# ML 과제

## 1. 데이터 로드 및 기본 탐색

▼ 전체 코드

```
# 데이터 불러오고 구조 확인
import pandas as pd
df = pd.read_csv("creditcard.csv")
df.head()
df.info()
df.describe()

# Class 분포 확인
print(df["Class"].value_counts())
print(df["Class"].value_counts(normalize=True))
```

먼저 `creditcard.csv` 데이터를 불러와 `head()`, `info()`, `describe()`를 통해 데이터의 전반적인 구조와 특성을 확인하였다.

해당 데이터셋은 총 284,807개의 거래 데이터로 구성되어 있다. `info()` 로 확인한 결과 각 관측치는 30개의 변수를 포함하고 있으며 모든 변수에 결측치는 없었다. 변수는 다음과 같다.

- `Time`
- `Amount`
- `V1 ~ V28`
- `Class`: 거래가 정상 거래인지(Class=0) 또는 사기 거래인지(Class=1)를 나타내는 이진 타깃 변수

  `describe()`를 통해 수치형 변수들의 분포를 확인한 결과, `V1`부터 `V28`까지의 변수들은 이미 전처리가 적용된 변수로, 평균이 0에 가깝고 표준편차가 비교적 일정한 분포를 보였다. 반면, 각각 거래 발생 시간과 거래 금액을 나타내는 원본 스케일 변수로 판단되는 `Time`과 `Amount` 변수는 평균과 표준편차가 상대적으로 크고 분포가 비대칭적인 특성을 나타냈다.

그 다음 우리의 타켓 변수인 Class의 분포를 확인하였다.

```
Class
0    284315
1       492
Name: count, dtype: int64

Class
0    0.998273
1    0.001727
Name: proportion, dtype: float64
```

전체 데이터 중 사기 거래가 차지하는 비율은 약 0.17%로, 정상 거래에 비해 매우 적은 수준으로 데이터셋이 극단적인 클래스 불균형(class imbalance) 문제를 가지고 있음을 확인할 수 있었다.

## 2. 샘플링

▼ 전체 코드

```
class1 = df[df["Class"]==1]
class0 = df[df["Class"]==0].sample(n=10000,random_state=4
2)
df_new = pd.concat([class1, class0], axis=0)

print(df_new["Class"].value_counts())
print(df_new["Class"].value_counts(normalize=True))

# 확인
df_new.head()
```

클래스 불균형 문제를 완화하기 위해 샘플링을 수행하여 새로운 분석용 데이터프레임을 구성할 수 있다. 먼저 사기 거래(Class 1)는 데이터 수가 매우 적기 때문에 모든 샘플을 유지하였고, 정상 거래(Class 0)는 무작위로 10,000건을 추출하였다. 이후 두 데이터를 결합하여 새로운 분석용 데이터프레임을 생성하고, 이를 `df_new`로 저장하였다.

`df_new`의 클래스별 거래 건수와 비율을 확인한 결과는 다음과 같다.

```
Class
010000
1492
Name: count, dtype: int64

Class
00.953107
10.046893
Name: proportion, dtype: float64
```

정상 거래는 10,000건, 사기 거래는 492건으로 의도한 바와 같이 데이터가 구성되었음을 확인할 수 있었다. 클래스 비율을 살펴보면 정상 거래가 약 95.3%, 사기 거래가 약 4.7%를 차지하여, 원본 데이터에 비해 사기 거래의 비중이 상대적으로 증가하는걸 확인하였다.

# 3. 데이터 전처리

▼ 전체 코드

```
# 전처리 후 원래 칼럼 제거
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

df_new["Amount_Scaled"] = scaler.fit_transform(df_new[["Am
ount"]])
df_new.drop(columns=["Amount"], inplace=True)

# 결과 확인
df_new.head()

# X, y 분리
X = df_new.drop(columns=["Class"])
```

```
y = df_new["Class"]

print(X.shape, y.shape)
```

모델 학습 앞서 데이터 전처리를 수행한다. 앞서 분포 분석 결과, 거래 금액을 나타내는 `Amount` 변수는 다른 변수들과 스케일 차이가 존재하는 원본 스케일 변수로 확인되었기 때문에, 표준화를 적용할 수 있다. 이를 위해 `StandardScaler`를 사용하여 `Amount` 변수를 평균 0, 표준편차 1을 갖도록 변환하고, 변환된 값을 `Amount_Scaled`라는 새로운 변수로 생성하고 기존의 `Amount` 변수는 데이터프레임에서 제거하였다. `df_new.head()` 를 통해 `Amount_Scaled` 변수가 정상적으로 생성된 것을 확인하였다.

이후 모델 학습을 위해 타깃 변수인 `Class`를 제외한 나머지 변수들을 독립 변수(`X`)로 설정하고, `Class` 변수를 종속 변수(`y`)로 분리하였다. 전처리 후 데이터의 차원은 `X`가 (샘플 수, 변수 수), `y`가 (샘플 수,) 형태로 구성되었음을 확인하였다.

```
(10492, 30) (10492,)
```

# 4. 학습/ 테스트 데이터 분리

▼ 전체 코드

```
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))
```

전체 데이터를 train/ test 데이터로 분리한다. 데이터 분할은 전체 데이터의 **80%를 학습 데이터**, **20%를 테스트 데이터**로 설정하였으며, 무작위 분할로 인한 클래스 비율 왜곡을 방지하기 위해 `stratify=y` 옵션을 적용하였다. 이를 통해 학습 데이터와 테스트 데이터 모두에서 사기 거래(Class 1)와 정상 거래(Class 0)의 비율이 원본 데이터와 유사하게 유지되도록 하였다. 또한 실험의 재현성을 확보하기 위해 `random_state=42`를 설정하였다.

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

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

분할 이후 학습 데이터와 테스트 데이터의 클래스 비율을 확인한 결과, 두 데이터셋 모두 정상 거래와 사기 거래의 비율이 거의 동일하게 유지됨을 확인하였다.

# 5. SMOTE

▼ 전체 코드

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)

X_train_sm, y_train_sm = smote.fit_resample(X_train, y_tra
in)

print("Before SMOTE")
print(y_train.value_counts())
```

```
print("After SMOTE")
print(y_train_sm.value_counts())
```

앞선 샘플링 과정을 통해 클래스 불균형을 일부 완화하였으나, 학습 데이터 내에서는 여전히 정상 거래(Class 0)에 비해 사기 거래(Class 1)의 비율이 낮은 상태이다. 이러한 불균형 상태에서 모델을 학습할 경우, 모델이 정상 거래 위주로 학습되어 사기 거래를 충분히 탐지하지 못하는 문제가 발생할 수 있다. 특히 본 문제에서는 사기 거래를 정확히 탐지하는 것이 핵심이므로, 이를 보완하기 위해 학습 데이터(`X_train`, `y_train`)에 대해 SMOTE(Synthetic Minority Over-sampling Technique)를 적용하였다.

SMOTE는 소수 클래스에 해당하는 샘플들의 특성 공간에서 이웃 관계를 기반으로 새로운 가상의 데이터를 생성하는 오버샘플링 기법으로, 단순히 기존 데이터를 복제하는 방식에 비해 과적합을 완화하면서 소수 클래스의 분포를 보다 균형 있게 확장할 수 있다는 장점이 있다.

SMOTE 적용 전후의 사기 거래 건수를 비교한 결과는 다음과 같다.

```
Before SMOTE
Class
0    7999
1     394
Name: count, dtype: int64

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

SMOTE 적용 이후 학습 데이터에서 정상 거래(Class 0)와 사기 거래(Class 1)의 샘플 수가 동일하게 맞춰졌음을 확인할 수 있다.

# 6. 모델 학습

▼ 전체 코드

```
# 모델 학습
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(
    max_iter=1000,
    class_weight=None,
    random_state=42
)

model.fit(X_train_sm, y_train_sm)

# 예측값, 예측 확률 출력
y_pred = model.predict(X_test)
y_pred

y_proba = model.predict_proba(X_test)[:, 1]
print(y_proba)

# 성능 출력: classification report/ PR-AUC

from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))

from sklearn.metrics import average_precision_score

pr_auc = average_precision_score(y_test, y_proba)
print("PR-AUC:", pr_auc)
```

이진 분류 문제에 적합한 Logistic Regression 모델을 사용하여 학습을 수행하였다. 앞 단계에서 SMOTE를 적용하여 클래스 불균형을 보정하였으므로, 추가적인 가중치 조정(`class_weight`)은 적용하지 않았다.

SMOTE가 적용된 학습 데이터(`X_train_sm`, `y_train_sm`)를 이용해 모델을 학습한 후, 테스트 데이터에 대해 예측 클래스(`predict`)와 사기 거래에 대한 예측 확률(`predict_proba`)을 산출하였다. 예측 확률은 각 거래가 사기 거래(Class 1)일 가능성을 의미한다.

```
array([0, 0, 0, ..., 0, 0, 0])

[1.15128251e-01 9.50589962e-02 4.45683029e-02 ... 4.42596210e
-04
 7.84810860e-04 7.46035621e-07]
```

출력된 예측 확률 값들은 전반적으로 낮은 값을 보였으며, 기본 임계값(threshold=0.5)을 기준으로 대부분의 샘플이 Class 0으로 분류되었음을 확인할 수 있다.

테스트 데이터에 대한 분류 성능을 평가하기 위해 다음과 같은 두 가지 지표를 사용하였다.

1. `classification_report` : Precision, Recall, F1-Score

```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      2001
           1       0.78      0.93      0.85        98

    accuracy                           0.98      2099
   macro avg       0.89      0.96      0.92      2099
weighted avg       0.99      0.98      0.99      2099
```

- Precision은 분류 모델이 positive(Class 1)로 판정한 샘플 중 실제로 positive인 샘플의 비율을 의미한다.
- Recall은 실제 positive(Class 1) 샘플 중 분류 모델이 positive로 올바르게 판정한 비율을 의미한다.
- F1-score은 precision과 recall의 조화 평균이다.

Class 1(사기 거래)에 대해 Recall은 0.93, Precision은 0.78, F1-score는 0.85로 나타났다. 이는 전체 사기 거래 중 약 93%를 모델이 성공적으로 탐지하였음을 의미하며, 모델이 사기 거

래를 놓치지 않고 탐지하는 데 높은 성능을 보였음을 시사한다. 다만, 사기 거래로 예측한 샘플 중 약 78%만이 실제 사기 거래로 확인되어 일부 오탐이 존재함을 알 수 있다.

한편, Class 0(정상 거래)에 대해서는 Precision이 1.00, Recall이 0.99로 매우 높은 성능을 보여, 정상 거래를 정확하게 분류하고 있음을 확인하였다.

2. `average_precision_score`: PR-AUC

```
PR-AUC: 0.9495554320771377
```

- PR-AUC는 Precision–Recall 곡선 아래 면적으로, 다양한 임계값(threshold)에 대해 모델의 분류 성능을 종합적으로 평가하는 지표이다. `average_precision_score`는 PR 곡선을 기반으로 한 평균 정밀도를 계산한다.

PR-AUC의 값은 약 0.95로 나타났으며, 이는 불균형 데이터 환경에서도 모델이 사기 거래를 비교적 안정적으로 구분하고 있음을 의미한다.

# 7. 최종 성능

정상 거래(Class 0)에 대해서는 Recall이 0.99, F1-score가 0.99로 나타나 목표 기준치보다 충분히 높은 성능을 보였다. 한편, 사기 거래(Class 1)에 대해서는 Recall이 0.93으로 목표 기준을 충족하였으나, F1-score는 0.85로 목표 기준인 0.88에는 다소 미치지 못하였다. PR-AUC의 값은 0.95로 나타나 목표 기준치를 상회하였다.

사기 거래(Class 1)의 F1-score 성능을 개선하기 위해, 기본값인 0.5로 설정되어 있던 threshold을 0.7로 조정해보았다.

코드와 결과는 다음과 같다.

```
threshold = 0.7
y_pred_thresh = (y_proba >= threshold).astype(int)
print(classification_report(y_test, y_pred_thresh))

              precision    recall  f1-score   support

           0       0.99      0.99      0.99      2001
```

```
           1       0.87      0.90      0.88        98

    accuracy                           0.99      2099
   macro avg       0.93      0.95      0.94      2099
weighted avg       0.99      0.99      0.99      2099
```

그 결과, 사기 거래(Class 1)에 대한 Precision이 0.78에서 0.87로 증가하였고, Recall은 0.93에서 0.90으로 소폭 감소하였다. Precision 증가 폭이 Recall 감소 폭보다 상대적으로 컸기 때문에, 두 지표의 조화 평균인 F1-score가 0.85에서 0.88로 개선되어 목표 기준치를 달성했다. 사기 거래가 전체 거래 중 차지하는 비중이 낮기 때문에 결정 임계값을 상향 조정하였고, 이 때문에 사기 거래로 분류되는 기준이 보다 엄격해져 오탐이 줄어든 것으로 해석할 수 있다.

또한 Logistic Regression은 선형 모델이기 때문에, 복잡한 비선형 패턴을 충분히 반영하는 데 한계가 있다. 따라서 Gradient Boosting, XGBoost, LightGBM과 같은 비선형 모델을 적용하거나, 하이퍼파라미터 튜닝을 통해 추가적인 성능 개선을 시도해볼 수 있다.