# 기초 통계/ML 과제 보고서

한원석

1. Iris 데이터셋을 활용해 클래스별 변수 평균 차이를 검정
   1. 먼저 iris 데이터셋을 불러와 구조를 확인합니다.

```
import seaborn as sns

iris = sns.load_dataset('iris')
print(iris.head())
print(iris.info())
```

✓ 0.1s Python

```
   sepal_length  sepal_width  petal_length  petal_width species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
None
```

꽃받침의 너비 길이, 꽃잎의 너비의 길이가 float 타입으로, 종이 범주형 타입으로 입력되어 있으며, 총 150개의 데이터가 결측치 없이 존재하고 있음을 알 수 있습니다.

   2. 종별 기술통계량은 다음과 같다.

```
iris.groupby('species')['petal_length'].describe()
```

✓ 0.0s Python

| species | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| setosa | 50.0 | 1.462 | 0.173664 | 1.0 | 1.4 | 1.50 | 1.575 | 1.9 |
| versicolor | 50.0 | 4.260 | 0.469911 | 3.0 | 4.0 | 4.35 | 4.600 | 5.1 |
| virginica | 50.0 | 5.552 | 0.551895 | 4.5 | 5.1 | 5.55 | 5.875 | 6.9 |

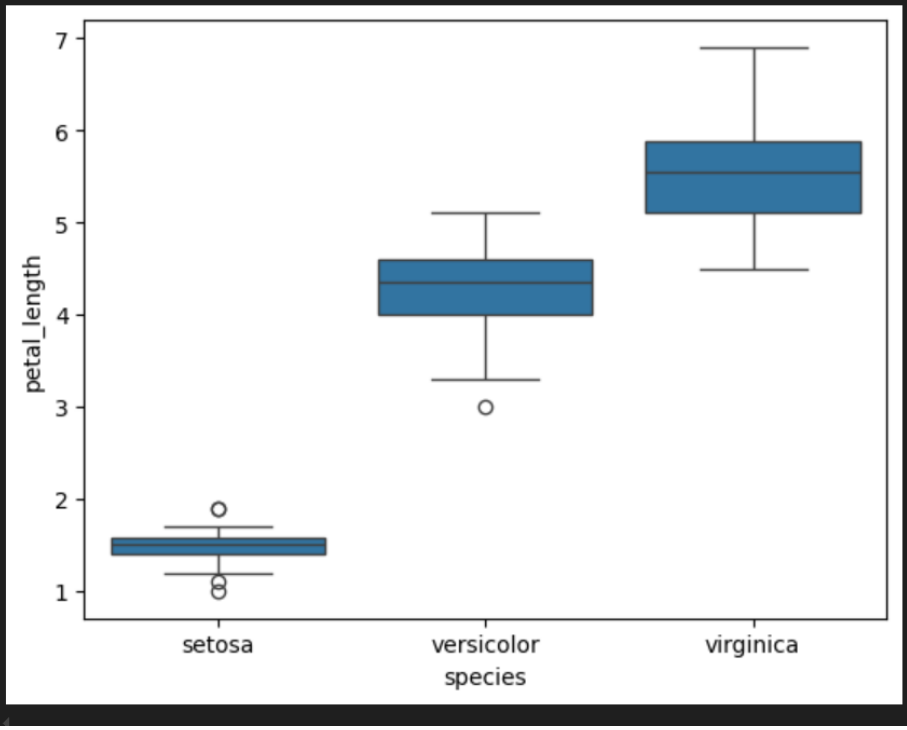
대체로 setosa보다 versicolor가, versicolor 보다 virginica가 큰 경향성을 보이고 있다는 것을 알 수 있다.

3. 해당 자료를 Boxplot으로 시각화하면 다음과 같다.

```
sns.boxplot(x='species', y='petal_length', data=iris)
```

✓ 0.0s Python

```
<Axes: xlabel='species', ylabel='petal_length'>
```




아까 본 경향성을 확연하게 체감할 수 있다.

4. Shapiro-Wilk 검정을 통해 해당 자료들이 정규분포를 따르는 데이터셋에서 나왔는지 검정할 수 있다. 이 때 귀무가설과 대립가설은 다음과 같다.

$H_0$ : 해당 데이터는 정규분포를 따른다.

$H_1$ : 해당 데이터는 정규분포를 따르지 않는다.

```
from scipy.stats import shapiro

groups = iris.groupby('species')['petal_length']

for species, group in groups:
    stat, p = shapiro(group)
    print(f"species : {species:15} p-value : {p}")
```

✓ 0.0s Python

```
species : setosa          p-value : 0.0548114671955363
species : versicolor      p-value : 0.15847783815657573
species : virginica       p-value : 0.10977536903223506
```

검정 결과에 따르면 세 그룹 모두 p-value가 0.05이상이므로 귀무가설을 기각하지 못하고, 정규성을 만족함을 알 수 있다.

5. Levene 검정을 통해 3개의 그룹이 등분산성을 가지는지 검정할 수 있다. 이 때 귀무가설과 대립가설은 다음과 같다.

   $H_0$ : 각 그룹의 분산이 동일하다.

   $H_1$ : 적어도 하나의 그룹의 분산은 동일하지 않다.

```python
from scipy.stats import levene

setosa = list(groups)[0][1]
versicolor = list(groups)[1][1]
virginica = list(groups)[2][1]

stat, p = levene(setosa, versicolor, virginica)
print(f"p-value = {p}")
```

✓ 0.0s Python

```
p-value = 3.1287566394085304e-08
```

   검정 결과에 따르면 p-value가 0.05 미만이므로 귀무가설이 기각되고, 대립가설이 채택되어 이 데이터는 등분산성을 만족하지 않는다.

6. 등분산성을 가정하고 평균의 차이를 구하기 위한 대립가설과 귀무가설을 수립하면 다음과 같다.

   $H_0$ : 각 그룹의 평균이 동일하다.

   $H_1$ : 적어도 하나의 그룹의 평균은 동일하지 않다.

7. One-way ANOVA를 통해 평균의 차이를 검정하면 다음과 같다.

```python
from scipy.stats import f_oneway

f_stat, p_value = f_oneway(setosa, versicolor, virginica)
print(f"p-value: {p_value}")
```

✓ 0.0s Python

```
p-value: 2.856776610961989e-91
```

   검정 결과에 따르면 p-value가 0.05 미만이므로 귀무가설이 기각되고, 대립가설이 채택되어 이 데이터는 그룹별 평균에 차이가 존재한다.

8. Tukey HSD를 통해 사후검정을 하면 다음과 같다. 이 사후 검정은 각각의 그룹을 비교하여 실질적으로 어떤 그룹의 평균끼리 차이가 존재하는 가를 확인할 수 있다.

```python
from statsmodels.stats.multicomp import pairwise_tukeyhsd

tukey = pairwise_tukeyhsd(endog=iris['petal_length'],
                          groups=iris['species'],
                          alpha=0.05)

print(tukey)
```

✓ 0.2s Python

```
  Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
  group1     group2   meandiff p-adj lower  upper  reject
-----------------------------------------------------
    setosa versicolor    2.798   0.0 2.5942 3.0018   True
    setosa  virginica     4.09   0.0 3.8862 4.2938   True
versicolor  virginica    1.292   0.0 1.0882 1.4958   True
-----------------------------------------------------
```

검정 결과에 따르면 모든 비교군에서 p-adj가 0으로, 이는 모두 p-value가 0.05 미만으로 아주 작다고 볼 수 있다. 따라서 모든 그룹의 평균의 차이가 유의미하게 존재한다.

9. Boxplot의 시각화 자료, ANOVA의 대립가설, Tukey HSD의 '모든 평균이 다르다'는 3가지 결과를 통해 각 그룹의 평균이 유의미하게 다르고, 그 길이는 virginica, versicolor, setosa순으로 길다는 것을 알 수 있다.

2. 실제 신용카드 사기 데이터셋을 활용해 클래스 불균형 상황에서 분류 모델을 학습
   1. 다음과 같은 코드를 통해 신용카드 정보에 대한 데이터를 불러온다.

```python
import pandas as pd

df = pd.read_csv('creditcard\creditcard.csv')

print(df.head())
print(df.info())
print(df.describe())

print(df['Class'].value_counts())
print(df['Class'].value_counts(normalize=True))
```

✓ 2.1s Python

총 31개의 열이 존재하며 시간, 금액, 그리고 28개의 마스킹된 벡터들이 float타입으로, Class가 0과 1의 int 타입으로 존재한다.

```
[8 rows x 31 columns]
Class
0    284315
1       492
Name: count, dtype: int64
Class
0    0.998273
1    0.001727
Name: proportion, dtype: float64
```

클래스의 개수와 비율을 확인해보면 클래스 간 데이터의 개수 차이가 굉장히 크다는 사실을 알 수 있다.

2. 클래스가 0인 데이터들을 일부 무작위로 undersampling하여 비율의 차이를 줄인다.

```python
fraud = df[df['Class'] == 1]
normal = df[df['Class'] == 0].sample(n=10000, random_state=42)

df_sampled = pd.concat([fraud, normal], axis=0).sample(frac=1, random_state=42)  # 섞기

print(df_sampled['Class'].value_counts())
```

✓ 0.0s Python

```
Class
0    10000
1      492
Name: count, dtype: int64
```

3. 숫자가 다른 열에 비해 큰 금액이 다른 열에 비해 과도하게 영향을 주지 않도록 정규화를 진행하고, 학습에 사용될 독립변수와, 종속변수를 구분한다.

```python
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df_sampled['Amount_Scaled'] = scaler.fit_transform(df_sampled[['Amount']])
df_sampled.drop('Amount', axis=1, inplace=True)

# X, y 분리
X = df_sampled.drop('Class', axis=1)
y = df_sampled['Class']
```

✓ 0.5s Python

4. 학습에 사용될 데이터와 평가에 사용될 데이터를 일정한 비율이 되로록 나누어준다.

```python
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)

print(y_train.value_counts(normalize=True))
print(y_test.value_counts(normalize=True))
```

✓ 0.7s Python

```
Class
0    0.953056
1    0.046944
Name: proportion, dtype: float64
Class
0    0.953311
1    0.046689
Name: proportion, dtype: float64
```

5. 아직 클래스간 비율의 차이가 심하므로 SMOTE를 통해 oversampling하여 비율을 일정하게 만든다. 이 때 SMOTE는 단순히 기존의 데이터를 복제하는 것이 아니라 선형보간을 해주어 데이터가 더 풍부하고 특정한 값에 매몰되지 않게 학습할 수 있도록 도와준다.

```python
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

print(y_train_smote.value_counts())
```

✓ 4.8s Python

```
Class
0    7999
1    7999
Name: count, dtype: int64
```

6. 학습을 진행하여 결과를 잘 예측할 수 있는 모델인지 평가한다.

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, average_precision_score

model = RandomForestClassifier(random_state=42)
model.fit(X_train_smote, y_train_smote)

# 예측
y_pred = model.predict(X_test)
y_proba = model.predict_proba(X_test)[:, 1]

# 평가
print(classification_report(y_test, y_pred, digits=4))
print("PR-AUC:", average_precision_score(y_test, y_proba))
```

✓ 5.7s Python

```
              precision    recall  f1-score   support

           0     0.9916    0.9990    0.9953      2001
           1     0.9759    0.8265    0.8950        98

    accuracy                         0.9909      2099
   macro avg     0.9837    0.9128    0.9451      2099
weighted avg     0.9908    0.9909    0.9906      2099

PR-AUC: 0.9156887960343137
```

7. 위의 결과를 통해 Recall ≥ 0.80, F1 ≥ 0.88, PR-AUC ≥ 0.90을 모든 클래스에 대하여 만족하고 있다. 따라서 이 모델을 잘 학습된 모델이라고 할 수 있다.