독버섯으로 인한 피해 감소
   : 섭취, 접촉 시 중독으로 인한 생명에 유해한 영향을 끼치는 독성 버섯을 식별하여 사람들의 건강과 생명을 보호할 수 있게 하기 위함이다.

2. 머신러닝과 관련된 지식 확보

: 수업시간에 배운 머신러닝과 Decision Tree와 관련된 것들을 더 깊이 탐색해보고 최근에 배우고 있는 딥러닝과 관련된 부분으로도 더 학습해보고 싶어서 이런 주제를 선택하게 되었다. 버섯의 독성분류와 관련된 것들은 머신러닝을 공부하기에 가장 기본적인 주제이고, 이미 앞서 많은 사람들이 이것을 주제로 진행해둔 프로젝트가 존재한다. 하지만, 이것이 항상 같은 결과를 도출하는 것이 아니기 때문에, 이를 주제로 선택하게 되었다.

## Dataset

[ https://www.kaggle.com/datasets/uciml/mushroom-classification ]

Kaggle 사이트의 mushroom classification 데이터를 사용함.

총 8156개의 버섯 데이터를 가지고 있으며, 캡 모양, 캡 표면, 캡 색상, 냄새, 줄기 크기, 줄기 모양, 줄기 표면, 줄기표면 아래 고리 등등의 특징을 칼럼으로 가지고, 관련된 세부사항들을 각각을 상징하는 알파벳을 정의하여 알파벳을 칼럼 값으로 하도록 이루어져 있다.

## Method

1. 다른 것들을 본격적으로 수행하기 전 데이터를 확인해주었다. NAN값의 유무와 관련하여 전처리를 진행할 것이 있는지, isnull() 메서드를 사용하여 확인했다.

2.

## Results

## Impact and Limitations

파트 0에서 과제의 주제를 '버섯 모양에 따른 유독성 판단과 각각의 요소들이 유독성과 가지는 관계를 선형적으로 표현하기'로 정한 이유는 비교적 쉬운 주제로 우리가 머신 러닝과 파이썬 코딩을 직접 공부하며 과제를 해결하기 위함이였다. 그러나 시간이 생각보다 부족했고 둘의 노력으로는 여러 힘듦과 어려움이 있었다.

연구적인 영향은 잠재적으로 독버섯 식별에 도움을 줄 수 있다는 점에서 중요하다. 이는 특히 야외 활동 시 안전을 위해 유용할 수 있다. 그러나 데이터셋이 특정 지역의 버섯만을 포함하고 있어 일반화의 한계가 있을 수 있다. 데이터의 편향이 결과에 영향을 미칠

가능성도 있으며, 이를 보완하기 위해 다양한 지역의 버섯 데이터를 추가로 수집할 필요가 있다.

## Challenge Goals

1. 머신 러닝, 딥러닝

   : 버섯 데이터를 바탕으로 여러가지 기준을 설정하여 분류한 후 머신 러닝을 수행할 수 있도록 한다. 기준에 대한 설정을 명확하게 해야만 정확한 분류가 가능하다.

- **활성화 함수 (Activation Function)**: ReLU 와 Sigmoid 활성화 함수를 사용합니다. ReLU 는 비선형성을 추가하고, Sigmoid 는 마지막 출력 값을 0 과 1 사이로 만듭니다.
- **순전파 (Forward Propagation)**: 입력 데이터를 각 층을 통과시키면서 변환합니다. 이 과정에서 ReLU와 Sigmoid 활성화 함수를 적용합니다.

딥러닝과 관련된 부분에서 학습한 개념들이다. 이 개념을 바로 알고 적용하기에 어려움이 있어, 다른 사람들이 짠 코드와 우리가 하려고 하는 것을 비교해가며 끼워맞추기 식으로 코드를 작성했던 것 같다.

2. 새로운 라이브러리 공부

   : 분류작업에 알맞은 라이브러리를 찾고 이를 적용하기 위해서 공부한다.

## Work Plan Evaluation

1) 데이터 수집 (3시간 소요 예정)
2) 머신 러닝 알고리즘 공부 및 구현 (3일 소요 예정)
3) 프레젠테이션 자료 제작 (5시간 소요 예정)
4) 발표 준비 (2시간 소요 예정)

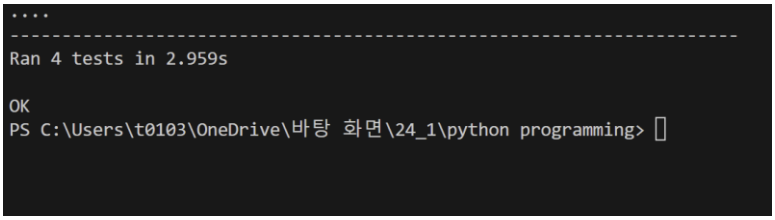
라는 계획을 세웠습니다. 하지만 데이터셋이 하나만으로 해결되어 1시간 이내로 시간이 소요되었고 대신 머신러닝에 대한 전반적인 지식을 쌓는 것이 많은 시간이 걸렸습니다. 뿐만 아니라 진행한 프로젝트가 딥러닝을 활용했다는 것에서 파이썬 프로그래밍 수업 14주차 내용과 일부 겹친다는 것을 확인할 수 있습니다. 수업 내용 뿐만 아니라 여러가지 부분에서의 지식을 충분히 습득하는데에 7일 이상의 시간이 소모되었습니다.

나머지 단계는 part1 이후이므로 계획대로 수행할 예정입니다

# Testing

```python
import unittest
import pandas as pd
import torch
from torch.utils.data import DataLoader
from coding import CustomDataset, NeuralNetwork
from sklearn.model_selection import train_test_split

class TestMushroomClassification(unittest.TestCase):

    def setUp(self):
        # 데이터 로드 및 전처리
        data = pd.read_csv("PythonFinalProject/mushrooms.csv")
        self.features = data.drop(["class"], axis=1)
        self.labels = data.loc[:, "class"]
        self.features = pd.get_dummies(self.features)
        self.labels = pd.get_dummies(self.labels)
        self.feature_count = len(self.features.columns)
        self.label_count = len(self.labels.columns)

        # 데이터셋 분할
        x_train, x_valid, y_train, y_valid = train_test_split(self.features, self.labels, test_size=0.4, stratify=self.la
        x_valid, x_test, y_valid, y_test = train_test_split(x_valid, y_valid, test_size=0.5, stratify=y_valid, random_sta

        self.train_dataset = CustomDataset(x_train, y_train)
        self.valid_dataset = CustomDataset(x_valid, y_valid)
        self.test_dataset = CustomDataset(x_test, y_test)

        self.train_loader = DataLoader(self.train_dataset, batch_size=32, shuffle=True)
        self.valid_loader = DataLoader(self.valid_dataset, batch_size=1)
        self.test_loader = DataLoader(self.test_dataset, batch_size=1)

        self.model = NeuralNetwork(input_size=self.feature_count, output_size=self.label_count)

    def test_dataset_length(self):
        # 데이터셋 크기 테스트
        self.assertEqual(len(self.train_dataset), len(self.train_dataset.targets))
        self.assertEqual(len(self.valid_dataset), len(self.valid_dataset.targets))
        self.assertEqual(len(self.test_dataset), len(self.test_dataset.targets))

    def test_model_forward_pass(self):
        # 모델의 순전파 테스트
        sample_data, _ = self.train_dataset[0]
        sample_data = sample_data.unsqueeze(0)  # 배치 차원 추가
        output = self.model(sample_data)
        self.assertEqual(output.shape, (1, self.label_count))

    def test_training_step(self):
        # 모델 학습 테스트
        optimizer = torch.optim.Adam(self.model.parameters(), lr=0.00001)
        criterion = torch.nn.BCELoss()

        self.model.train()
        for batch in self.train_loader:
            inputs, targets = batch
            optimizer.zero_grad()
            outputs = self.model(inputs)
            loss = criterion(outputs, targets)
            loss.backward()
            optimizer.step()
            break  # 첫 배치에 대해서만 테스트

        self.assertTrue(loss.item() > 0)
```

```python
    def test_validation_step(self):
        # 모델 검증 테스트
        criterion = torch.nn.BCELoss()

        self.model.eval()
        with torch.no_grad():
            for batch in self.valid_loader:
                inputs, targets = batch
                outputs = self.model(inputs)
                loss = criterion(outputs, targets)
                break  # 첫 배치에 대해서만 테스트

        self.assertTrue(loss.item() > 0)

if __name__ == "__main__":
    unittest.main()
```



```
....
----------------------------------------------------------------------
Ran 4 tests in 2.959s

OK
PS C:\Users\t0103\OneDrive\바탕 화면\24_1\python programming>
```

파일로도 첨부했습니다

## Collaboaration

우리의 아이디어와 코드를 중심으로 깃허브 내에 사용자들의 코드를 참고하였다.