과제 1. Iris 데이터셋을 활용해 클래스별 변수 평균 차이를 검정

1. 데이터 불러오기 + 구조 확인

#Code

```
#데이터 불러오기 + 구조 확인
import seaborn as sns
iris = sns.load_dataset('iris')
print(iris.head())
```

- seaborn 라이브러리의 load_dataset 함수를 사용하여 iris 데이터 불러오고, head()로 데이터프레임의 상위 5개 행/전체 구조 확인

#Result

	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

- 컬럼은 sepal_length, sepal_width, petal_length, petal_width, species로 구성되어 있으며, 각 특성은 float64(연속형), species는 범주형으로 구성

2. 기술통계량 산출

#Code

```
#기술통계량 산출
desc = iris.groupby('species')['petal_length'].describe()
count = iris['species'].value_counts()
print(desc)
print(count)
```

- groupby를 사용하여 species별로 petal_length의 평균, 표준편차, 최소/최대, 사분위수, 데이터 개수 확인

#Result

	count	mean	std	min	25%	50%	75%	max
species								
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

```
species
setosa        50
versicolor    50
virginica     50
Name: count, dtype: int64
```

- 각 그룹은 50개의 표본으로 구성되어 있으며, 평균값은 Setosa < Versicolor < Virginica

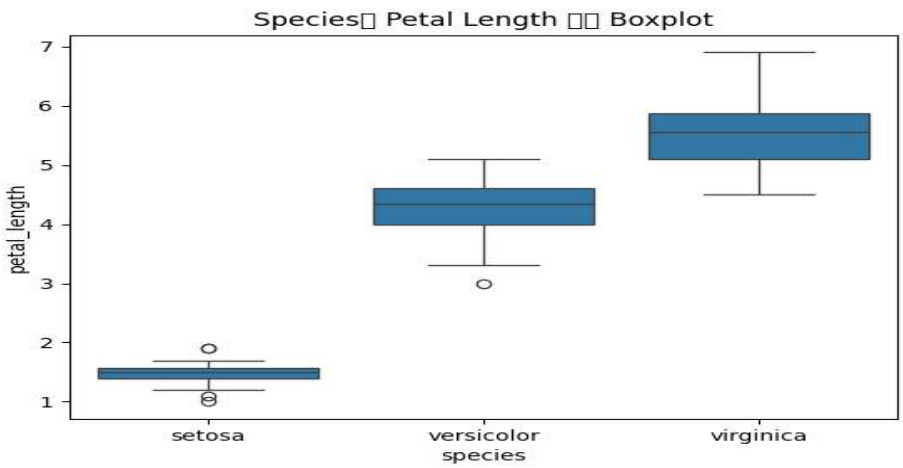
3. 시각화

#Code

```
#시각화
import matplotlib.pyplot as plt
sns.boxplot(x='species', y='petal_length', data=iris)
plt.title("Species_Petal_Length_Boxplot")
plt.show()
```

- species별 petal_length의 분포를 보기 위하여 species를 x축, petal_length를 y축으로 지정

#Result



- virginica의 Petal_length의 평균과 중앙값이 가장 크고, setosa가 가장 낮음. virginica가 가장 길고 setosa가 가장 짧으며, 세 그룹 간 분포 차이가 뚜렷함. setosa와 versicolor species는 이상치 발생.
- 특히 setosa는 중앙값과 전체 분포 범위가 가장 낮으며, IQR도 매우 작아 대부분의 데이터가 좁은 구간에 모여있음. Petal Length가 짧고 데이터가 균일하게 분포함
- 전체적으로 Setosa < Versicolor < Virginica 순으로 Petal Length의 중앙값과 분포 범위

가 증가하며, Virginica의 Petal Length는 분포가 넓고, 일부 데이터는 Versicolor와 약간 겹치나, 전체적으로 높은 값을 가짐. 특히, 해당 결과를 Boxplot으로 표현하여 ris의 세 품종은 Petal Length의 중앙값과 분포 폭에서 모두 뚜렷한 차이를 나타냄을 알 수 있음

4. 정규성 검정

#Code

```
#정규성 검정
from scipy.stats import shapiro
for species in iris['species'].unique():
    stat, p = shapiro(iris[iris['species'] == species]['petal_length'])
    print(f"{species}: p-value = {p:.4f}")
```

-species별로 검정을 실시하고 [H0: 각 그룹의 petal_length는 정규분포를 따른다. / H1: 정규분포를 따르지 않는다.]로 가설을 설정함.

#Result

```
setosa: p-value = 0.0548
versicolor: p-value = 0.1585
virginica: p-value = 0.1098
```

- 모든 그룹의 p-value가 0.05보다 크므로, 정규성을 만족한다고 판단

5. 등분산성 검정

#Code

```
#등분산성 검정
from scipy.stats import levene
setosa = iris[iris['species'] == 'setosa']['petal_length']
versicolor = iris[iris['species'] == 'versicolor']['petal_length']
virginica = iris[iris['species'] == 'virginica']['petal_length']
stat, p = levene(setosa, versicolor, virginica)
print(f"Levene p-value = {p:.4f}")
```

-Levene 검정 실시, [H0: 세 그룹의 분산이 같다. / H1: 적어도 한 그룹의 분산이 다르다.]로 가설을 설정함.

#Result

-p-vaule가 0,05보다 훨씬 작으므로, 귀무가설을 기각함에따라 세그룹의 분산은 서로 다름으

로 판단

6. 가설 수립

[귀무가설(H0): 3개 species의 petal_length 평균은 모두 같다./ 대립가설(H1): 적어도 한 그룹의 평균이 다르다.]로 가설을 설정함.

7. ANOVA 실행

#Code

```
#ANOVA 실행
from scipy.stats import f_oneway
f_stat, p = f_oneway(setosa, versicolor, virginica)
print(f"F-statistic = {f_stat:.4f}, p-value = {p:.4f}")
```

#Result

```
F-statistic = 1180.1612, p-value = 0.0000
```

-p-value가 0.05 미만이므로 귀무가설을 기각. 그룹 간 평균에 유의미한 차이가 있음.

8. 사후검정

#Code

```
#사후검정
from statsmodels.stats.multicomp import pairwise_tukeyhsd
tukey = pairwise_tukeyhsd(endog=iris['petal_length'], groups=iris['species'], alpha=0.05)
print(tukey.summary())
```

#Result

```
   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
-----------------------------------------------------
```

- 모든 그룹 쌍(setosa-versicolor, setosa-virginica, versicolor-virginica) 간의 결과를 분석하면, p-adj가 0.05 미만이며, reject = True 결과를 통해 두 집단 간 평균 차이에 대한 귀무가설(H0: 평균이 같다)을 기각할 수 있는지 여부를 확인함으로써 차이가 있다는 대립가설을 채택함.(세 쌍 간 모두 유의미한 차이가 있음)

9. 결과 요약

① 기술통계 및 시각화: 각 그룹의 Petal Length의 평균, 분산, 중앙값 및 사분위수를 확인하고 Boxplot으로 분포를 시각화

→ Virginica > Versicolor > Setosa 순으로 평균과 중앙값이 뚜렷하게 구분되며, 세 집단 사이 경계가 거의 겹치지 않음을 확인

② 정규성 및 등분산성 검정: Shapiro-Wilk로 각 품종의 Petal Length 분포가 정규분포를 만족함을 확인했으나, Levene 검정 결과, 세 집단 간 분산이 다름을 확인

③ ANOVA: 세 그룹의 평균 차이를 검정한 결과, F-통계량이 매우 크고 p값이 0에 가깝게 나타나 귀무가설(모든 그룹 평균이 같다)을 기각하여 세 품종 간 평균 Petal Length에 통계적으로 유의한 차이가 있음이 확인

④ 사후검정(Tukey HSD): 모든 그룹 쌍에서 평균 차이가 유의함(reject=True)을 확인하여 각 품종 간 Petal Length의 평균이 모두 서로 다르다는 결론을 내림

결론 요약 : Box-plot, ANOVA, Tukey HSD 결과를 종합한 결과, Virginica의 Petal Length가 통계적으로 가장 길고, Setosa가 가장 짧으며, 세 그룹 간 서로 유의미한 차이가 있음. 이에 따라, Iris 품종에 따라 Petal Length 평균이 다름을 알 수 있음

2. 실제 신용카드 사기 데이터셋을 활용해 클래스 불균형 상황에서 분류 모델을 학습

1. 데이터 로드 및 기본 탐색

#Code

```
##1.데이터 로드 및 기본 탐색
import pandas as pd
import numpy as np

# 데이터 로드
df = pd.read_csv('C:/Users/da396/creditcard.csv')

# 데이터 구조 확인
print(df.head())
print(df.info())
print(df.describe())

# Class 비율 확인
print('정상 거래(Class=0) 건수:', (df['Class'] == 0).sum())
print('사기 거래(Class=1) 건수:', (df['Class'] == 1).sum())
print('전체 거래 중 사기 거래 비율:', round((df['Class'] == 1).mean()*100, 4), '%')
```

#Result

```
[5 rows x 31 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
```

```
memory usage: 67.4 MB
None
              Time         V1  ...      Amount         Class
count  284807.000000  2.848070e+05  ...  284807.000000  284807.000000
mean    94813.859575  1.175161e-15  ...      88.349619       0.001727
std     47488.145955  1.958696e+00  ...     250.120109       0.041527
min         0.000000 -5.640751e+01  ...       0.000000       0.000000
25%     54201.500000 -9.203734e-01  ...       5.600000       0.000000
50%     84692.000000  1.810880e-02  ...      22.000000       0.000000
75%    139320.500000  1.315642e+00  ...      77.165000       0.000000
max    172792.000000  2.454930e+00  ...   25691.160000       1.000000

[8 rows x 31 columns]
정상 거래(Class=0) 건수: 284315
사기 거래(Class=1) 건수: 492
전체 거래 중 사기 거래 비율: 0.1727 %
```

- 총 31개의 열로 구성되어 있으며, 각 컬럼의 타입은 class만 int64이고, 나머지는 float64이며 결측치는 존재하지 않음. 전체 거래 건수는 284,807건, 이 중 사기 거래(Class=1)는 492건(0.17%), 정상 거래(Class=0)는 284,315건(99.83%)으로 데이터가 불균형함을 확인

2. 샘플링

#Code

```
## 2.샘플링
# 사기 거래는 모두 유지
fraud = df[df['Class'] == 1]

# 정상 거래는 10,000건만 무작위 샘플링
normal = df[df['Class'] == 0].sample(n=10000, random_state=42)

# 두 데이터 합치기
df_sampled = pd.concat([fraud, normal])

# Class 비율 재확인
print(df_sampled['Class'].value_counts(normalize=True))
print(df_sampled['Class'].value_counts())
print('샘플링 후 전체 거래 중 사기 거래 비율:', round((df_sampled['Class'] == 1).mean()*100, 4), '%')
```

- 사기 거래 전체는 보존하고, 정상 거래는 10,000건만 랜덤 샘플링 후 두 데이터를 결합하여 새로운 분석용 데이터프레임을 만듬.

#Result

```
Class
0    0.953107
1    0.046893
Name: proportion, dtype: float64
Class
0    10000
1      492
Name: count, dtype: int64
샘플링 후 전체 거래 중 사기 거래 비율: 4.6893 %
```

- 10,000건만 무작위로 추출하여 클래스를 재확인한 결과, 사기 거래 비율이 4.7%로 상승

3. 데이터 전처리

#Code

```
## 3.데이터 전처리
from sklearn.preprocessing import StandardScaler

# Amount 표준화 (Amount_Scaled로 대체)
scaler = StandardScaler()
df_sampled['Amount_Scaled'] = scaler.fit_transform(df_sampled[['Amount']])
df_sampled = df_sampled.drop('Amount', axis=1)

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

- 표준화를 적용한 Amount_Scaled 변수로 대체하여 기존의 Amount 컬럼을 삭제하고 입력 X(특징)와 y(목표)를 분리함. 즉, Class를 제외한 전부가 X, y는 Class

4. 학습 데이터와 테스트 데이터 분할

#Code

```
## 4.학습 데이터와 테스트 데이터 분할
from sklearn.model_selection import train_test_split

# 학습:테스트 = 8:2
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)
print('학습셋 클래스 비율:\n', y_train.value_counts(normalize=True))
print('테스트셋 클래스 비율:\n', y_test.value_counts(normalize=True))
```

#Result

```
학습셋 클래스 비율:
 Class
0    0.953056
1    0.046944
Name: proportion, dtype: float64
테스트셋 클래스 비율:
 Class
0    0.953311
1    0.046689
Name: proportion, dtype: float64
```

- 학습/테스트셋 분할 결과 모두 Class의 비율이 앞서 확인했던 사기 탐지 class 비율인 4%로 동일하게 유지됨을 알 수 있음

5. SMOTE 적용

#Code

```
## 5.SMOTE 적용
from imblearn.over_sampling import SMOTE

# SMOTE 적용 전
print('SMOTE 적용 전 클래스 분포:', y_train.value_counts())

# SMOTE 적용 (random_state=42)
smote = SMOTE(random_state=42)
X_train_sm, y_train_sm = smote.fit_resample(X_train, y_train)

# SMOTE 적용 후
print('SMOTE 적용 후 클래스 분포:', y_train_sm.value_counts())
```

- SMOTE는 소수 클래스의 샘플 수를 인공적으로 생성하여 클래스 불균형 문제를 완화하는 오버샘플링의 기법 중 하나로(데이터가 불균형하면 소수 클래스를 충분히 학습하지 못하는 문제 발생 가능성 있음), 이를 활용함으로써 소수 클래스에 대한 예측 성능을 향상시킬 수 있음. 즉, 앞 단계에서 확인한 클래스 분포 비율로 알 수 있는 데이터 불균형 문제를 해결하기 위함.

#Result

```
SMOTE 적용 전 클래스 분포: Class
0    7999
1     394
Name: count, dtype: int64
SMOTE 적용 후 클래스 분포: Class
0    7999
1    7999
Name: count, dtype: int64
```

-SMOTE 적용 전 데이터셋에서는 사기 클래스의 데이터 수가 매우 적었으나, 적용 후에는 정상 거래와 사기 거래의 데이터 수가 같아짐으로써 데이터셋의 클래스 균형이 개선됨

6. 모델 학습

#Code

```
## 6.모델 학습 및 평가
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, average_precision_score

# 모델 학습
clf = RandomForestClassifier(random_state=42, class_weight='balanced', n_estimators=100)
clf.fit(X_train_sm, y_train_sm)

# 예측값 및 예측 확률
y_pred = clf.predict(X_test)
y_proba = clf.predict_proba(X_test)[:, 1]

# 평가지표 출력
print('classification_report\n', classification_report(y_test, y_pred, digits=4))
print('PR-AUC:', average_precision_score(y_test, y_proba))
```

-랜덤포레스트를 사용하여 모델을 학습하였으며, class_weight='balanced'로 불균형 문제를 추가적으로 보정한 후, 테스트셋에서 예측값과 예측 확률을 평가한 후 실제 값과 비교하여 평가 지표를 출력함.

#Result

```
classification_report
              precision    recall  f1-score   support

           0     0.9945    0.9975    0.9960      2001
           1     0.9457    0.8878    0.9158        98

    accuracy                         0.9924      2099
   macro avg     0.9701    0.9426    0.9559      2099
weighted avg     0.9922    0.9924    0.9923      2099

PR-AUC: 0.9537470537926648
```

- Class 0과 1 모두 recall >= 0.80, F1 >= 0.88, PR-AUC >= 0.90을 달성함

7. 최종 성능 평가

#Code

```
# threshold 조정
custom_threshold = 0.2
y_pred_custom = (y_proba >= custom_threshold).astype(int)
print(classification_report(y_test, y_pred_custom, digits=4))
print('조정 후 PR-AUC:', average_precision_score(y_test, y_proba))
```

#Result

```
    accuracy                         0.9924      2099
   macro avg     0.9701    0.9426    0.9559      2099
weighted avg     0.9922    0.9924    0.9923      2099

PR-AUC: 0.9537470537926648
              precision    recall  f1-score   support

           0     0.9984    0.9630    0.9804      2001
           1     0.5621    0.9694    0.7116        98

    accuracy                         0.9633      2099
   macro avg     0.7803    0.9662    0.8460      2099
weighted avg     0.9781    0.9633    0.9679      2099
```

- Threshold를 0.1~0.5를 설정하여 조정한 결과, 0.2가 가장 우수한 결과를 출력함.