ML Report

2022135030
박준범

1. 데이터 로드 및 기본 탐색

```
credit_df = pd.read_csv("creditcard.csv")
credit_df.head()
```

✓ 0.8s Python

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.3637
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.2554
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.5146
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.3870
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.8177

5 rows × 31 columns

```
credit_df.describe()
```

✓ 0.2s

	Time	V1	V2	V3	V4	V5	V6	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.
mean	94813.859575	1.175161e-15	3.384974e-16	-1.379537e-15	2.094852e-15	1.021879e-15	1.494498e-15	-5
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.

8 rows × 31 columns

```
credit_df.info()
```

✓ 0.0s Python

```
<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
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
...
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

```
print(credit_df['Class'].value_counts())
print(credit_df['Class'].value_counts(normalize=True)) # 비율 확인
```
✓ 0.0s

```
Class
0    284315
1       492
Name: count, dtype: int64
Class
0    0.998273
1    0.001727
Name: proportion, dtype: float64
```

총 284,807건의 신용카드 거래 데이터로 구성, 30개의 feature와 1개의 target 변수를 포함한다.

-클래스 비율

Class 0(정상 거래): 284315건, 약 99.83%

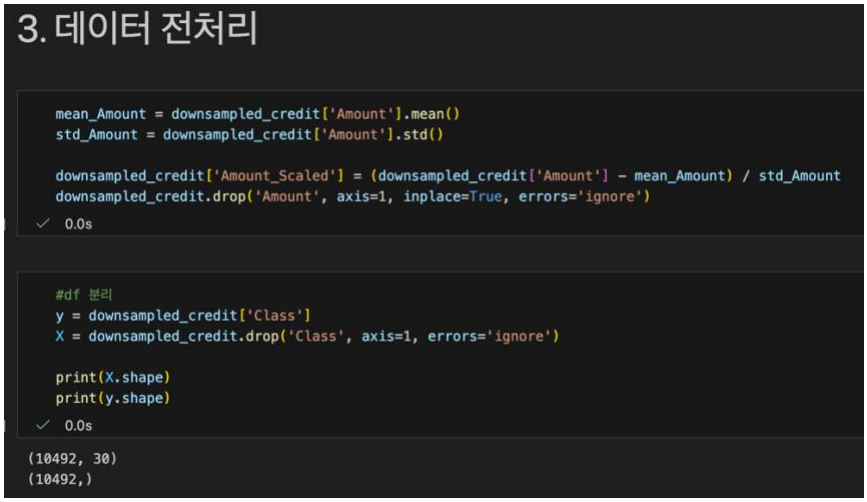
Class 1(사기 거래): 492건, 약 0.17 %

=> 극심한 불균형 데이터셋

2. 샘플링

다운 샘플링 진행: 정상 거래 데이터를 10,000건으로 무작위 샘플링하여 기존 사기 거래(492건)와 합친 새로운 데이터셋을 구축

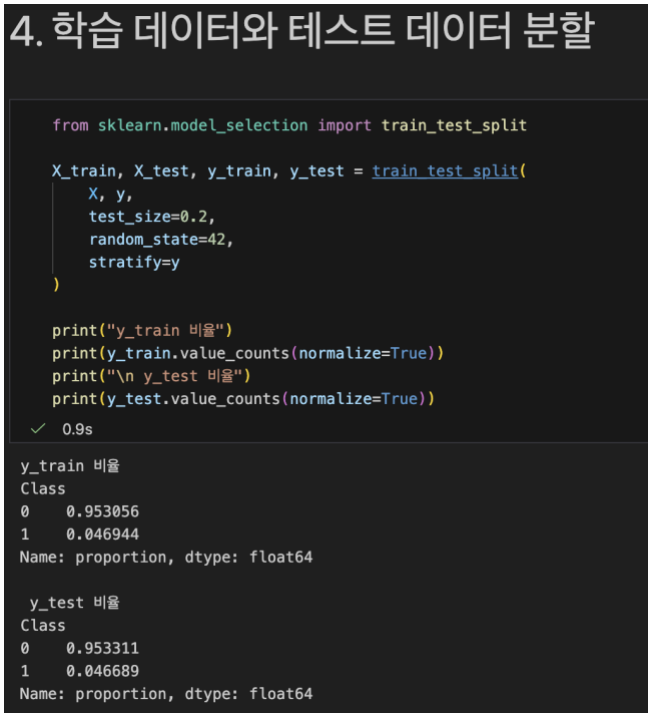
3. 데이터 전처리



Amount 변수가 다른 변수들에 비해 변수 값의 범위가 크므로 표준화 진행 (Amount_Scaled) 후 원본 변수를 제거

모델 학습을 위해 독립 변수(X)와 종속 변수(y)를 분리. 타겟 변수인 class를 y로 설정하고, 이를 제외한 나머지 feature들을 X로 구성

4. 학습 데이터와 테스트 데이터 분할



train_test_split을 사용해 학습셋:테스트셋 비율을 8:2로 나누고

stratify=y 옵션으로 클래스 비율 유지, 분할된 데이터의 Class 비율을 출력

5. SMOTE(Synthetic Minority Over-sampling Technique) 적용

5. SMOTE 적용

```
from imblearn.over_sampling import SMOTE
```

✓ 0.4s

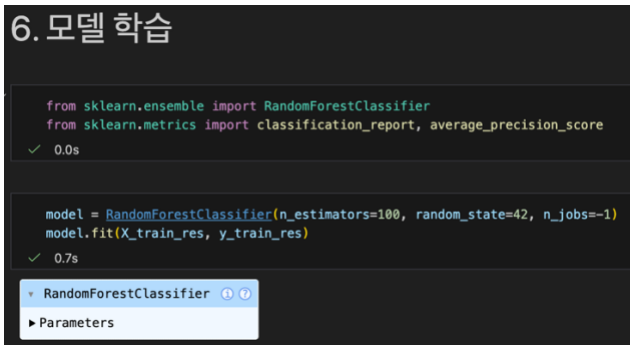
```
sm = SMOTE(random_state=42)

before_1count = (y_train == 1).sum()
X_train_res, y_train_res = sm.fit_resample(X_train, y_train)

after_1count = (y_train_res == 1).sum()

print(f"SMOTE 적용 전 사기 거래 건 수: {before_1count}")
print(f"SMOTE 적용 후 사기 거래 건 수: {after_1count}")
```

✓ 0.0s

```
SMOTE 적용 전 사기 거래 건 수: 394
SMOTE 적용 후 사기 거래 건 수: 7999
```

다운샘플링만으로는 정보 손실이 발생할 수 있으므로, 이러한 정보 손실을 최소화하고 클래스 간 불균형 해소를 위해 SMOTE 기법을 적용하여 사기 거래 데이터를 늘려준다.

6. 모델 학습



정상 거래(Class 0)과 사기 거래(Class 1)을 분류하는 binary classification이므로, RandomForestClassifier 모델을 사용하여 학습 진행

```python
y_pred = model.predict(X_test) #Threshold 0.5
y_prob = model.predict_proba(X_test)[:, 1]

print("예측값 (Predict)")
print(y_pred[:10])
print("\n예측 확률 (Predict Proba)")
print(y_prob[:10])

print("\nClassification Report")
print(classification_report(y_test, y_pred))

ap_score = average_precision_score(y_test, y_prob)
print(f"PR-AUC(Average Precision): {ap_score:.4f}")
```

✓ 0.0s

```
예측값 (Predict)
[0 0 0 0 0 0 1 0 0 0]

예측 확률 (Predict Proba)
[0.02 0.   0.03 0.   0.18 0.   1.   0.   0.03 0.05]

Classification Report
              precision    recall  f1-score   support

           0       0.99      1.00      1.00      2001
           1       0.95      0.89      0.92        98

    accuracy                           0.99      2099
   macro avg       0.97      0.94      0.96      2099
weighted avg       0.99      0.99      0.99      2099

PR-AUC(Average Precision): 0.9538
```

SMOTE를 통해 균형을 맞춘 학습 데이터를 바탕으로 모델을 학습 진행, 이후 테스트셋을 통해 각 거래에 대한 예측값(predict)과 사기 거래일 확률인 예측 확률(predict_proba)을 산출

7. 최종 성능 평가

Class 0 (정상): Precision 0.99 / Recall 1.00 / F1 1.00

Class 1 (사기): Precision 0.95 / Recall 0.89 / F1 0.92

PR-AUC(Average Precision): 0.9538

=> 목표 달성

현재 목표치를 달성하였으나, 향후 성능을 더욱 고도화하기 위해 LightGBM/XGBoost와 같이 다른 앙상블 모델과의 성능 비교를 수행해 볼 수 있다.

또한 Threshold 조정함으로써 recall을 더욱 높이는 방향으로 모델 최적화가 가능할 것이다.