2025-1 기계학습 팀 프로젝트

Credit Card Fraud Detection

1조 강나언 서동주 이정연 이현석

CONTENTS

1. Dataset Description

1. Dataset Description

Problem Scenario

Credit Card Fraud Detection

Anonymized credit card transactions labeled as fraudulent or genuine

- 2013년 9월에 발생한 유럽 카드 소지자들의 신용카드 거래 내역
- 총 거래 건수는 284,807건이며, 이 중 492건(0.172%)이 사기로 확인
- 데이터 수집 기간: 이틀(48시간) 동안 발생한 모든 거래 정보
- 데이터에는 PCA(주성분 분석)를 통해 변환된 수치형 입력 변수들만 포함
- 신용카드 회사는 사기성 거래를 빠르고 정확하게 탐지함으로써, 고객이 실제로 구매하지 않은 항목에 대해 요금을 청구당하지 않도록 해야 한다.

- Binary Classification
- Kaggle Dataset
- AUPRC (Precision-Recall Curve 아래 면적) 사용 권장

1. Dataset Description

Column Overview

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
265518	161919.0	1.946747	-0.752526	-1.355130	-0.661630	1.502822	4.024933	-1.479661	1.139880	1.406819	...	0.076197	0.297537	0.307915	0.690980	-0.350316	-0.388907	0.077641	-0.032248	7.32	0
180305	124477.0	2.035149	-0.048880	-3.058693	0.247945	2.943487	3.298697	-0.002192	0.674782	0.045826	...	0.038628	0.228197	0.035542	0.707090	0.512885	-0.471198	0.002520	-0.069002	2.99	0
42664	41191.0	-0.991920	0.603193	0.711976	-0.992425	-0.825838	1.956261	-2.212603	-5.037523	0.000772	...	-2.798352	0.109526	-0.436530	-0.932803	0.826684	0.913773	0.038049	0.185340	175.10	0
198723	132624.0	2.285718	-1.500239	-0.747565	-1.668119	-1.394143	-0.350339	-1.427984	0.010010	-1.118447	...	-0.139670	0.077013	0.208310	-0.538236	-0.278032	-0.162068	0.018045	-0.063005	6.10	0
82325	59359.0	-0.448747	-1.011440	0.115903	-3.454854	0.715771	-0.147490	0.504347	-0.113817	-0.044782	...	-0.243245	-0.173298	-0.006692	-1.362383	-0.292234	-0.144622	-0.032580	-0.064194	86.10	0

V1, V2, ⋯, V28 원본 거래 내역을 익명화(보호)하기 위해 PCA를 통해 얻은 28개의 주성분

Amount 해당 거래의 금액(화폐 단위: 유로(EUR)

Time 각 거래 발생 시점과 데이터셋의 첫 번째 거래 발생 시점 사이의 경과된 시간(초)

Class Class 1: 사기거래 (fraud)
Class 0: 정상 거래 (Legitimate)

총 31개의 컬럼

2. EDA

2. EDA

Missing Values

```
# 결측치 확인
total_cells = train.shape[0] * train.shape[1]

# 결측치 총 개수
total_missing = train.isnull().sum().sum()

# 결측치 비율
missing_rate = total_missing / total_cells * 100 # % 단위
print(f"Missin Rate {missing_rate: .2f}%")
```

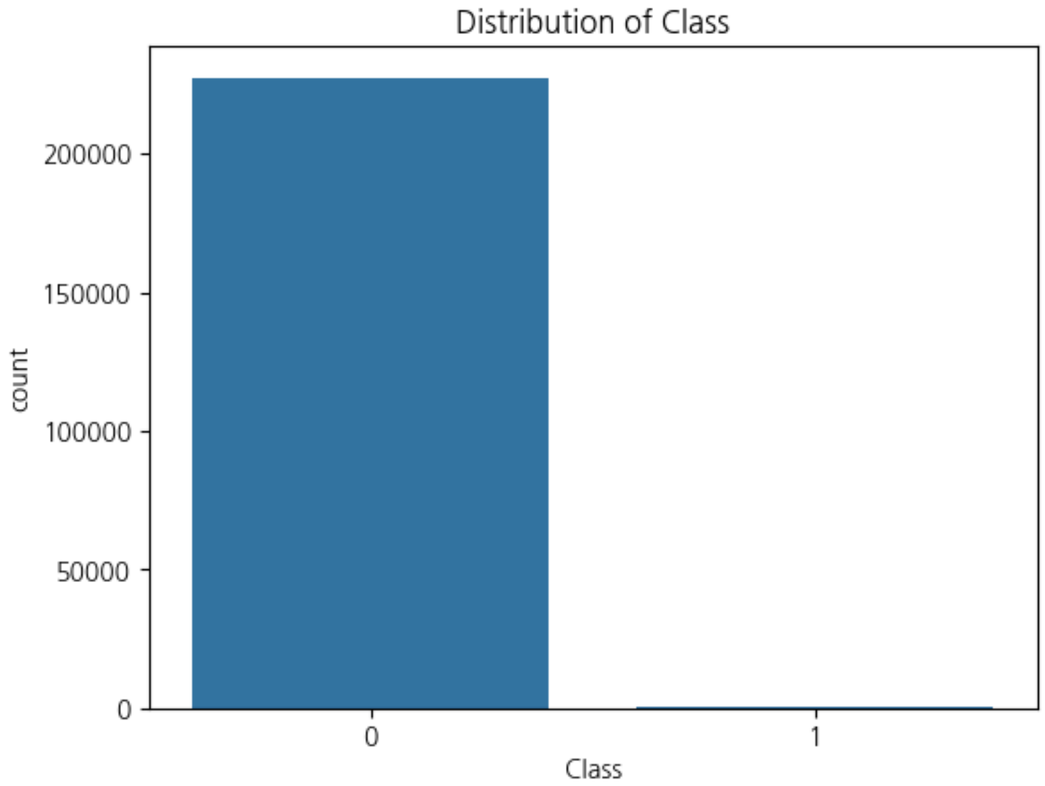
```
Missin Rate  0.00%
```

Library	Function
pandas	DataFrame.isnull() Series.sum()
전체 데이터프레임의 결측값 개수와 비율 확인	

결측치 없음.

2. EDA

Target Class Distribution



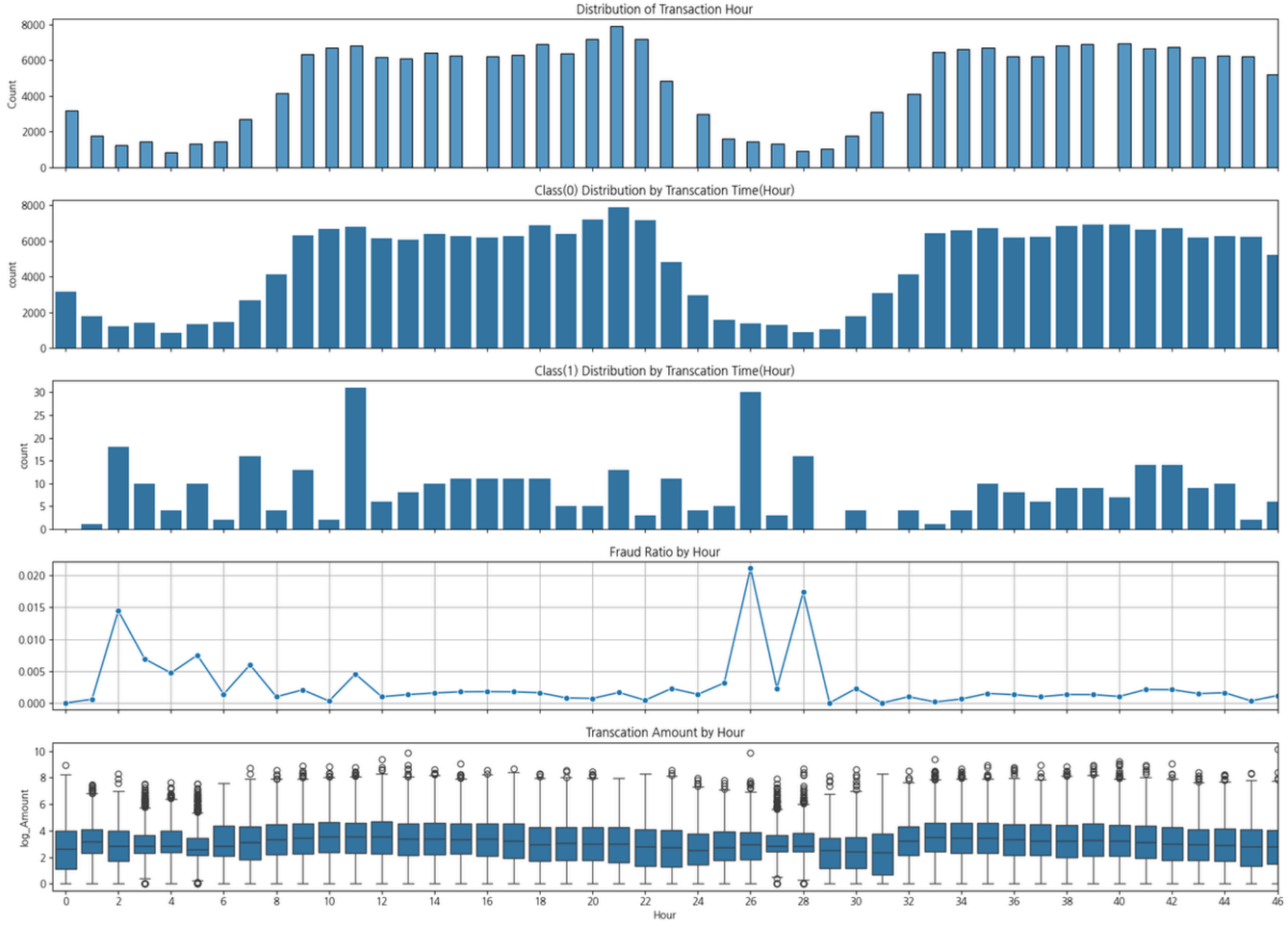

```python
# Class 분포 확인
sns.countplot(x='Class', data = train)
plt.title('Distribution of Class')
plt.savefig('class distribution.png')
plt.show()
print(train['Class'].value_counts())
```

Library	Function
seaborn, pandas	sns.countplot(), Series.value_counts()
Target 클래스 분포 시각화, 클래스별 샘플 수(불균형) 확인	

Target 변수인 'Class' column 의 비율이 매우 불균형함.

2. EDA

‘Time’ Variable Analysis

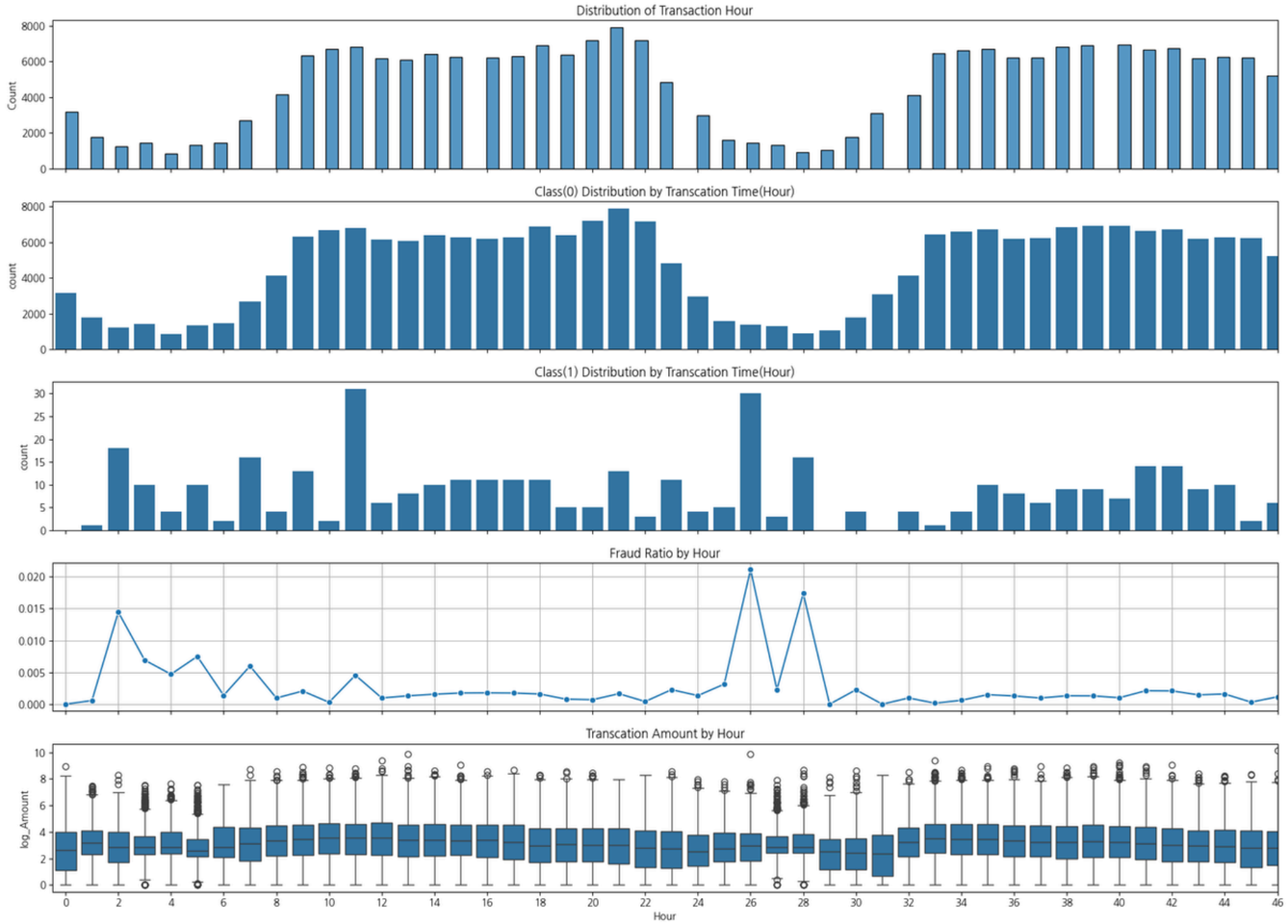



```python
# 시간대별 거래 수
# 시간 단위로 변환
train['Hour'] = (train['Time'] // 3600).astype(int)

fig, axes = plt.subplots(5, 1, figsize=(18, 13), sharex=True)

# 전체 거래 분포
sns.histplot(train['Hour'], bins=100, ax = axes[0])
axes[0].set_xticks(range(0, 48, 2))
axes[0].set_title('Distribution of Transaction Hour')

# 정상거래
sns.countplot(data = train[train['Class'] == 0], x='Hour', ax = axes[1])
axes[1].set_title('Class(0) Distribution by Transcation Time(Hour)')
axes[1].set_xticks(range(0, 48, 2))  # 2시간 간격
axes[1].tick_params(axis='x', rotation=45)

# 사기거래
sns.countplot(data = train[train['Class'] == 1], x='Hour', ax = axes[2])
axes[2].set_title('Class(1) Distribution by Transcation Time(Hour)')
axes[2].set_xticks(range(0, 48, 2))
axes[2].tick_params(axis='x', rotation=45)

# 시간대별 사기 비율
fraud_ratio = train.groupby('Hour')['Class'].mean()
sns.lineplot(x=fraud_ratio.index, y=fraud_ratio.values, marker='o', ax=axes[3])
axes[3].set_title('Fraud Ratio by Hour')
axes[3].grid(True)
axes[3].set_xticks(range(0, 48, 2))

# 시간대별 거래 금액
sns.boxplot(data = train, x='Hour', y='log_Amount', ax=axes[4])
axes[4].set_title('Transcation Amount by Hour')
axes[4].set_xticks(range(0, 48, 2))

plt.tight_layout()
plt.savefig('시간대별 거래 수.png')
plt.show()
```

Library	Function
seaborn, matplotlib.pyplot	histplot, countplot, boxplot, subplots, … etc
원본 데이터셋 train/test로 분리	

2. EDA

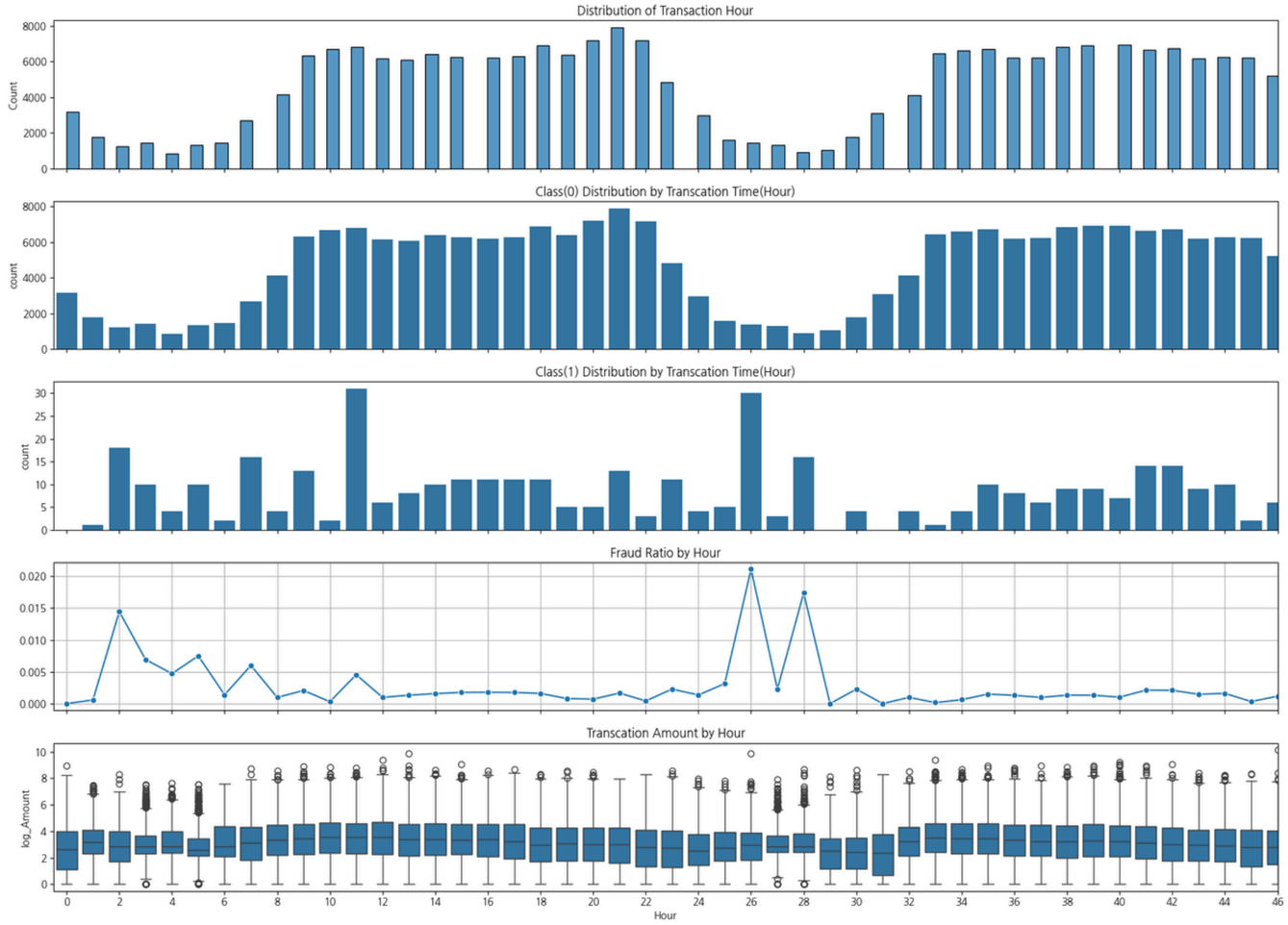
‘Time’ Variable Analysis



- **Distribution of Transaction Hour:** 전체 거래가 48시간 동안 어떤 시간대에 몰려 있는지 확인
- **Class(0) Distribution by Transaction Time(Hour):** 정상 거래만 시간축에 놓고 막대로 세어 봄 → 정상 활동 패턴을 이해
- **Class(1) Distribution by Transaction Time(Hour):** 데이터셋 첫 거래 이후 경과 시간별 사기 거래 분포 탐색
- **Fraud Ratio by Hour:** 경과 시간대별 사기 비율을 직관적으로 표시
- **Transaction Amount by Hour:** 시간대별 금액 편차·극단치 여부 확인

2. EDA

‘Time’ Variable Analysis



사기 거래 집중 시간대	<2·10·26시간 경과> 건수와 비율 모두 상승
고위험 구간	<2·26시간 경과> 전체 거래량은 적지만 사기 비율 높아 탐지 우선 대응 필요
안정적 구간	<10·20·42시간 경과> 거래량 많으나 사기 비율 낮아 정상 거래 주도
거래량 분포	로그 스케일 기준 거래량은 전반적으로 고르게 분포 거래량만으로 사기 여부 식별 제한

2. EDA

‘Amount’ Variable Analysis

```python
# 거래 금액 분포
sns.histplot(train['Amount'], bins=50, kde=True)
plt.title('Distribution of Transaction Amount')
plt.show()

# 로그 변환 후 확인
sns.histplot(np.log1p(train['Amount']), bins=50, kde=True)
plt.title('Distribution of Transaction Amount (log)')
plt.show()
```

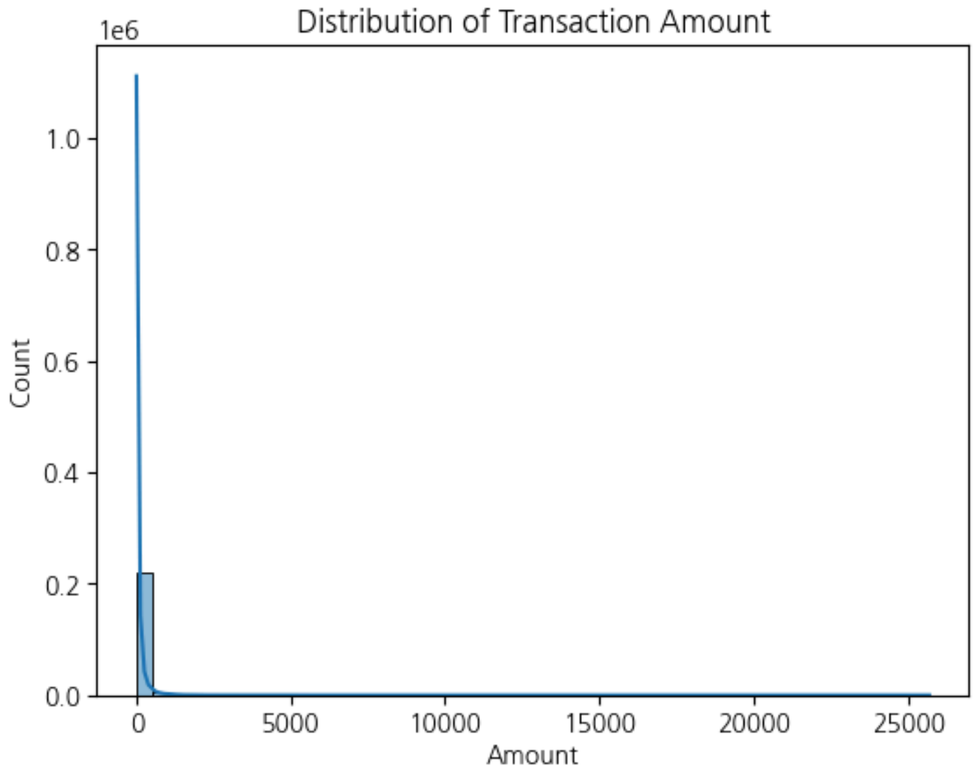
Library	Function
seaborn	histplot
금액 분포 파악 및 로그 변환 유용성 검증	

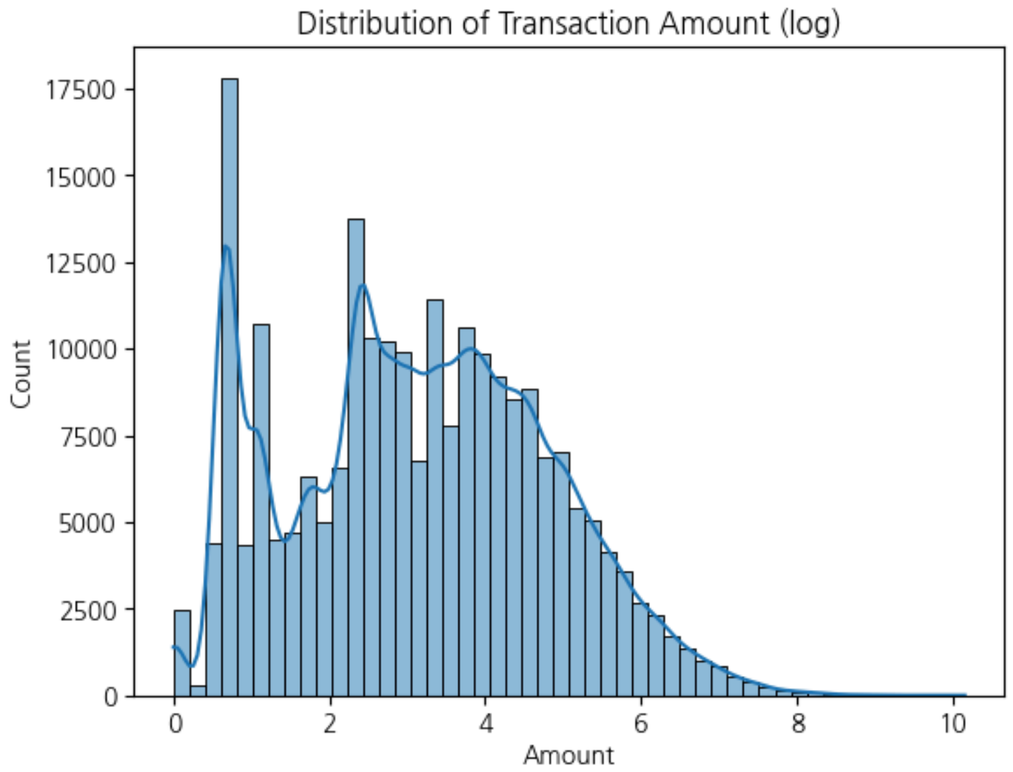
거래 금액의 분포를 히스토그램으로 시각화하고, 로그 변환 후 변화 확인

2. EDA

‘Amount’ Variable Analysis

출력결과



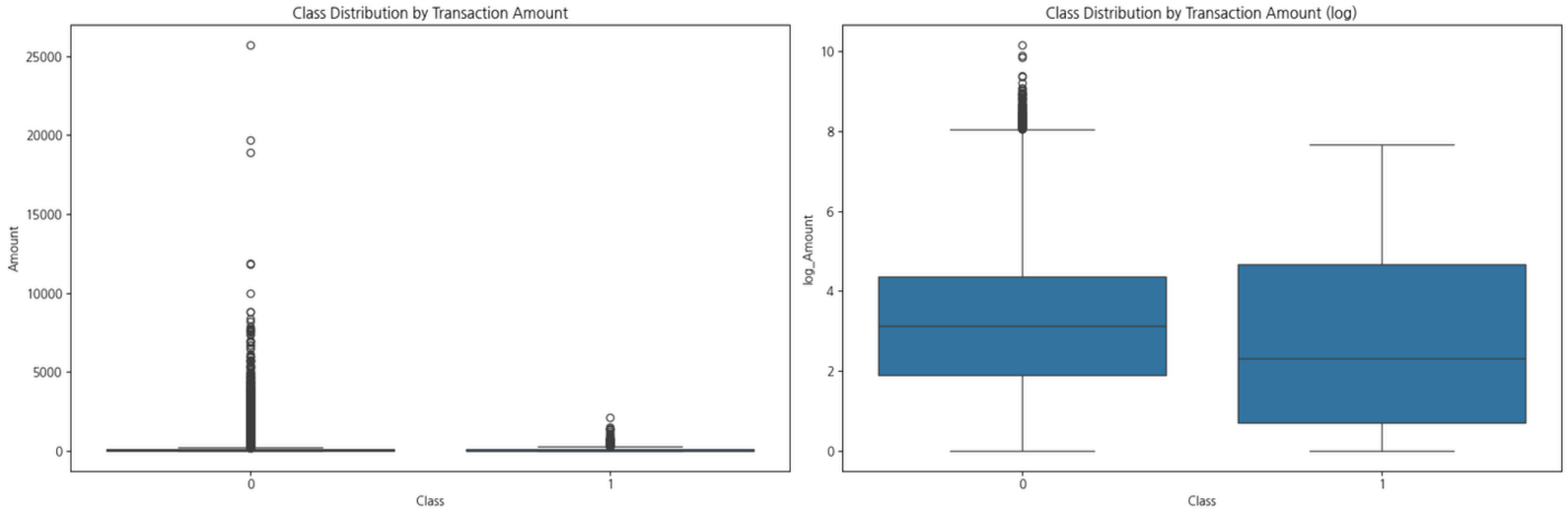


로그 변환 후 거래 금액 분포가 정규분포에 더욱 근접해짐.

2. EDA

‘Amount’ Variable Analysis

출력결과



사기 거래의 경우 거래 금액의 분산이 상대적으로 작고 거래 금액이 낮음.

2. EDA

PCA Features (V1 ~ V28 Analysis)

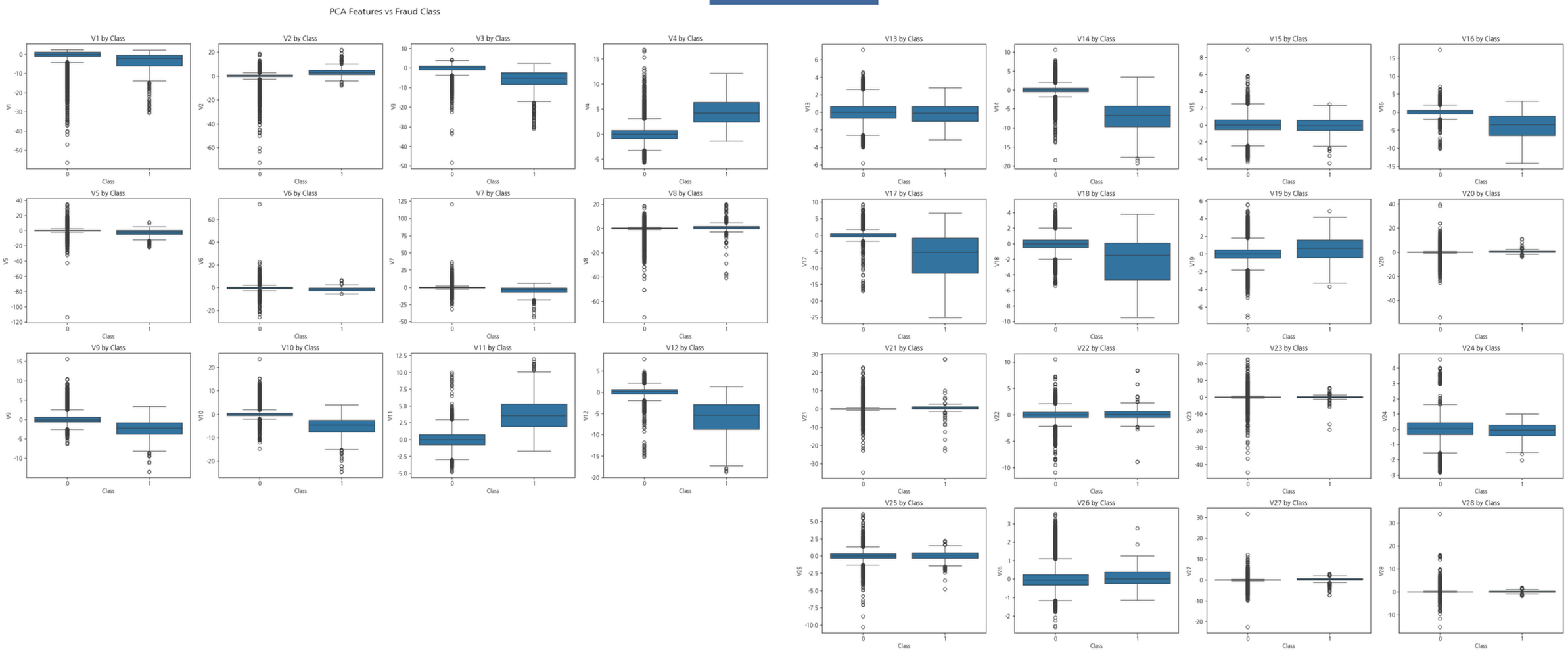
```python
# 분석 대상 특성들
pca_features = [f'V{i}' for i in range(1, 29)]

# 시각화 (boxplot)
n_cols = 4
n_rows = (len(pca_features) + n_cols - 1) // n_cols
plt.figure(figsize=(n_cols * 5, n_rows * 4))

for i, col in enumerate(pca_features):
    plt.subplot(n_rows, n_cols, i + 1)
    sns.boxplot(data=train, x='Class', y=col)
    plt.title(f'{col} by Class')
    plt.xlabel('Class')
    plt.ylabel(col)

plt.tight_layout()
plt.suptitle('PCA Features vs Fraud Class', fontsize=16, y=1.02)
plt.savefig('pca features vs fraud class.png')
plt.show()
```

Library	Function
matplotlib, seaborn	boxplot, subplot
PCA 주성분 값의 클래스별 분포 차이 일괄적으로 시각화	

2. EDA

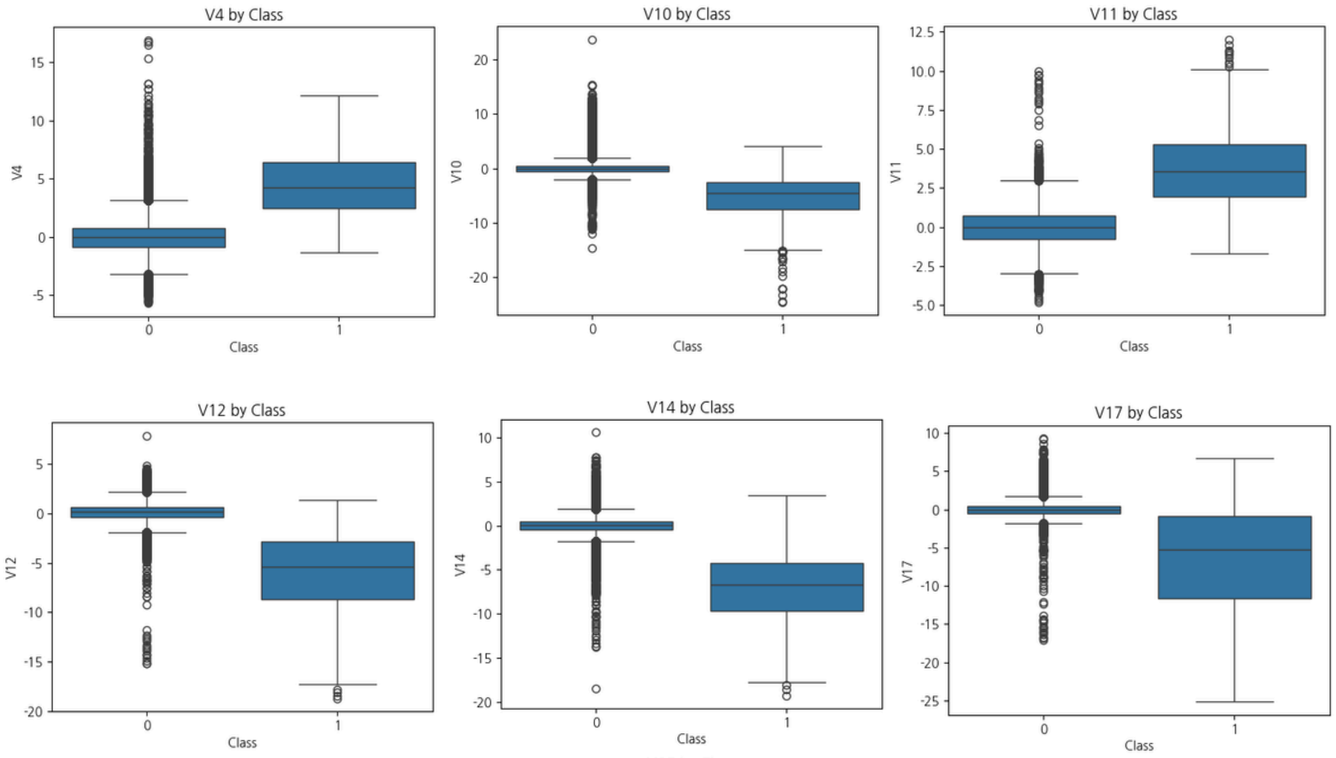
PCA Features (V1 ~ V28 Analysis)

출력결과



2. EDA

PCA Features (V1 ~ V28 Analysis)



사기거래와 정상거래 간의 분포가 확실히 드러나는 것을 볼 수 있음. (V4, V10, V11, V12, V14, V17)

3. Feature Engineering

3. Feature Engineering

Scaling - StandardScaler()

```
# Time, Amount 스케일링
df['Hour'] = (df['Time'] // 3600).astype(int)
df['Hour_mod'] = df['Hour'] % 24
df['Hour_sin'] = np.sin(2 * np.pi * df['Hour_mod'] / 24)

scaler = StandardScaler()
df['Hour_sin'] = scaler.fit_transform(df[['Hour_sin']])

#이상치인 경우 class1 비율이 높으므로 이러한 특성을 남겨 두기 위해 standard
df['Amount'] = np.log1p(df['Amount'])
scaler = StandardScaler()
df['Amount'] = scaler.fit_transform(df[['Amount']])
```

Library	Function
numpy, sklearn, pandas	sin, pi, log1p, StandardScaler
시간 주기성 보존, 금액 분포 대칭화, 표준화 → 안정성 및 성능 향상	

Time에서 Hour→sin파를 만들고, Amount를 로그+표준화하여 스케일을 맞춤.

3. Feature Engineering

What is Oversampling?

01 SMOTE (Synthetic Minority Over-sampling Technique)

- 소수 클래스(사기) 샘플을 k-NN 기반으로 합성하여 인위 생성

02 클래스 불균형 완화

- 소수 클래스 비율을 적절히 증가시켜 학습 안정성 확보

03 주요 효과

- 소수 클래스 샘플 수를 인위적으로 늘려 불균형 완화
- 모델이 희소한 패턴(사기 등)을 더 안정적으로 학습
- Precision·Recall 등 분류 성능 전반 향상 기대

Oversampling

```
from imblearn.over_sampling import SMOTE
# 오버샘플링
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
```

```
print(df.columns)
print(X_train.columns)
print(X_test.columns)
print(y_train.shape)
```

```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class', 'Hour', 'Hour_mod', 'Hour_sin'],
      dtype='object')
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Hour_sin'],
      dtype='object')
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Hour_sin'],
      dtype='object')
(227845,)
```

출력결과

Library	Function
imblearn	SMOTE
학습 데이터 균형을 맞추고 탐지 성능 향상	

4. Modeling & Evaluation

4. Modeling & Evaluation

Which Model to Use?

수업에서 다룬 모델들

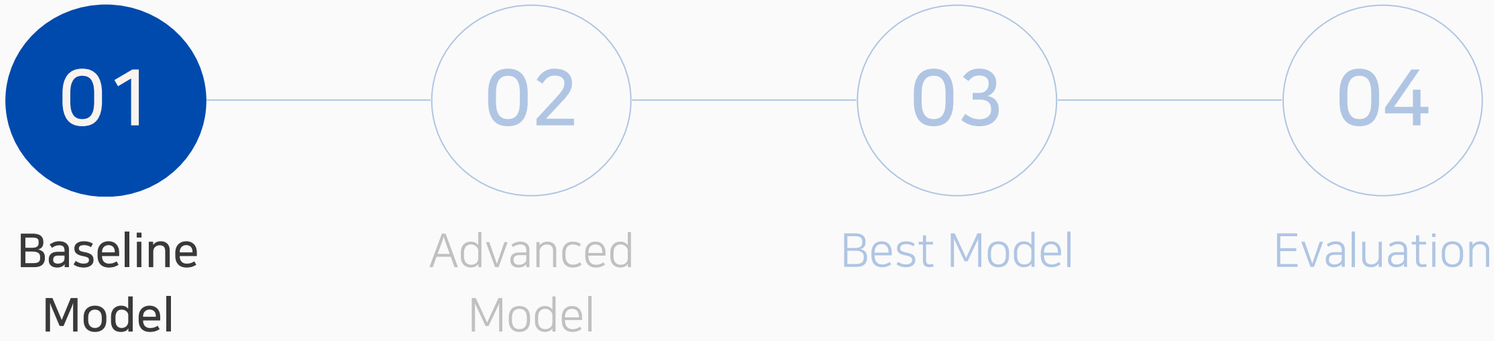
- LogisticRegression
- DecisionTreeClassifier
- RandomForestClassifier
- KNeighborsClassifier
- MLPClassifier (Neural Network)
- AdaBoost
- Bagging
- etc...

▶▶▶▶

전략

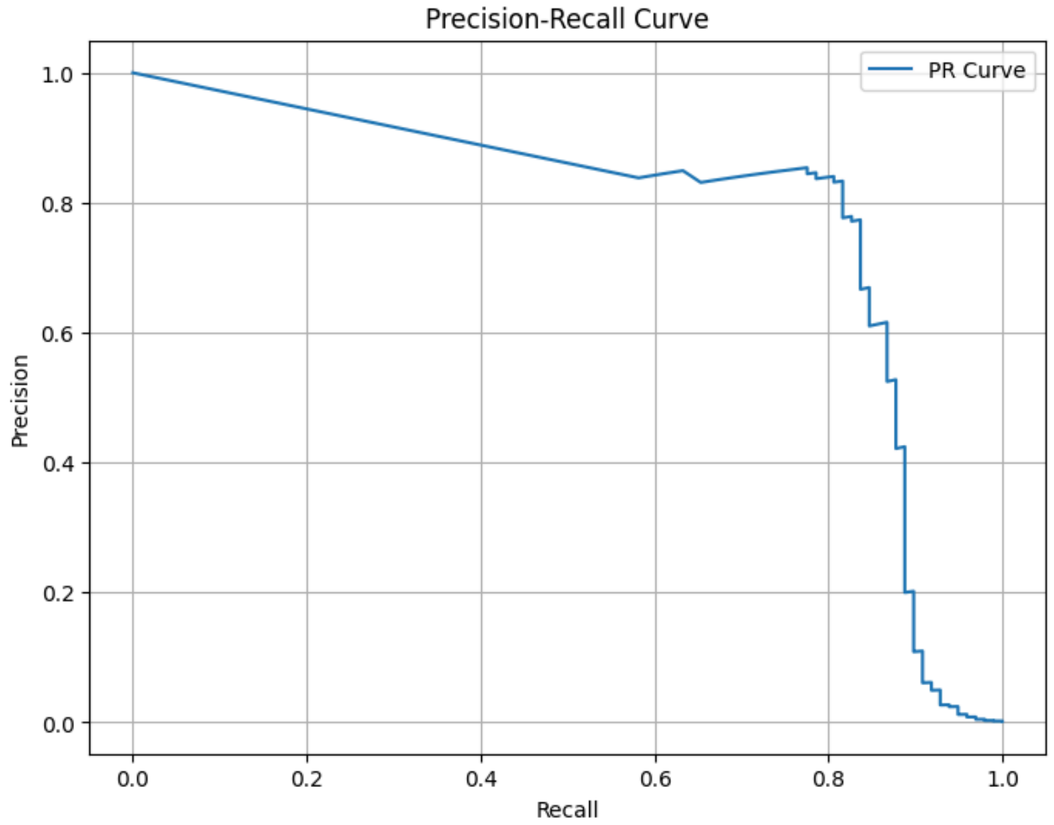
Base Line Model Training
↓
Advanced Model Training
↓
Best Model Selection
↓
Evaluation

4. Modeling & Evaluation



4. Modeling & Evaluation

Baseline Model - LogisticRegression(L1)




```python
# 로지스틱 회귀 - L1
lr = LogisticRegression(
    penalty='l1',
    C=1.0,                      # 규제 강도: 작을수록 규제 강해짐
    class_weight='balanced',  # 소수 클래스(사기)에 더 높은 패널티
    solver='liblinear',           # L1도 함께 테스트하면 'liblinear' or 'saga'
    random_state=42,
    max_iter=500,
)

# 모델 학습
lr.fit(X_train_resampled, y_train_resampled)

# 결과 출력
print_result(lr, 'Logistic Regression(L1)')
```

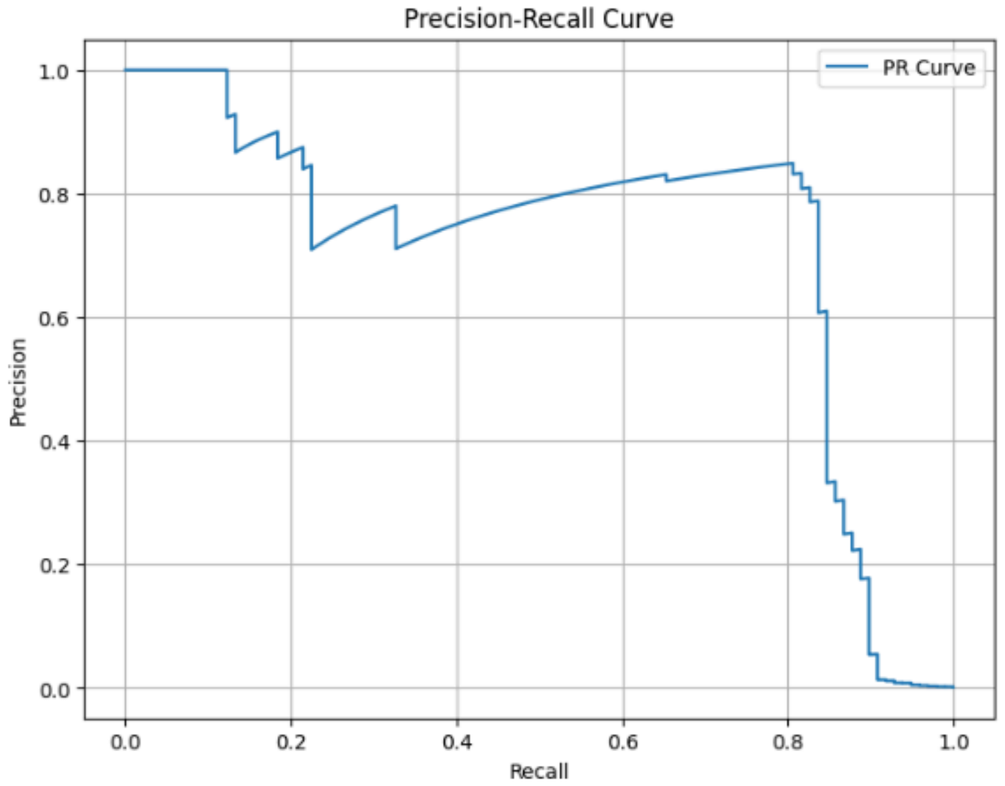
평가 지표	(0/1)
Accuracy	0.99
Precision	1.0/0.12
Recall	0.99/0.90
F1 Score	0.99/0.22
AUPRC	0.7822

Parameter
penalty = 'l1'
C = 1.0
class_weight = 'balance'
solver = 'libinear'
random_state = 42
max_iter = 500

Recall을 보았을 때 사기거래를 90% 잡았지만 Precision 12%로 오탐이 많음

4. Modeling & Evaluation

Baseline Model - LogisticRegression(L2)




```python
# 로지스틱 회귀 - L2
# 나언 코드 반영해서 규제, 파라미터 똑같이 반영
lr = LogisticRegression(
    penalty='l2',
    C=1.0,                    # 규제 강도: 작을수록 규제 강해짐
    class_weight='balanced',
    solver='liblinear',          # L1도 함께 테스트하면 'liblinear' or 'saga'
    random_state=42,
    max_iter=500,
)
# 모델 학습
lr.fit(X_train_resampled, y_train_resampled)

# 결과 출력
print_result(lr, 'Logistic Regression(L2)')
```

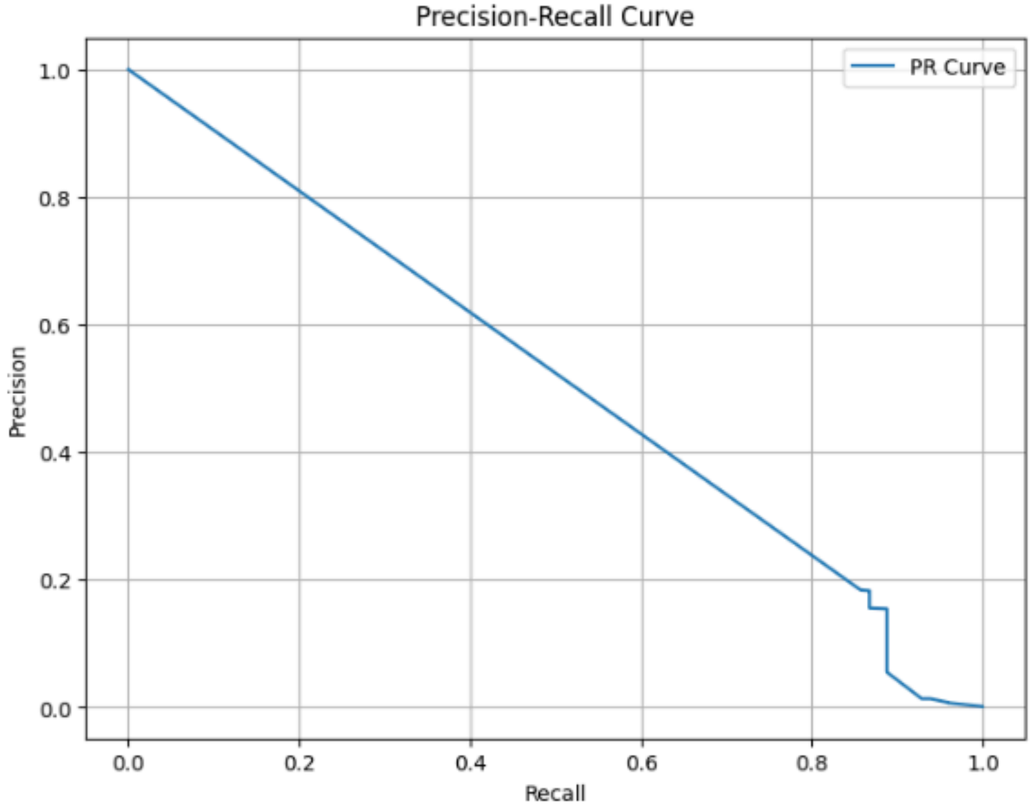
평가 지표	(0/1)
Accuracy	0.99
Precision	1.00/0.11
Recall	0.99/0.90
F1 Score	0.99/0.19
AUPRC	0.7171

Parameter
penalty = 'l2'
C = 1.0
class_weight = 'balance'
solver = 'libinear'
random_state = 42
max_iter = 500

Recall을 보았을 때 사기거래를 90% 잡았지만 Precision 11%로 오탐이 많음.

4. Modeling & Evaluation

Baseline Model - DecisionTree(Gini)




```python
# DecisonTree - Gini
dt = DecisionTreeClassifier(
    criterion='gini',
    random_state=42,
    max_depth=4
)

# 모델 학습
dt.fit(X_train_resampled, y_train_resampled)

# 결과 출력
print_result(dt, 'DecisionTreeClassifier(gini)')
```

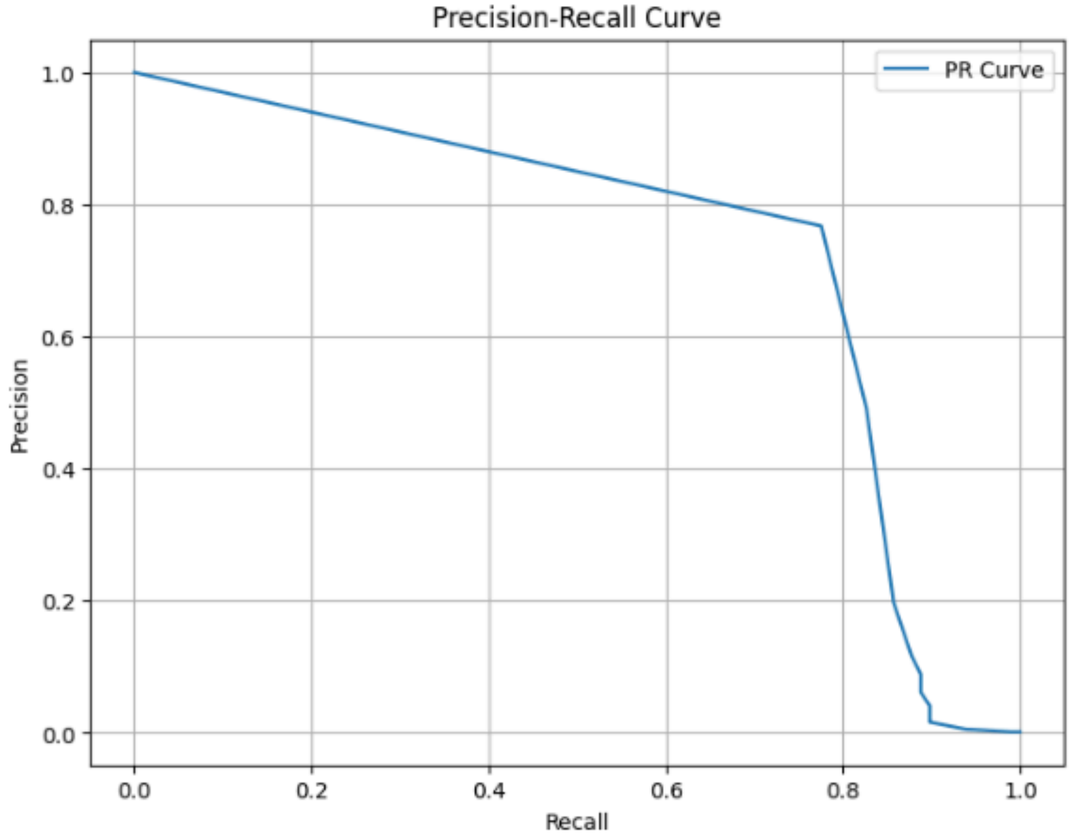
평가 지표	(0/1)
Accuracy	0.99
Precision	1.00/0.09
Recall	0.98/0.89
F1 Score	0.99/0.16
AUPRC	0.5145

Parameter
criterion = ‘gini’
max_depth = 4
random_state = 42

Recall을 보았을 때 사기 탐지율은 높지만 Precision 9%로 오탐이 과다함

4. Modeling & Evaluation

Baseline Model - DecisionTree(Entropy)




```python
# DecisonTree - Entropy
dt = DecisionTreeClassifier(
    criterion='entropy',
    random_state=42,
    max_depth=4
  )

# 모델 학습
dt.fit(X_train_resampled, y_train_resampled)

# 결과 출력
print_result(dt, 'DecisionTreeClassifier(entropy)')
```

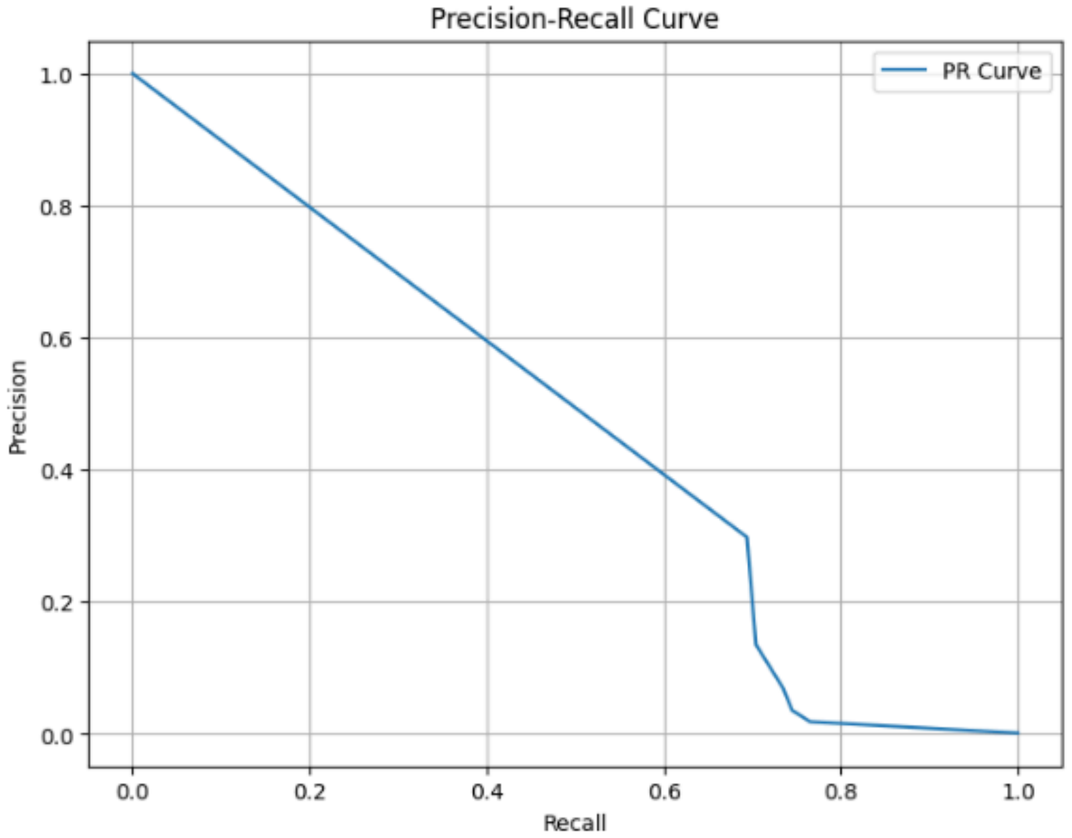
평가 지표	(0/1)
Accuracy	0.99
Precision	1.00/0.06
Recall	0.98/0.89
F1 Score	0.99/0.12
AUPRC	0.7335

Parameter
criterion = ‘entropy’
max_depth = 4
random_state = 42

Recall을 보면 사기 탐지는 유지됐지만 Precision이 9%에 그쳐 오탐이 많음.

4. Modeling & Evaluation

Baseline Model - KNN




```python
# KNN
knn = KNeighborsClassifier(
    n_neighbors=5,
    p=2,
    metric='minkowski'
)

# 모델 학습
knn.fit(X_train_resampled, y_train_resampled)

# 결과 출력
print_result(knn, 'KNN')
```

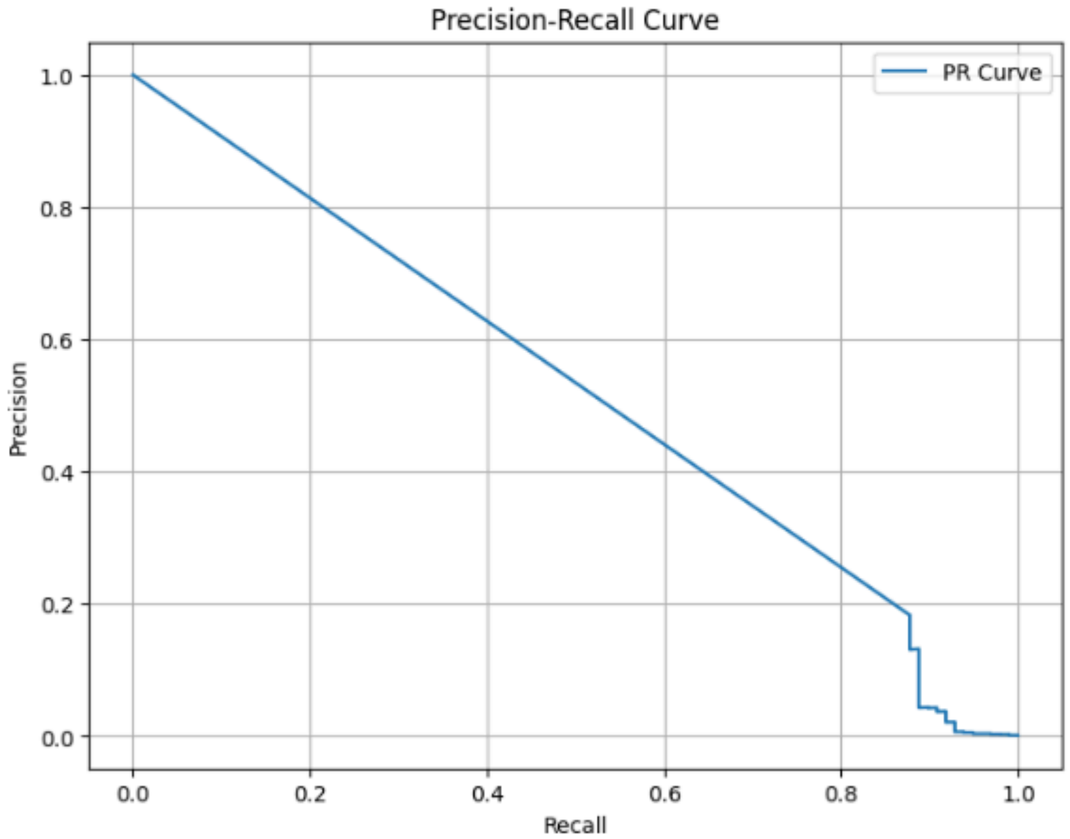
평가 지표	(0/1)
Accuracy	0.99
Precision	1.00/0.07
Recall	0.98/0.73
F1 Score	0.00/0.13
AUPRC	0.4593

Parameter
n_neighbors = 5
p = 2
metric = ‘minkowski’

Recall을 보면 사기 탐지율 73%를 달성했으나 Precision 7%로 오탐율이 지나치게 높음.

4. Modeling & Evaluation

Baseline Model - MLPClassifier




```python
# MLPClassifier
mlp = MLPClassifier(
    hidden_layer_sizes=(100,),   # 은닉층 구조: 예시로 히든 노드 100개짜리 한 층
    activation='relu',           # 활성화 함수 (ReLU)
    solver='adam',               # 최적화 알고리즘 (Adam)
    alpha=0.0001,                # L2 정규화 (weight_decay)
    learning_rate='adaptive',    # 학습률 조정 방식
    max_iter=200,                # 최대 반복(epoch) 수
    random_state=42,
    early_stopping=True,    # 개선 없으면 자동으로 멈춤
)
mlp.fit(X_train_resampled, y_train_resampled)

# 결과 출력
print_result(mlp, 'MLPClassifier')
```

평가 지표	(0/1)
Accuracy	0.99
Precision	1.00/0.08
Recall	0.98/0.89
F1 Score	0.99/0.15
AUPRC	0.5224

Parameter
hidden_layer_sizes = (100,)
activation = ‘relu’
solver = ‘adam’
alpha = 0,0001
learning_rate = ‘adaptive’
max_iter = 200
random_state = 42
early_stopping = True

Recall을 보았을 때 사기 거래는 높은 비율로 포착되지만, Precision이 8%로 오탐이 과하게 발생함.

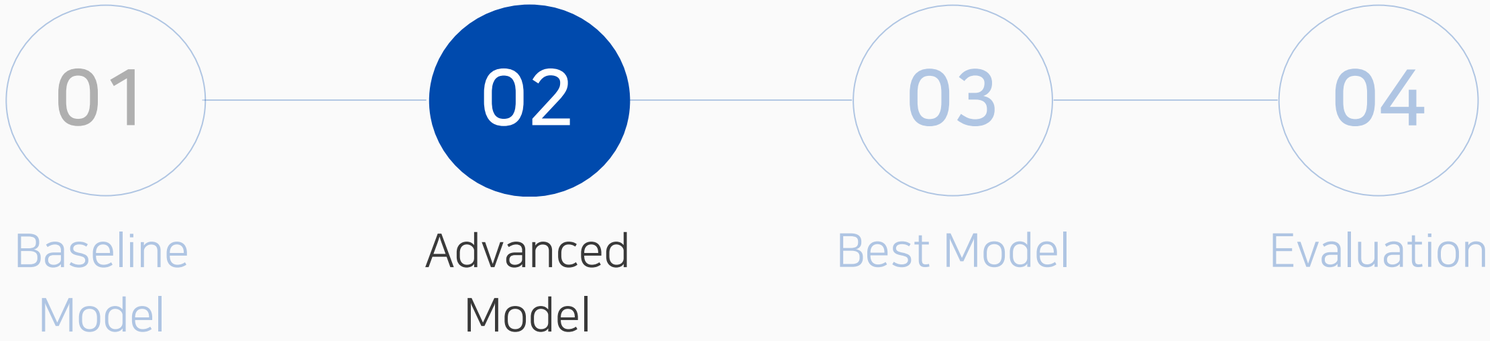
4. Modeling & Evaluation

Baseline Model

Model	AUPRC (Area Under PR Curve)
LogisticRegression (L1 정규화)	0.7822
LogisticRegression (L2 정규화)	0.7171
DecisionTree - Gini	0.5145
DecisionTree - Entropy	0.7335
KNN	0.4593
MLP Classifier	0.5224

L1 규제를 적용한 로지스틱 회귀 모델이 가장 뛰어난 성능을 보임.

4. Modeling & Evaluation

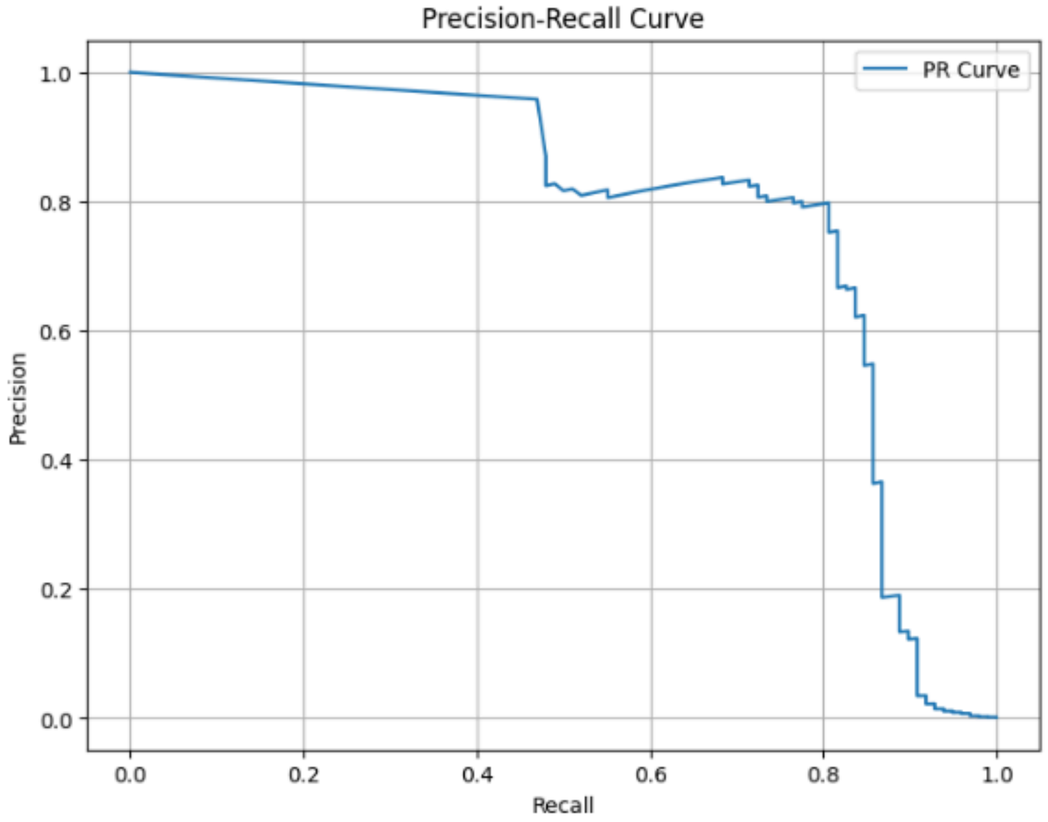


Advanced Model - Ensemble (Voting Classifier)

```python
model = [('lr', lr), ('dt', dt), ('mlp', mlp)]
# 투표방식: soft
voting_soft = VotingClassifier(
    estimators=model, #('knn', knn)
    voting='soft',
    n_jobs=-1
)

# 투표방식: hard
voting_hard = VotingClassifier(
    estimators=model, #('knn', knn)
    voting='hard',
    n_jobs=-1
)

voting_soft.fit(X_train_resampled, y_train_resampled)
voting_hard.fit(X_train_resampled, y_train_resampled)

print_result(voting_soft, 'VotingClassifier(soft)')
print_result(voting_hard, 'VotingClassifier(hard)')
```

모델의 다양성을 고려하여
Logistic, DecisionTree(entropy), MLPClassifier를
VotingClassifier로 앙상블 진행

Advanced Model - Ensemble (Voting Classifier / voting = ‘soft’)



평가 지표	(0/1)
Accuracy	0.99
Precision	1.00/0.13
Recall	0.99/0.90
F1 Score	0.99/0.23
AUPRC	0.7802

Parameter
estimators =[(‘lr’, lr), (‘dt’, dt), (‘mlp’, mlp)]
voting = ‘soft’
n_jobs = -1

높은 탐지율(Recall)과 높은 AUPRC를 달성했으나 Precision은 13%에 그침.

Advanced Model - Ensemble (Voting Classifier / voting = ‘hard’)

```
y_pred_hard = voting_hard.predict(X_test)
print("Hard Voting Report:₩n", classification_report(y_test, y_pred_hard))
print("Confusion Matrix:₩n", confusion_matrix(y_test, y_pred_hard))
```

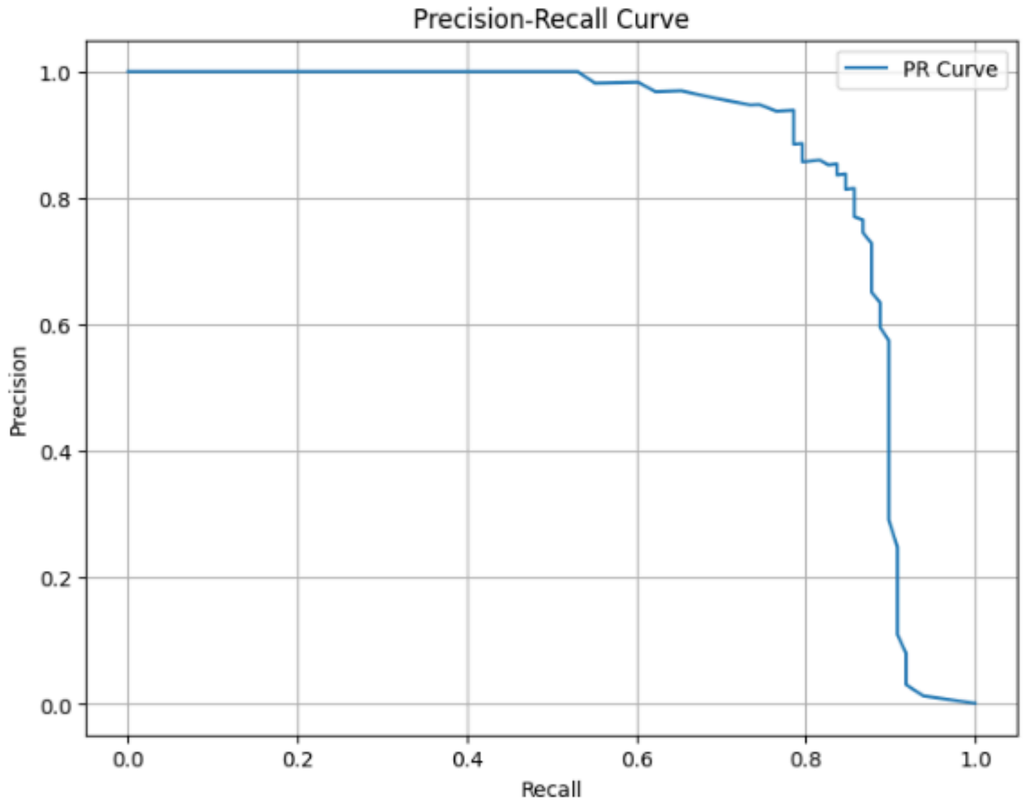
평가 지표	(0/1)
Accuracy	0.99
Precision	1.00/0.13
Recall	0.99/0.90
F1 Score	0.99/0.22
AUPRC	-

Parameter
estimators =[(‘lr’, lr), (‘dt’, dt), (‘mlp’, mlp)]
voting = ‘hard’
n_jobs = -1

Soft voting 방식이 주요 평가 지표에서 소폭 높은 성능을 보여 Soft voting 방식 선택

4. Modeling & Evaluation

Advanced Model - RandomForest




```python
# 랜덤포레스트
rf = RandomForestClassifier(
    criterion='entropy',
    random_state = 42,
    n_estimators=100,
    n_jobs=-1,
    class_weight="balanced")

# 모델학습
rf.fit(X_train_resampled, y_train_resampled)

# 결과 출력
print_result(rf, 'RandomForest')
```

평가 지표	(0/1)
Accuracy	0.99
Precision	0.99/0.84
Recall	0.99/0.83
F1 Score	0.09/0.84
AUPRC	0.8702

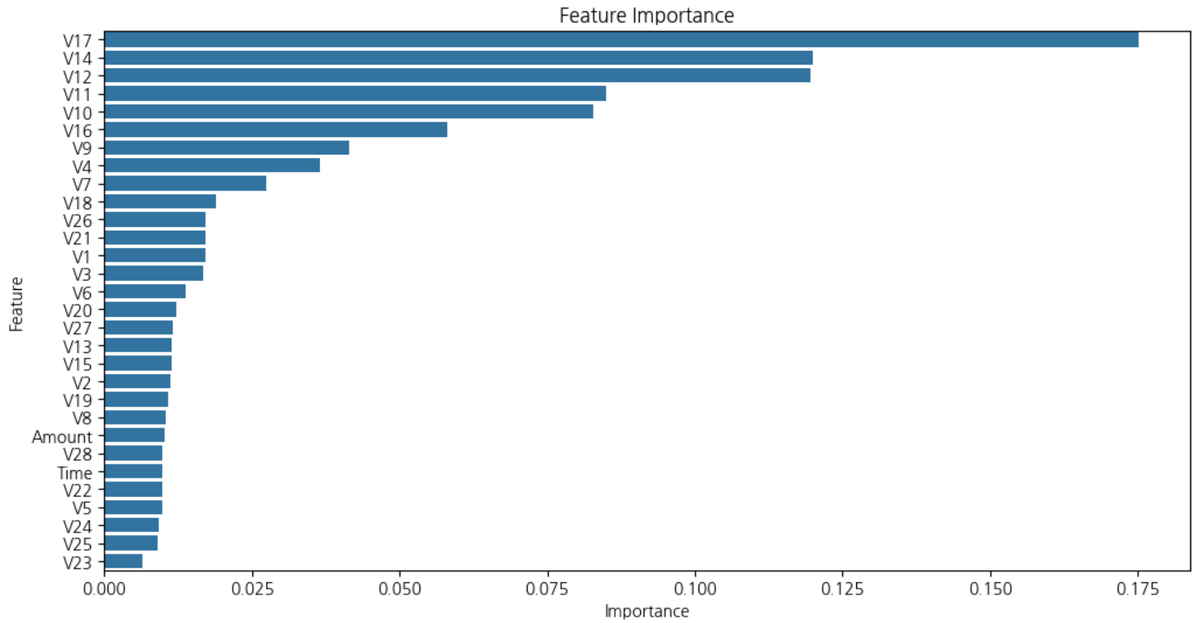
Parameter
criterion = 'entropy'
n_estimators = 100
random_state = 42
class_weight = 'balanced'
n_jobs = -1

Precision 84%로 사기를 정확히 탐지하면서 오탐과 누락을 모두 최소화

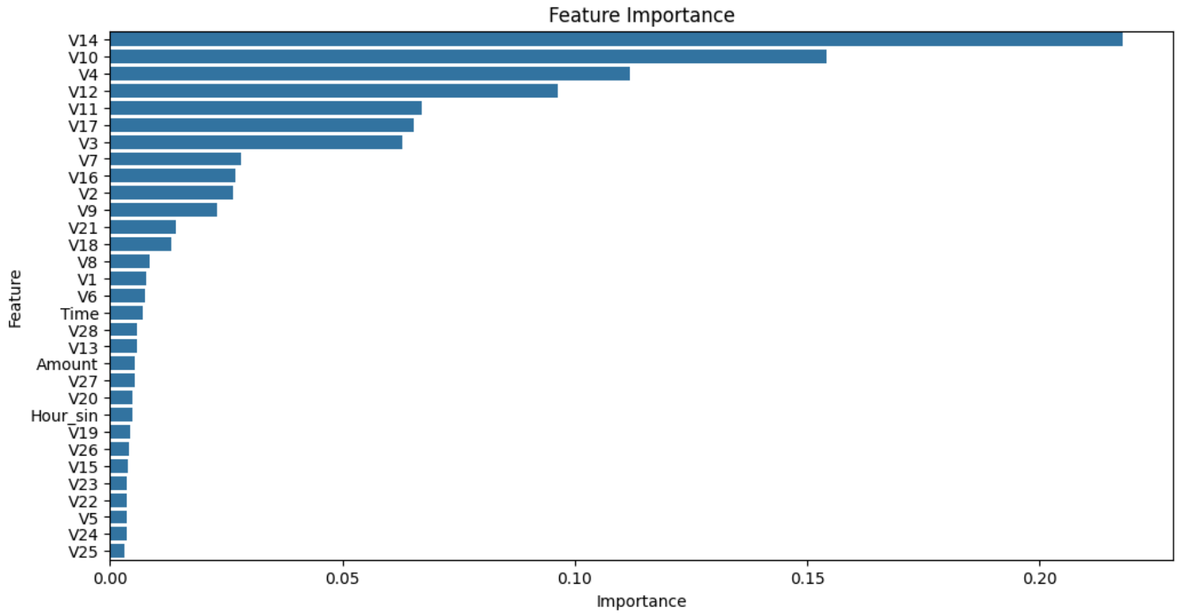
4. Modeling & Evaluation

Advanced Model - RandomForest

전처리하지 않은 원본 데이터 기반 RandomForest의 featureimportances



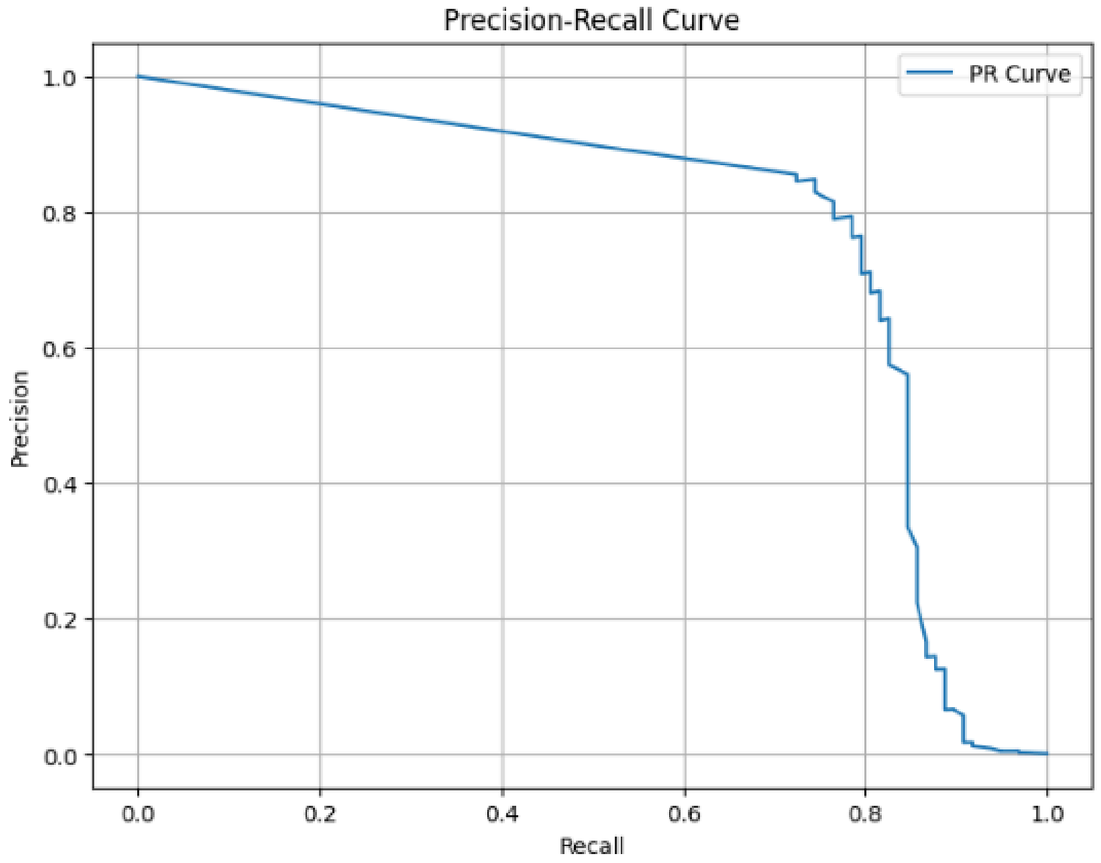
전처리를 거친 원본 데이터 기반 RandomForest의 featureimportances



- 거래 금액(Amount)과 주성분 V14·V10 등의 중요도가 크게 상승
- 피처 간 스케일 차이가 완화되고 소수 클래스 정보가 보강되어, 모델이 핵심 변수의 신호를 효과적으로 학습했기 때문으로 보임.

4. Modeling & Evaluation

Advanced Model - Bagging



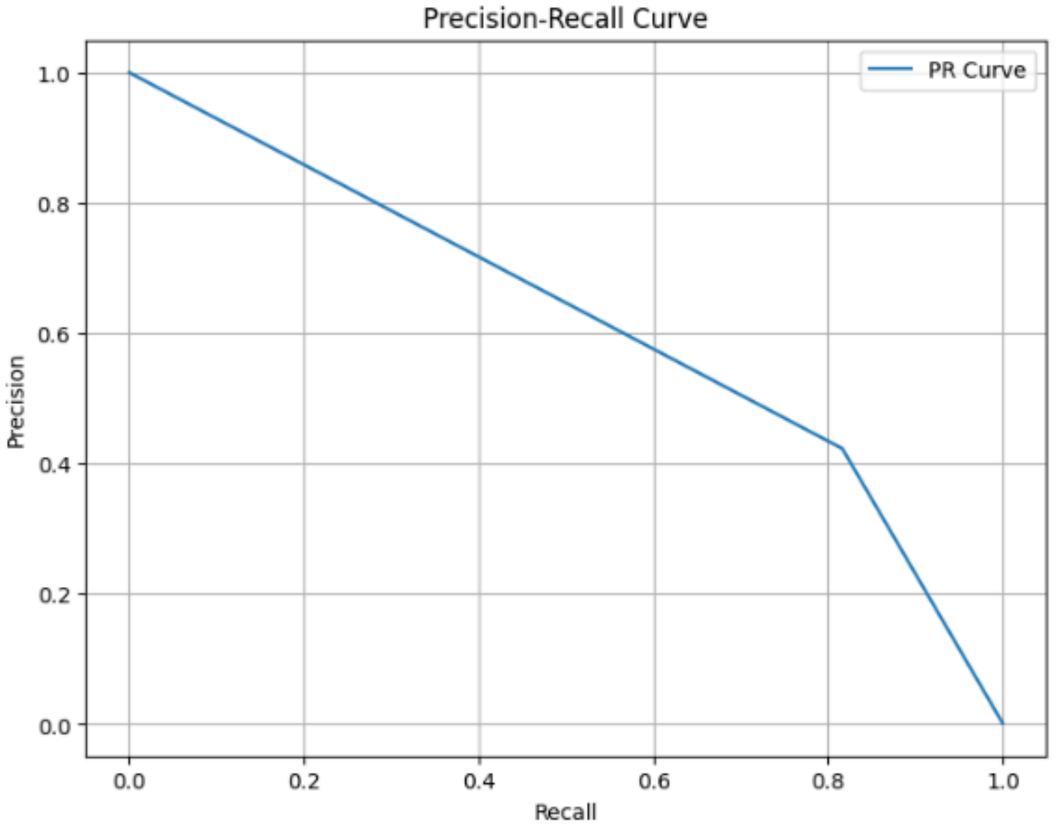

```python
# Bagging
bag = BaggingClassifier(
    estimator=DecisionTreeClassifier(  # 기본 추정기로 결정 트리 사용
        criterion='entropy',           # 정보 이득(Entropy) 기준으로 분할
        random_state=42,               # 재현성을 위한 랜덤 시드 고정
        max_depth=4                    # 트리 최대 깊이 제한으로 과적합 방지
    ),
    n_estimators=100,                  # 앙상블에 사용할 트리(추정기) 개수
    random_state=42,                   # 부트스트랩 샘플링 시드 고정
    max_samples=0.5,                   # 각 트리 학습에 사용될 훈련 샘플 비율 (50%)
    n_jobs=-1,                         # 모든 CPU 코어를 사용하여 병렬 학습
    bootstrap=True                     # 복원 추출 방식(중복 허용)으로 샘플링
)

# 모델학습
bag.fit(X_train_resampled, y_train_resampled)

# 결과출력
print_result(bag, 'Bagging')
```

평가 지표	(0/1)
Accuracy	0.97
Precision	0.99/0.06
Recall	0.97/0.89
F1 Score	0.98/0.12
AUPRC	0.772

Parameter
base_estimator = DecisionTreeClassifier()
n_estimators = 100
max_samples = 1.0
bootstrap = True
oob_score = True
random_state = 42

사기 탐지(Recall)는 우수하나, Precision이 6%에 그쳐 오탐이 과도함.

Advanced Model - AdaBoost




```python
# AdaBoost
ada = AdaBoostClassifier(
    estimator=DecisionTreeClassifier(random_state=42),
    n_estimators=100,
    random_state=42
)

# 모델 학습
ada.fit(X_train_resampled, y_train_resampled)

# 결과출력
print_result(ada, 'AdaBoostClassifier')
```

평가 지표	(0/1)
Accuracy	0.99
Precision	0.99/0.42
Recall	0.99/0.81
F1 Score	0.99/0.55
AUPRC	0.62

Parameter
estimator = DecisionTreeClassifier()
n_estimators = 100
learing_rate = 1.0
random_state = 42

사기 탐지율(Recall) 82%, Precision 42%

4. Modeling & Evaluation

Advanced Model

Model	AUPRC (Area Under PR Curve)
Ensemble (VotingClassifier)	0.7802
RandomForestClassifier	0.8702
Bagging	0.772
AdaBoost	0.62

RandomForest의 성능이 가장 뛰어남.

4. Modeling & Evaluation

Best Model

* Model

- RandomForestClassifier + SMOTE(1:1)

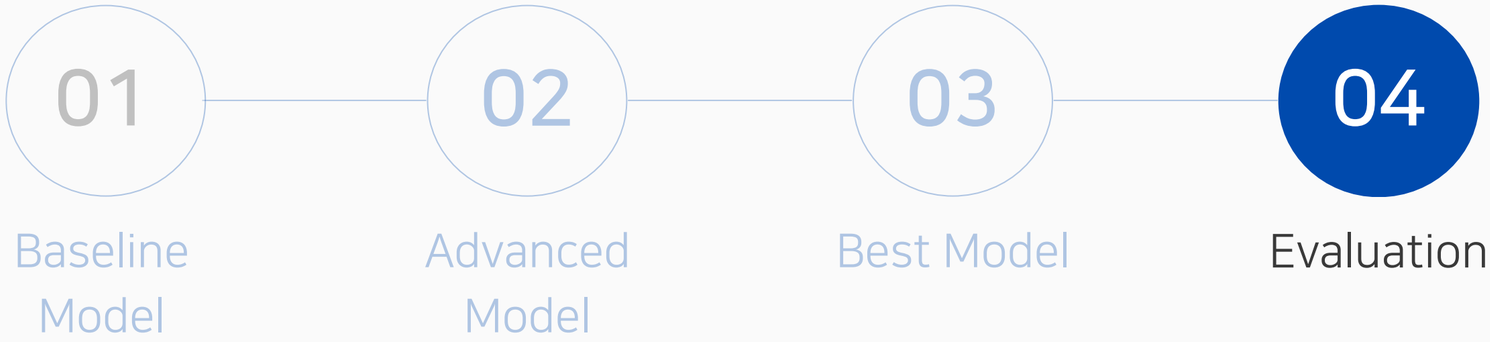
* Feature Engineering

- SMOTE 오버샘플링(사기:정상 = 1:1)
- StandardScaler 적용

* Hyperparameters

- n_estimators=100, class_weight='balanced', random_state=42

4. Modeling & Evaluation



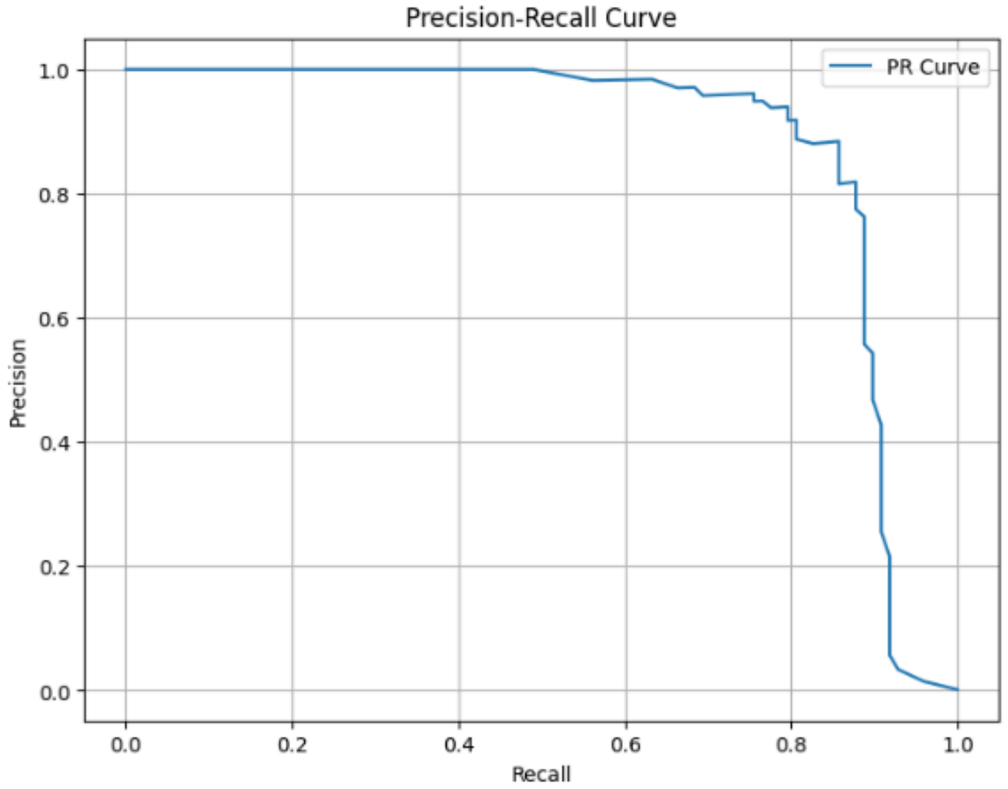
Evaluation (1) – Over Sampling Adjustment

```python
# 나언 코드 참고
# 오버샘플링 변경, 스케일링 수정(위의 코드에서 변경함)
smote = SMOTE(sampling_strategy=0.1,random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# 오버샘플링 확인
print("Resampled X shape:", X_train_resampled.shape)
print("Resampled y shape:", y_train_resampled.shape)
print("Resampled y Series name:", y_train_resampled.name)      # Series 이름 확인
print("Resampled y index (일부):", y_train_resampled.index[:5])  # 인덱스 일부 샘플
```

```
Resampled X shape: (250196, 31)
Resampled y shape: (250196,)
Resampled y Series name: Class
Resampled y index (일부): RangeIndex(start=0, stop=5, step=1)
```

사기:정상 비율을 1:462에서 1:10으로 보정하여 희소 클래스 학습 안정성 확보

Evaluation (1) – Over Sampling Adjustment



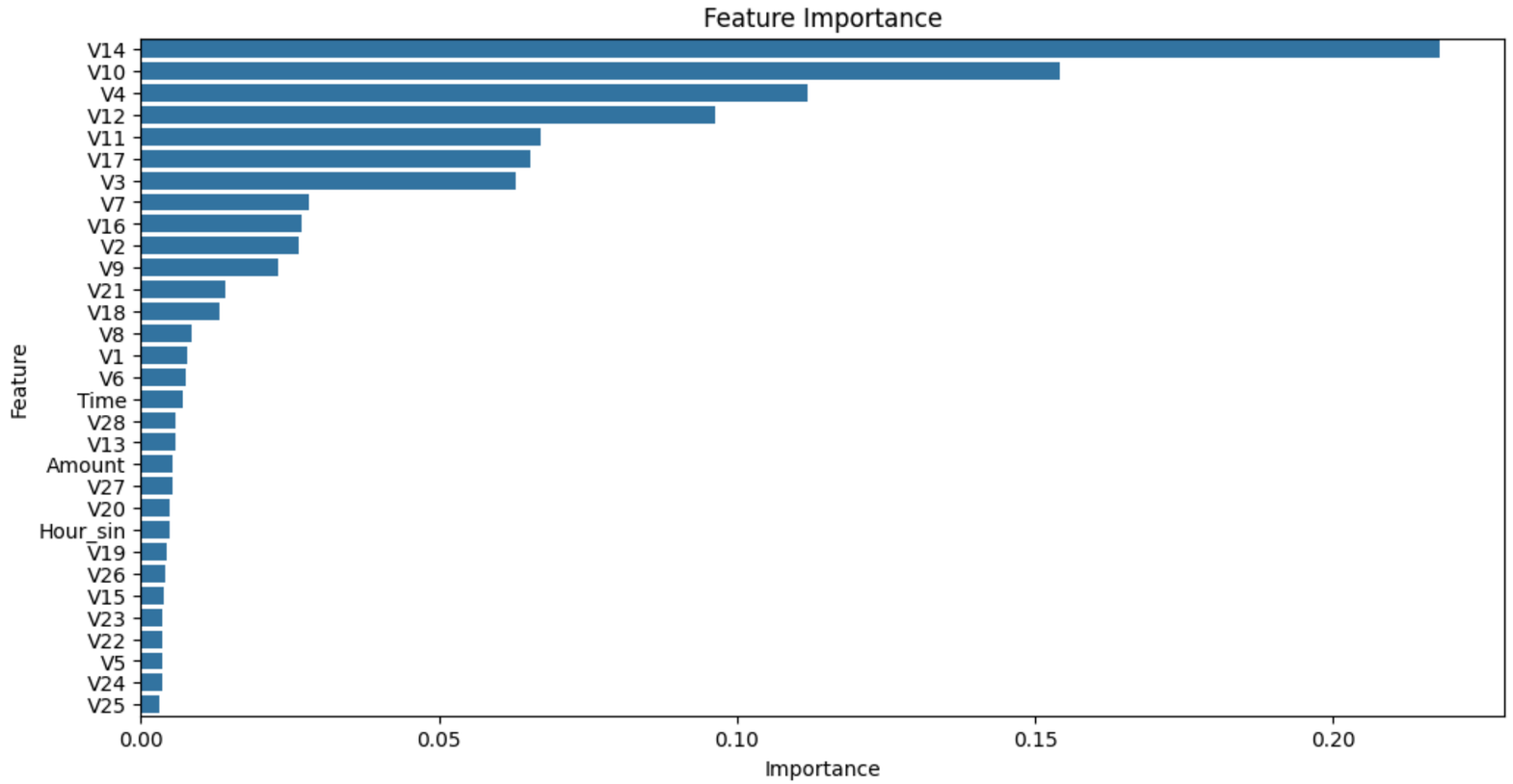

```python
# ver1. 오버샘플링 비율만 변경
# 랜덤포레스트
rf_ver1 = RandomForestClassifier(random_state = 42,
                                 n_estimators=100,
                                 n_jobs=-1,
                                 class_weight="balanced"
)

rf.fit(X_train_resampled, y_train_resampled)

# 결과 출력
print_result(rf_ver1, 'RandomForest-Evaluation_ver1')
```

평가 지표	(0/1)
Accuracy	0.99
Precision	0.99/0.85
Recall	0.99/0.85
F1 Score	0.99/0.85
AUPRC	0.8807

Parameter
n_estimators = 100
random_state = 42
class_weight = ‘balanced’
n_jobs = -1

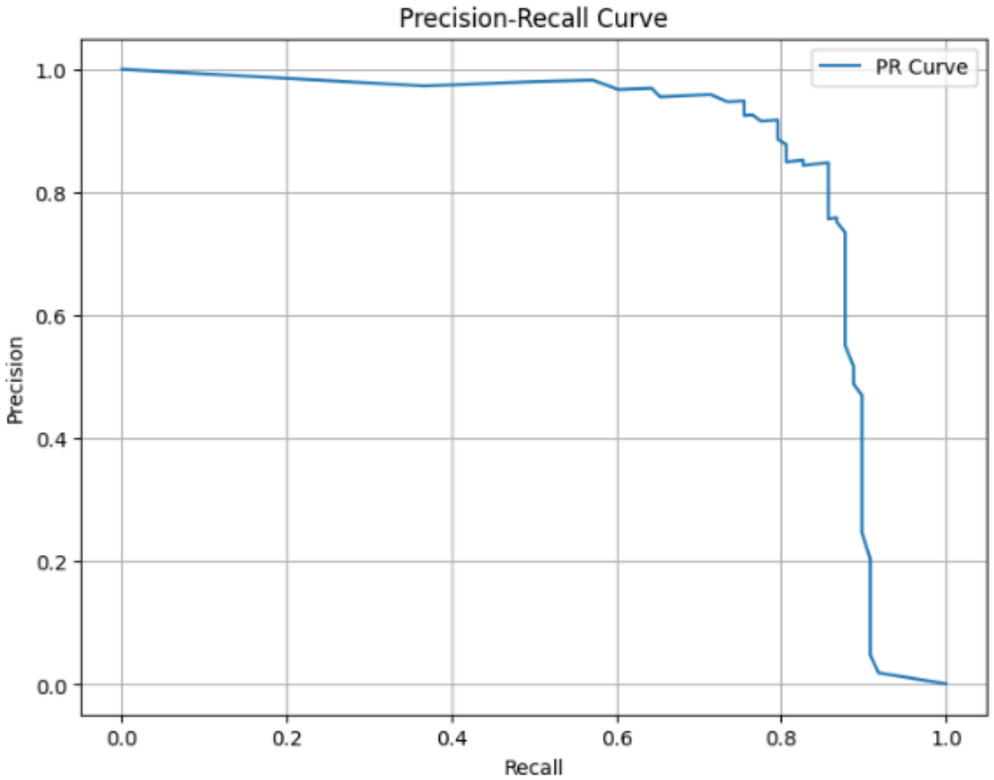
오버샘플 비율 튜닝으로 랜덤포레스트가 Precision·Recall 모두 86% 대로 균형 잡힌 최고 성능을 달성

Evaluation (2) – Over Sampling & Feature Adjustment



중요도가 낮은 피쳐 리스트: “V25”, “V24”, “V5”, “V22”, “V23”, “V8” (선정 방식: rf.feature_importances_)

Evaluation (2) – Over Sampling & Feature Adjustment




```python
# ver2. 오버샘플링 비율 변경 + 중요도 낮은 피처 상위 6개 제외하고 학습 진행
# 나언 코드 참고
# 3) 중요도가 낮아 보이는 피처 리스트(예: V23, V25, V24, V5, V22, V8, V19, V2, V15, V13, V27, V20, V6, V3) 정의
drop_cols = [
    "V23", "V25", "V24", "V5", "V22", "V8"
]

# 4) 불필요한 컬럼 드롭
X_reduced = X_train_resampled.drop(columns=drop_cols)
X_reduced_test = X_test.drop(columns=drop_cols)

# BestModel: RandomForest로 학습 진행
rf_ver2 = RandomForestClassifier(random_state = 42,
                                 n_estimators=100,
                                 n_jobs=-1,
                                 class_weight="balanced")

# 모델학습
rf.fit(X_reduced, y_train_resampled)

# 결과출력
print_result(rf_ver2, 'RandomForest-Evaluation_ver2')
```

평가 지표	(0/1)
Accuracy	0.99
Precision	0.99 / 0.84
Recall	0.99 / 0.83
F1 Score	0.99 / 0.84
AUPRC	0.8576

Parameter
n_estimators = 100
random_state = 42
class_weight = ‘balanced’
n_jobs = -1

불필요해 보이는 피처를 제거해 모델을 경량화 했으나 오히려 성능 하락함.

5. Conclusion

5. Conclusion

Optimal Model

* Model

- RandomForestClassifier + SMOTE(1:10)

* Feature Engineering

- SMOTE 오버샘플링(사기:정상 = 1:10)
- StandardScaler 적용

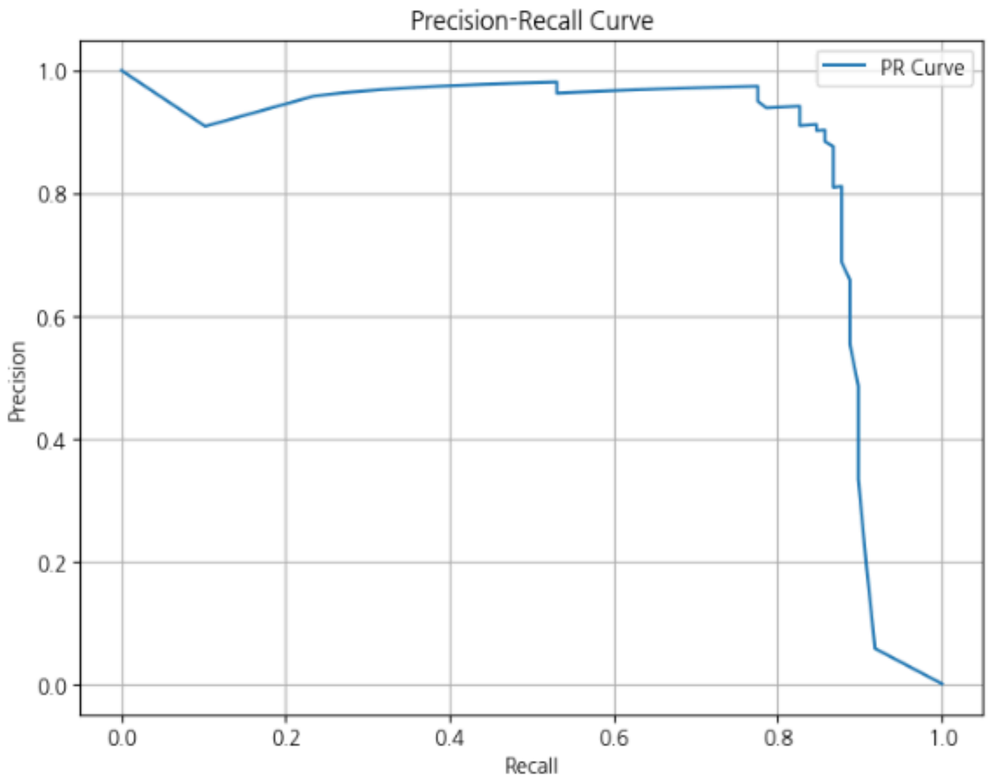
* Hyperparameters

- n_estimators=100, class_weight='balanced'

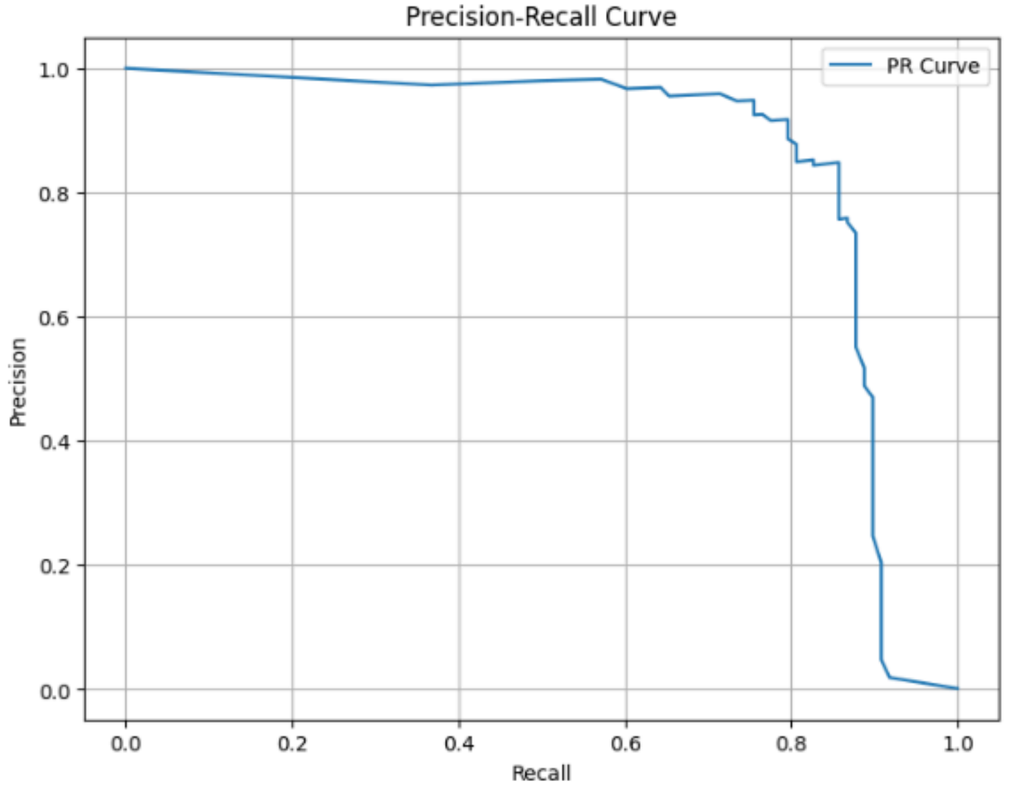
5. Conclusion

Optimal Model

전처리하지 않은 원본 데이터 기반 RandomForest (성능: 0.8582)



RandomForest + 오버샘플링 비율 변경 (성능 : 0.8807)



오버샘플링 비율 조정으로 AUPRC가 0.8582 → 0.8807로 향상되어,
RandomForest + SMOTE(1:10) 모델을 최종적으로 선정.

5. Conclusion

Results

01 Evaluation 요약

- 1. SMOTE 비율 변경(1:462→1:10) → AUPRC 0.8582 → 0.8807로 개선
- 2. SMOTE + 피처 개수 조절 → 추가 성능 향상 x

02 최종 모델 & 핵심 인사이트

- 최적 모델: RandomForest + SMOTE(1:10)
- class_weight='balanced'를 적용해 클래스 불균형을 보강함으로써 모델의 사기 탐지 성능 향상에 기여
- Feature 제거 시 성능 하락 → 해당 Feature들이 핵심 정보 보유하고 있음을 알 수 있음.

5. Conclusion

Discussion

01 피처엔지니어링 심화 부족

- Amount × Time 조합 피처 등 추가 생성해 보지 못함.
- 이상치 존재 여부 및 이상치 개수 기반 피처를 학습에 반영하지 못함.

02 비용 민감 학습 미실시

- precision-Recall trade-off를 위한 임계값 조정, 또는 비용 민감(Class Weight, Cost-Sensitive) 기법 실험이 부족함.

감사합니다
Thank you

Q&A