프로브 프로젝트 3조
2024-06-03

신용위험 평가
채무불이행 분석 모델

목차

01 데이터 - 데이터 분석 주제

신용 위험 평가 모델 개발

대출 신청자의 다양한 정보를 사용하여 채무 불이행 여부를 예측하는 모델 개발.
다양한 데이터 파일과 변수들을 활용하여 신뢰성 있는 예측 모델 구축이 목표

01 데이터 - 데이터 분석 주제

채무 불이행자란?

은행이나 신용카드사 등의 제도권 금융기관에서 돈을 빌린 후 갚지 않고 일정기간 연체한 자.

채무 불이행이 가져오는 위험은?

- 금융기관의 손실 증가

: 채무 불이행으로 인해 금융기관은 대출금 회수가 어려워지고
이는 이자율 상승이나 자금 조달 곤란 등 문제로 이어질 수 있음

- 국가 경제적 파장

: 대규모 채무 불이행이 발생할 경우 국가 전체적인 경제에 악영향.
개인의 소비 감소, 기업의 투자 감소, 실업률 상승 등의 현상 발생 가능.
=> 이는 GDP 성장률 하락과 같은 경제 지표 악화로 이어질 수 있음

01 데이터 - 주요 데이터 파일 및 변수

데이터 특징

본 데이터는 총 400개가 넘는 데이터로 이루어져 있으며, 과거 신용 기록, 인적사항,
세금 납부, 학력 등등 다양한 데이터가 있으며,
특징으로는 해당 사람에 대한 역사적인 기록으로 값이 이루어진 변수들이 많이 있음.

01 데이터 - 주요 데이터 파일 및 변수

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	case_id	target	credit_loan	secured_lo	overdue_p	avg_repayr	bureau_tot	debtoutsta	debtoverdu	max_numberofoverdueinstlmax_1039L			

credit_loans_cnt: 신용대출의 수
secured_loand_cnt: 담보 대출의 수
overdue_proportion: 총 상환기간(월) 대비 연체하여 지불한 월의 비율
avg_repayment_month: 대출별 상환기간의 평균
bureau_totalamount: 대출신청한 금액 총합
debtoutstand_525A: 앞으로 갚아야 하는 남은 금액의 총합
debtoverdue_47A: 연체된 금액의 총합
max_numberofoverdueinstlmax_1039L:
numberofoverdueinstlmax_1039L(활성 계약의 미납 할부 수)의 최대값

02 EDA -전처리1

```
#결측값 비율이 20% 이상되는 변수 제거
missing_ratio = df.isnull().mean()

# 결측값 비율이 20% 이상인 변수 제거
df = df.loc[:, missing_ratio < 0.2]
```

=> missing_ratio < 0.2는 결측값 비율이 20% 미만인 열의 불리언 인덱스를 생성하고, df.loc[:, ...]를 사용하여 이러한 열만 선택

```
df.shape
```

```
(1526659, 102)
```

=> 1,526,659개의 행과 102개의 열을 가지고 있음을 나타냄

```
df.set_index('case_id', inplace=True)
```

=> df.set_index('case_id', inplace=True)는 'case_id' 열을 데이터프레임의 인덱스로 설정
inplace=True는 원본 데이터프레임을 수정함을 의미

```
df = df.dropna() #
```

=> df = df.dropna()는 데이터프레임에서 결측값이 있는 모든 행을 제거

```
df.shape
```

```
(720358, 101)
```

=> 720,358개의 행과 101개의 열을 가지고 있음을 나타냄

```
#M으로 끝나는 변수는 전부 범주형 변수로 간주
cat_cols = [col for col in df.columns if col.endswith('M')]
df[cat_cols] = df[cat_cols].astype('category')
df[cat_cols].info()
```

=> 열 이름이 'M'으로 끝나는 열의 목록을 만듦
=> 이러한 열을 범주형 데이터 타입으로 변환
=> 변환된 열의 정보를 출력

02 EDA -전처리1

```
df_0 = df[df['target']==0] #타겟값이 0인 데이터
df_1 = df[df['target']==1] #타겟값이 1인 데이터
```

```
def summary(df):
    summ = pd.DataFrame(df.dtypes, columns=['data type'])
    summ['결측 비율'] = df.isnull().sum().values/len(df)
    summ['중복값개수'] = df.duplicated().sum()
    summ['범주개수'] = df.nunique().values
    desc = pd.DataFrame(df.describe(include='all').transpose())
    summ['최소'] = desc['min'].values
    summ['최대'] = desc['max'].values
    summ['평균'] = desc['mean'].values
    summ['표준편차'] = desc['std'].values
    summ['최빈값'] = desc['top'].values
    summ['최빈값 비율'] = desc['freq'].values/len(df)

    return summ
```

```
summary(df_0).to_csv('fianl_0_summary.csv', encoding='utf-8-sig')
summary(df_1).to_csv('final_1_summary.csv', encoding='utf-8-sig')
```

빨간색이 target=1인 친구들 contaddr_matchlist_1032L paytype1st_925L paytype_783L 제거 -> 값의 비율이 똑같아서 의미가 없음

```
df = df.drop(['contaddr_matchlist_1032L','paytype1st_925L','paytype_783L']
```

비슷한 변수들 합치기

```
feature = pd.read_csv('feature_definitions.csv' , index_col=0)
columns_to_select = [col for col in df.columns if col in feature.index]
feature_static = feature.loc[columns_to_select]

#variable : Description 형식으로 하나의 컬럼으로 만들어줘
feature_static['variable'] = feature_static.index + ' : ' + feature_static[
feature_static['variable'] = feature_static['variable'].str.replace('₩n', '
feature_static = feature_static['variable']

feature_static.to_csv('feature_final.csv', index=False)
```

1.

clientscnt12m_3712952L clientscnt3m_3712950L clientscnt6m_3712949L 평균내기 ->clientscnt

2. lastapplicationdate_877D 삭제

3. numactivecreds_622L numactivecredschannel_414L numactiverelcontr_750L 평균내기 -> numactivecreds

4.posfpd10lastmonth_333P ,posfpd30lastmonth_3976960P,posfstqpd30lastmonth_3976962P 의 평균 ->posfpd

5.days120_123L,days180_256L,days30_165L,days360_512L,days90_310L의 평균 -> days

6.firstquarter_103L,fourthquarter_440L,secondquarter_766L,thirdquarter_1082L 의 평균 ->quarter

02 EDA -전처리1

```
df_0 = df[df['target']==0]
df_1 = df[df['target']==1]
```

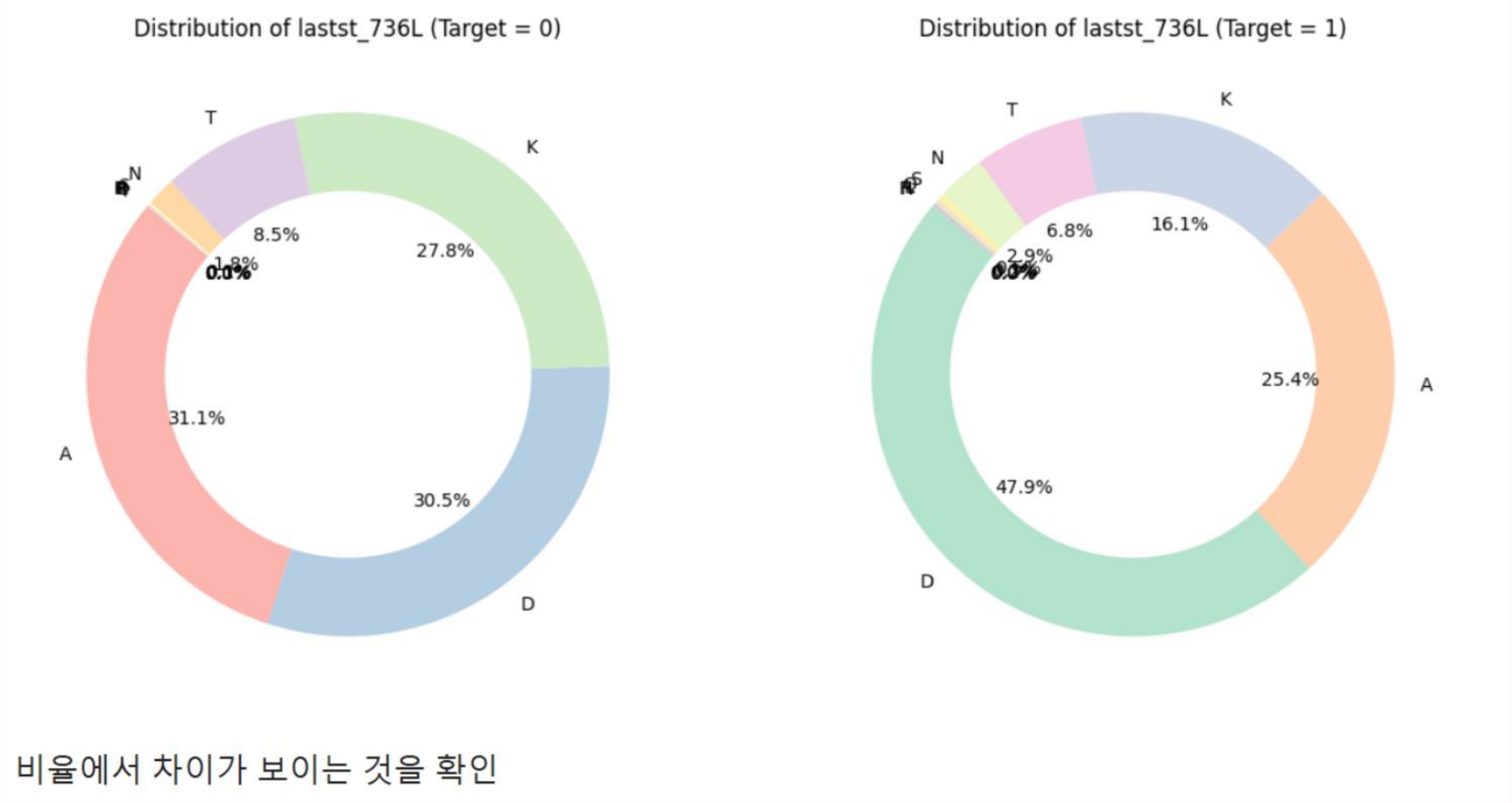
=> df 데이터프레임을 'target' 값이 0인 경우와 1인 경우로 분리함
df_0는 'target' 값이 0인 데이터를, df_1는 'target' 값이 1인 데이터를 포함

```
# 'credtype_322L' 변수의 값별 빈도 계산
value_counts_0 = df_0['lastst_736L'].value_counts()
value_counts_1 = df_1['lastst_736L'].value_counts()
```

=> 이 줄은 df_0과 df_1에서 'lastst_736L' 변수의 값별 빈도를 계산합니다.
value_counts_0와 value_counts_1는 각각 타겟 값이 0과 1인 경우의 'lastst_736L' 값의 빈도 수를 나타냄

```
# 도넛 플롯 그리기
fig, axes = plt.subplots(1, 2, figsize=(14, 7))

colors_0 = plt.cm.Pastel1(range(len(value_counts_0)))  # 파스텔톤 색상 설정
colors_1 = plt.cm.Pastel2(range(len(value_counts_1)))  # 파스텔톤 색상 설정

# Target = 0 도넛 플롯
axes[0].pie(value_counts_0, labels=value_counts_0.index, colors=colors_0, autopct='%1.1f%%', startangle=140, wedgeprops=dict(width=0.3))
axes[0].set_title('Distribution of lastst_736L (Target = 0)')

# Target = 1 도넛 플롯
axes[1].pie(value_counts_1, labels=value_counts_1.index, colors=colors_1, autopct='%1.1f%%', startangle=140, wedgeprops=dict(width=0.3))
axes[1].set_title('Distribution of lastst_736L (Target = 1)')

plt.show()
```

02 EDA -전처리1

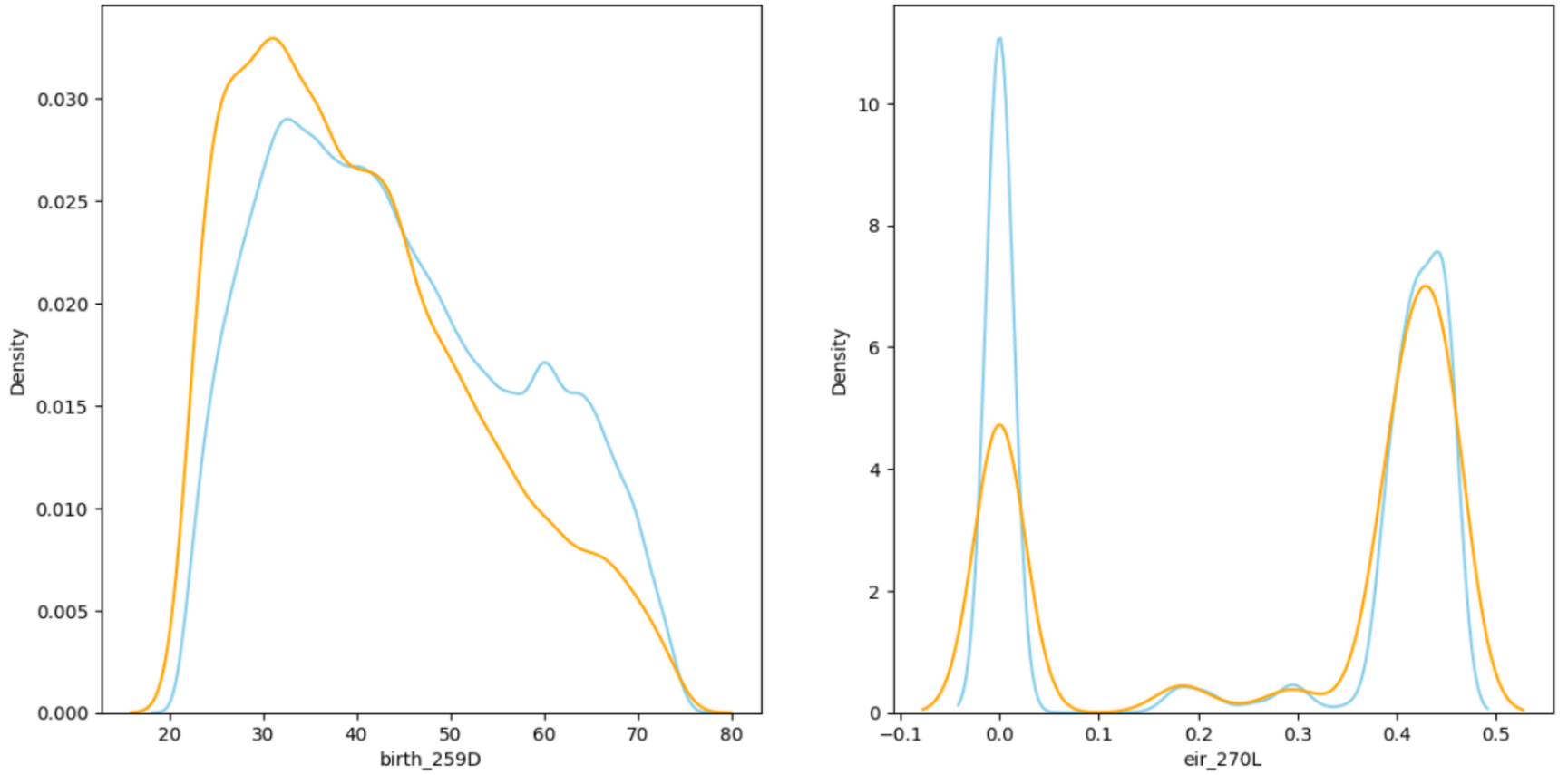


비율에서 차이가 보이는 것을 확인

02 EDA -전처리1

```
#sns.kdeplot for df_0 , df_1
fig, ax = plt.subplots(1, 2, figsize=(14, 7))

sns.kdeplot(df_0['birth_259D'], ax=ax[0], color='skyblue', label='Target = 0')
sns.kdeplot(df_1['birth_259D'], ax=ax[0], color='orange', label='Target = 1')

sns.kdeplot(df_0['eir_270L'], ax=ax[1], color='skyblue', label='Target = 0')
sns.kdeplot(df_1['eir_270L'], ax=ax[1], color='orange', label='Target = 1')
```

9]: <Axes: xlabel='eir_270L', ylabel='Density'>

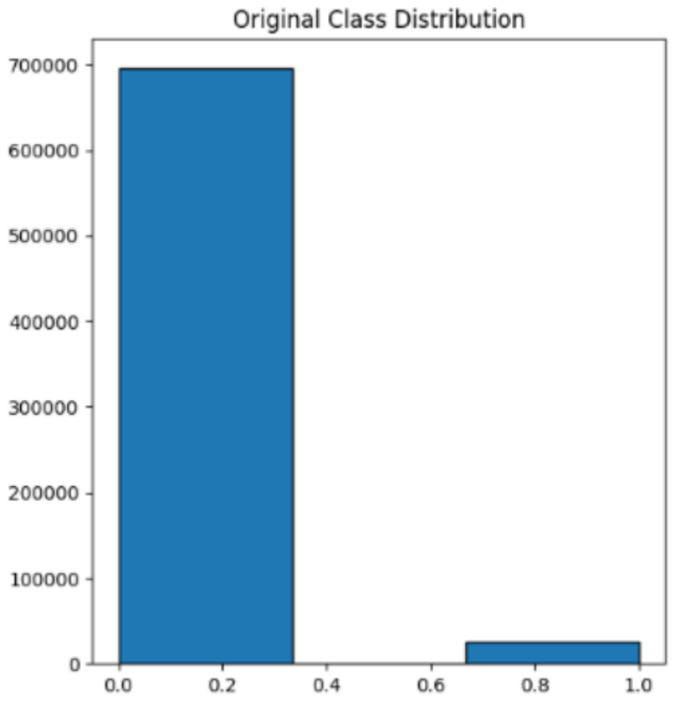


03 XAI - 언더 샘플링

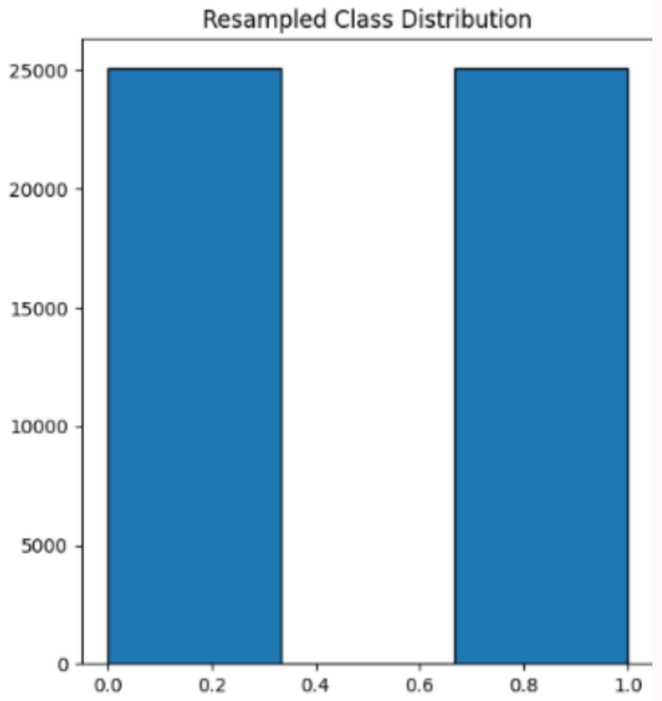
```
df['target'].value_counts() #타겟값 분포 확인
```

```
target
0    695292
1     25066
Name: count, dtype: int64
```

```
[ ]  X = df.drop('target', axis=1)
     y = df['target']

     # 언더샘플링 적용
     rus = RandomUnderSampler(random_state=42)
     X_res, y_res = rus.fit_resample(X, y)

     # 언더샘플링된 데이터프레임 생성
     df_res = pd.DataFrame(X_res, columns=X.columns)
     df_res['target'] = y_res

     # 시각화
     fig, ax = plt.subplots(1, 2, figsize=(12, 6))

     ax[0].hist(y, bins=3, edgecolor='k')
     ax[0].set_title('Original Class Distribution')
     ax[1].hist(y_res, bins=3, edgecolor='k')
     ax[1].set_title('Resampled Class Distribution')

     plt.show()
```



=> 원본 데이터의 클래스 분포



=> 언더샘플링된 데이터의 클래스 분포

```
[ ]  df_res['target'].value_counts()
```

```
target
0    25066
1    25066
Name: count, dtype: int64
```

03 XAI - logistic regression (1)

```python
# 로지스틱
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# 예시 데이터프레임 생성 (실제 데이터프레임인 merged_df를 사용)
# merged_df = pd.read_csv('your_file.csv')  # 실제 데이터 로드

# 입력 변수와 목표 변수 분할
X = df_res.drop(columns=['target'])
y = df_res['target']

# 훈련 세트와 테스트 세트 분할
X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size=0.3, random_state=42)


# 로지스틱 회귀 모델 생성 및 훈련
logistic_model = LogisticRegression()
logistic_model.fit(X_train, y_train)

# 테스트 세트로 모델 평가
y_pred = logistic_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

# 정확도 출력
print(f"로지스틱 회귀 모델의 정확도: {accuracy}")
```

test

```
로지스틱 회귀 모델의 정확도: 0.5823803191489362
```

train

```
로지스틱 회귀 모델의 정확도: 0.5883962156616893
```

```python
from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(y_test, y_pred)

print(conf_matrix)
```

```
[[4012 3518]
 [2763 4747]]
```

target1과 target0을 균형있게 틀림. 성능이 높진 않음.

03 XAI - SVM

SVM

커널함수를 방사형기저함수(rbf)로 지정함

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix
import pandas as pd
from sklearn.model_selection import train_test_split

# SVM 모델 생성 및 훈련
svm_model = SVC(kernel='rbf')
svm_model.fit(X_train, y_train)

# 테스트 세트로 모델 평가
y_pred = svm_model.predict(X_test)

# 정확도 출력
accuracy = accuracy_score(y_test, y_pred)
print(f"SVM 모델의 정확도(test): {accuracy}")

# 학습 세트로 모델 평가
y_pred2 = svm_model.predict(X_train)
# 정확도 출력
accuracy2 = accuracy_score(y_train, y_pred2)
print(f"SVM 모델의 정확도(train): {accuracy2}")


# Confusion Matrix (Contingency Table) 생성
conf_matrix = confusion_matrix(y_test, y_pred)


print(conf_matrix)
```

```
SVM 모델의 정확도(test): 0.5753324468085106
SVM 모델의 정확도(train): 0.5861734868346061
[[3020 4510]
 [1877 5633]]
```

=> 앞서서 했던 logistic regression과 거의 동일한 해석.
1 일때에 비해서 1 일때는 예측성능이 좀 더 낮음. target이 0일때 800개 정도 더 틀림.
train/test에서 성능도 비슷함. 다만 성능이 낮음. 따라서 모델이 데이터를 잘 학습할 수 있을만큼 충분히 복잡하지 않은 것이기 때문에 일단은 더 복잡한 모델을 써봐야겠다

03 XAI - Random Forest (1)

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import pandas as pd
from sklearn.model_selection import train_test_split

# Random Forest 모델 생성 및 훈련
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# 테스트 세트로 모델 평가
y_pred_rf = rf_model.predict(X_test)

# 정확도 출력
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print(f"Random Forest 모델의 정확도: {accuracy_rf}")

# Confusion Matrix (Contingency Table) 생성
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)

# 혼동행렬
print(conf_matrix_rf)
```

```
Random Forest 모델의 정확도: 0.7339760638297872
[[5314 2216]
 [1785 5725]]
```

```python
[ ] # 학습 세트로 모델 평가
    y_pred_rf = rf_model.predict(X_train)

    # 정확도 출력
    accuracy_rf = accuracy_score(y_train, y_pred_rf)
    print(f"Random Forest 모델의 정확도: {accuracy_rf}")
```

```
Random Forest 모델의 정확도: 1.0
```

더 복잡한 모델을 쓰니 target이 0일때와 1일때 예측성능은 거의 비슷해졌다. 또한 정확도도 높아짐 하지만 학습데이터에서 정확도는 1인 반면 시험데이터에선 0.75이므로 과적합이 발생했다.

따라서 과적합을 막기 위해 max_depth를 설정하여 최대트리깊이가 7층까지만 갈 수 있도록 제한하자.

03 XAI - Random Forest (2)

과적합을 막기 위해 max_depth를 설정하여 최대트리깊이가 7층까지만 갈 수 있도록 제한하자.

```
[ ]  max_depth = 7  # 최대 트리 깊이 설정
     rf_model = RandomForestClassifier(n_estimators=100, max_depth=max_depth, random_state=42)
     rf_model.fit(X_train, y_train)

     # 테스트 세트로 모델 평가
     y_pred = rf_model.predict(X_test)
     y_pred2 = rf_model.predict(X_train)

     # 정확도 출력
     accuracy = accuracy_score(y_test, y_pred)
     accuracy2 = accuracy_score(y_train, y_pred2)
     print(f"랜덤 포레스트 모델의 정확도 (최대 트리 깊이 {max_depth}): {accuracy}")
     print(f"랜덤 포레스트 모델의 정확도 (최대 트리 깊이 {max_depth}): {accuracy2}")
```

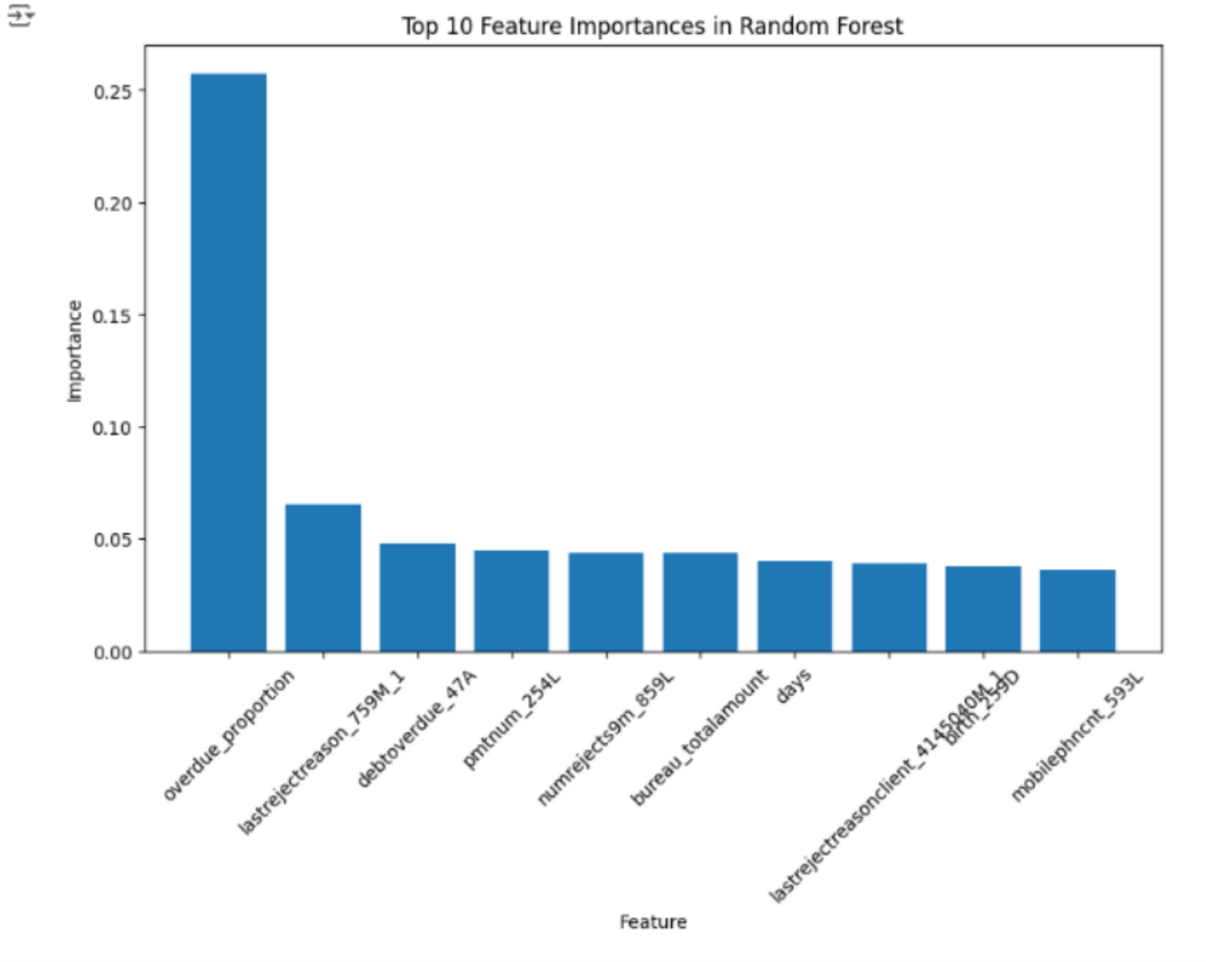
```
랜덤 포레스트 모델의 정확도 (최대 트리 깊이 7): 0.7217420212765957
랜덤 포레스트 모델의 정확도 (최대 트리 깊이 7): 0.7367776131311979
```

```
[ ]  # Confusion Matrix (Contingency Table) 생성
     conf_matrix_rf = confusion_matrix(y_test, y_pred)

     # 혼동행렬
     print(conf_matrix_rf)
```

```
[[5182 2348]
 [1837 5673]]
```

학습/시험데이터에서의 정확도가 비슷해짐. 과적합 문제 해결.

03 XAI - Random Forest (4)



1. 'overdue_proportion'은 고객의 연체 비율을 나타내는 변수
이는 고객의 연체 기록이 모델의 예측에 가장 큰 영향을 미침을 의미.

2. 'lastrejectionreason_759M_1'은 최근 대출 거절 사유를 나타내는 변수
이는 최근에 대출이 거절된 이유가 모델의 예측에 중요한 역할을 함을 의미.

3. 'debtoverdue_47A'는 특정 유형의 연체 부채를 나타내는 변수
이는 특정 유형의 연체 부채가 있는 경우, 고객의 신용 위험이 높아짐을 의미

4. 'pmturn_254L'은 아마도 특정 기간 동안의 결제 회전율을 나타내는 변수
결제 회전율이 높을수록, 고객이 신용을 잘 관리하고 있을 가능성이 높음을 의미

5. 'numrejectsm_859L'은 아마도 최근 몇 개월 동안의 대출 거절 횟수를 나타내는 변수
대출 거절 횟수가 많을수록, 고객의 신용 위험이 높다고 예측할 수 있음

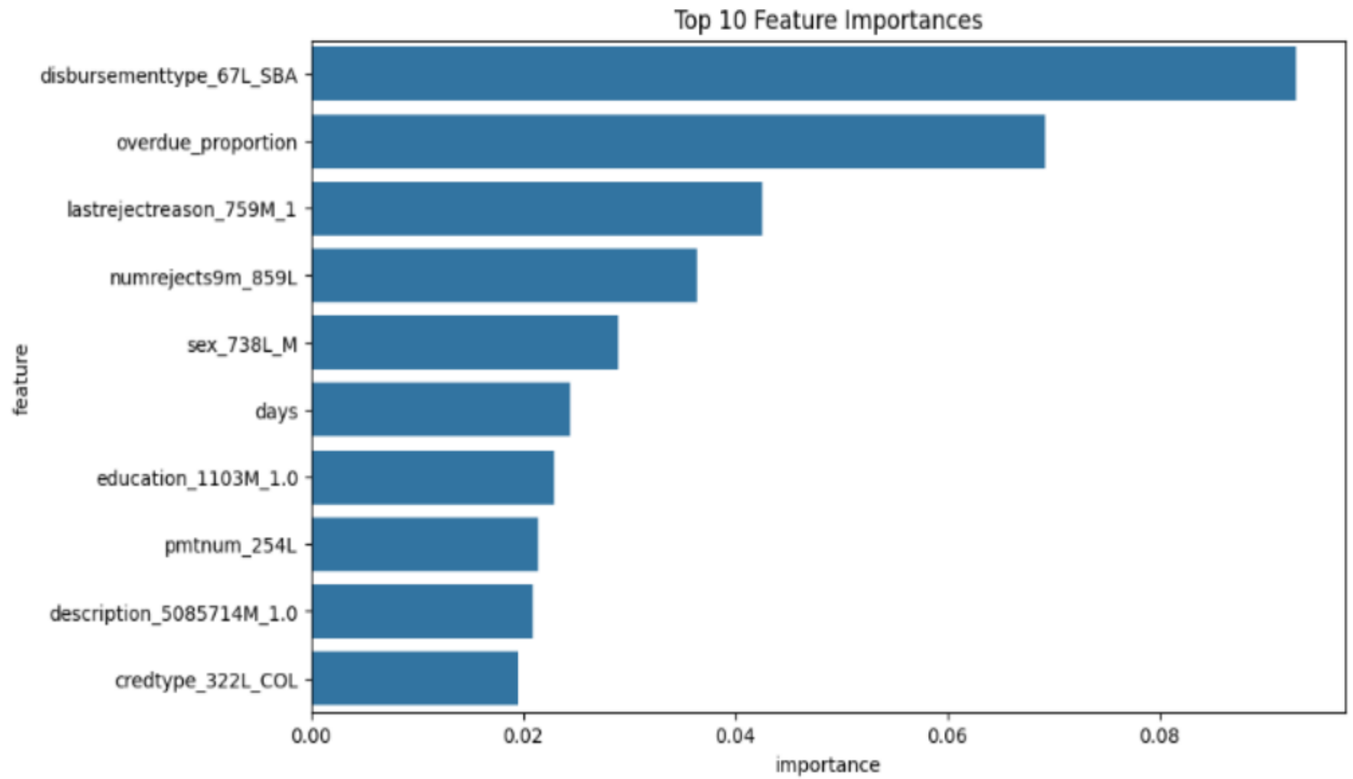
6. 'bureau_totalamount'는 신용 기관에 보고된 총 부채 금액을 나타내는 변수
총 부채 금액이 많을수록, 고객의 신용 위험이 높음을 의미

=> 이 변수들은 모두 고객의 신용 위험을 평가하는 데 중요한 역할을 합니다. 높은 연체 비율, 최근 대출 거절 사유, 특정 유형의 연체 부채, 결제 회전율, 대출 거절 횟수, 총 부채 금액 등은 모두 모델이 고객의 신용 위험을 예측하는 데 중요한 정보. 이러한 변수들은 고객의 과거 신용을 반영하며, 미래의 신용 위험을 평가하는 데 중요한 지표로 사용됨

03 XAI - XG 부스트 (1)

```
[ ]
    # XGBoost 모델 학습
    clf = XGBClassifier(random_state=42)
    clf.fit(X_train, y_train)

    # 예측
    y_pred = clf.predict(X_test)

    # 평가
    print(confusion_matrix(y_test, y_pred))
    print(classification_report(y_test, y_pred))

    # 피처 중요도 계산
    feature_importances = clf.feature_importances_
    features = X.columns
    feature_importance_df = pd.DataFrame({'feature': features, 'importance': feature_importances})

    # 상위 10개의 피처 중요도 추출
    top_10_features = feature_importance_df.sort_values(by='importance', ascending=False).head(10)

    # 시각화
    plt.figure(figsize=(10, 6))
    sns.barplot(x='importance', y='feature', data=top_10_features)
    plt.title('Top 10 Feature Importances')
    plt.show()
```

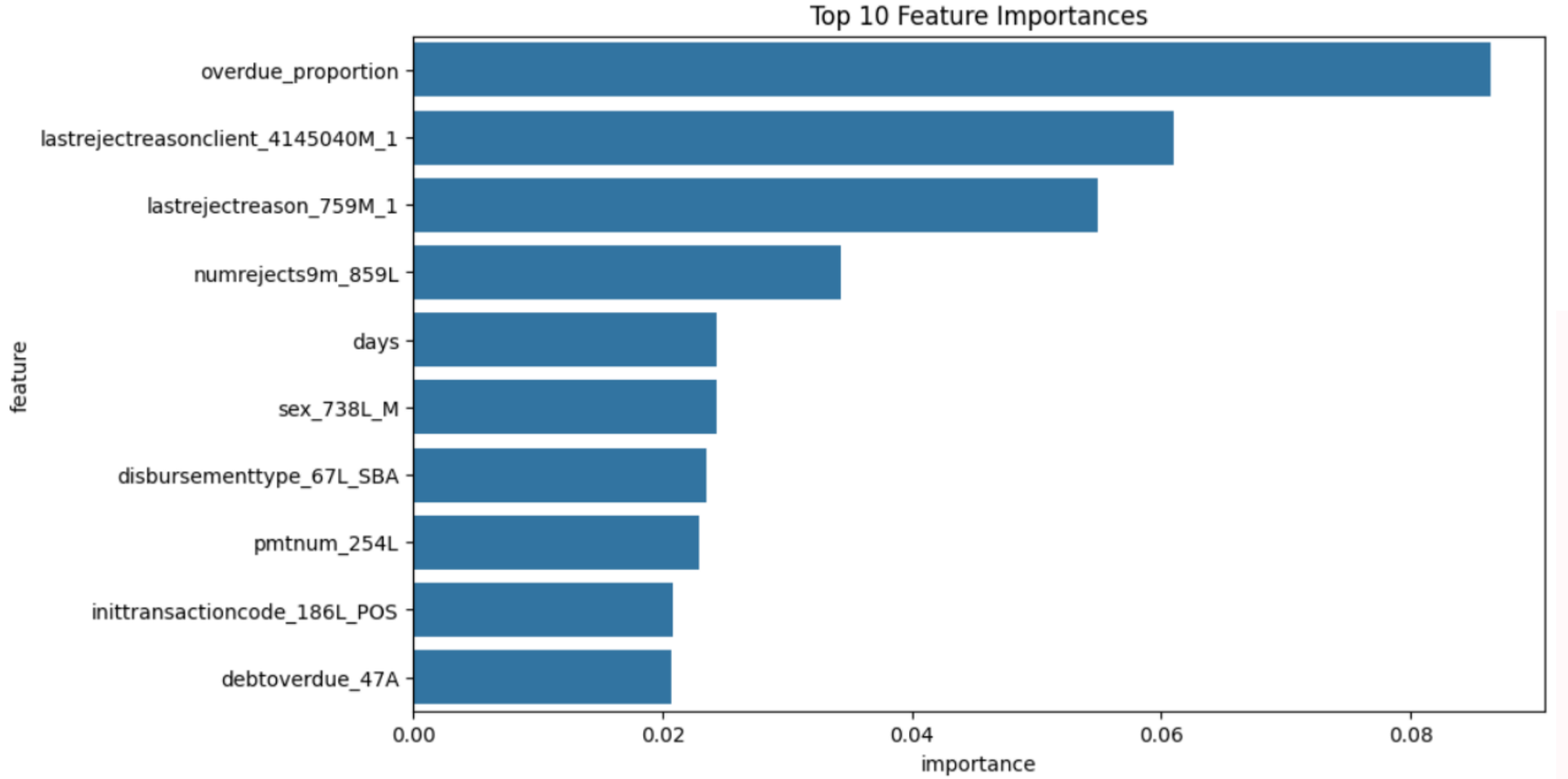
```
[[5374 2156]
 [1807 5703]]
              precision    recall  f1-score   support

           0       0.75      0.71      0.73      7530
           1       0.73      0.76      0.74      7510

    accuracy                           0.74     15040
   macro avg       0.74      0.74      0.74     15040
weighted avg       0.74      0.74      0.74     15040
```



03 XAI - XG 부스트 (2)

XG부스트에 대해서 하이퍼 파라미터 최적화를 통해 성능을 좀 더 높여보자 최적화 방식은 random search를 사용했으며 이는 속도 빠르게 선택함

```
Best parameters found:  {'colsample_bytree': 0.6375493160325166, 'learning_rate': 0.0648597991321922, 'max_depth': 7, 'min_child_weight': 5, 'n_estimators': 178, 'subsample': 0.9819461122652776}
Best F1 score:  0.7492758089670799
[[5369 2161]
 [1643 5867]]
              precision    recall  f1-score   support

           0       0.77      0.71      0.74      7530
           1       0.73      0.78      0.76      7510

    accuracy                           0.75     15040
   macro avg       0.75      0.75      0.75     15040
weighted avg       0.75      0.75      0.75     15040
```



03 XAI - LIME 설명

LIME (Local Interpretable Model-agnostic Explanations)

뭐야?

LIME은 우리가 AI 모델이 왜 특정 예측을 했는지 로컬하게(즉, 개별적인 예측에 대해) 설명해주는 방법이야. 모델의 내부 구조나 매개변수에 상관없이 사용할 수 있어.

어떻게 작동해?

1. **특정 데이터 포인트 선택**: 먼저 설명하고 싶은 예측을 선택해.
2. **데이터 변형**: 그 예측 주변의 비슷한 데이터를 만들어내.
3. **모델 예측**: 그 데이터를 모델에 넣어 예측값을 받아.
4. **로컬 선형 모델 학습**: 변형된 데이터와 예측값으로 로컬 선형 모델을 학습해.
5. **설명 제공**: 이 선형 모델의 가중치를 통해 원래 예측을 설명해줘.

예시

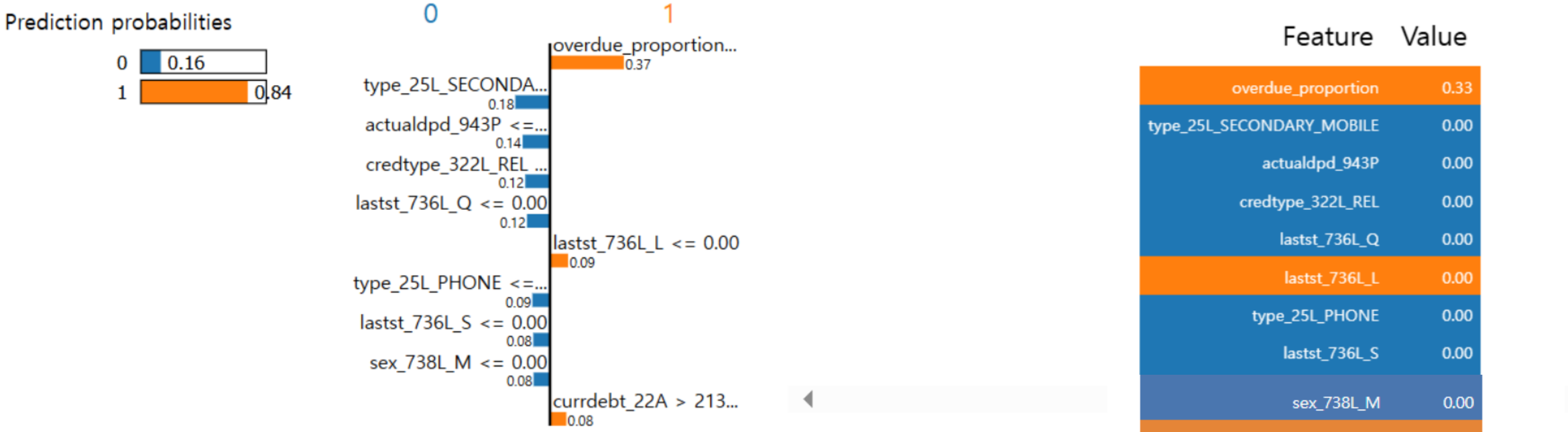
예를 들어, 이미지 분류 모델이 어떤 이미지를 "고양이"로 예측했어. LIME을 사용하면:

1. 그 고양이 이미지를 선택해.
2. 이미지의 일부 픽셀을 바꾸거나 마스크해서 비슷한 이미지를 여러 개 만들어.
3. 각 변형된 이미지를 모델에 넣어 예측값을 얻어.
4. 변형된 이미지와 예측값으로 로컬 선형 모델을 학습해.
5. 선형 모델의 가중치를 통해 원래 이미지의 어떤 부분이 "고양이"라고 예측되게 했는지 설명할 수 있어.

03 XAI - LIME 결과

In [24]:
```
import lime
import lime.lime_tabular
```

In [25]:
```
import numpy as np

# LIME 설명 객체 생성
explainer = lime.lime_tabular.LimeTabularExplainer(X_train.values, feature_names=X_train.columns, class_names=['0', '1'], discretize_continuous=True)

# 단일 예측 설명
i = 0  # X_test에서 예측을 설명할 인덱스
exp = explainer.explain_instance(X_test.values[i], clf.predict_proba, num_features=15)
exp.show_in_notebook(show_table=True, background_color='#f5f5f5')
```



03 XAI - SHAP 설명

SHAP (Shapley Additive Explanations)

뭐야?

SHAP는 게임 이론에서 나온 샤플리 값을 사용해서 AI 모델의 예측을 설명해주는 방법이야. 각 피처(feature)가 예측값에 어떻게 기여하는지 계산해줘.

어떻게 작동해?

1. **샤플리 값 계산**: 각 피처가 예측값에 얼마나 기여하는지 모든 가능한 피처 조합에 대해 계산해.
2. **피처 중요도 도출**: 각 피처의 평균적인 기여도를 계산해서 중요도를 평가해.
3. **설명 제공**: 각 피처의 기여도를 통해 모델의 예측을 설명해줘.

예시

예를 들어, 신용 대출 승인 모델이 어떤 사람의 대출을 승인했어. SHAP을 사용하면:

1. 그 사람의 정보를 모델에 넣어서 예측값을 얻어.
2. 각 피처(예: 나이, 소득, 신용 점수 등)가 예측값에 얼마나 기여하는지 계산해.
3. 샤플리 값으로 각 피처의 기여도를 계산해.
4. "신용 점수"가 높아서 승인된 경우, 이 피처의 기여도가 높게 나타나겠지.

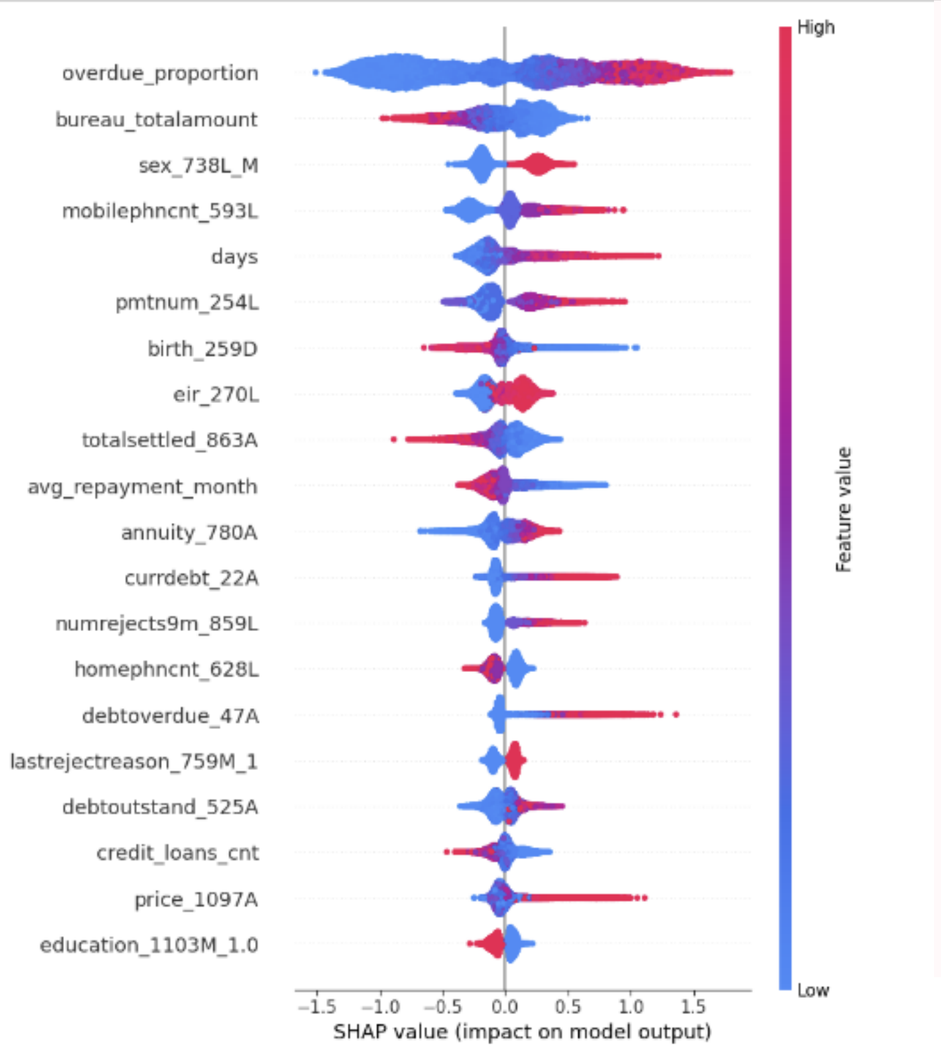
비교

- **LIME**은 특정 예측에 대해 빠르게 이해하고 싶을 때 좋아.
- **SHAP**은 모델 전체 예측 구조를 이해하고 피처 중요도를 공정하게 평가하는 데 좋아.

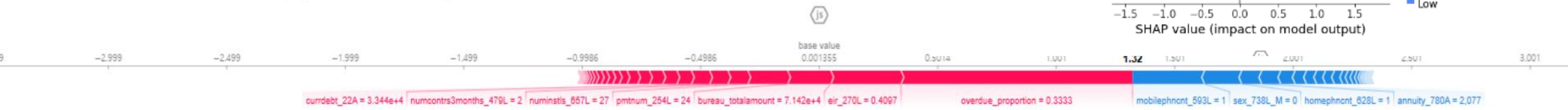
두 방법 모두 모델을 더 잘 이해하고 신뢰할 수 있게 도와주는 중요한 도구야. 필요에 따라 적절히 사용하면 돼!

03 XAI - SHAP 결과

```
[ ]  import shap

     # SHAP 값 계산
     explainer = shap.TreeExplainer(best_clf)
     shap_values = explainer.shap_values(X_test)

     # 시각화
     shap.summary_plot(shap_values, X_test)
     shap.initjs()
     shap.force_plot(explainer.expected_value, shap_values[0], X_test.iloc[0])
```



점은 각각 개인 관측치이며 점이 한 변수에 대해서 오른쪽으로 가있다 = 해당 변수가 채무 불이행하도록 영향을 줬다. 오른쪽으로 간 정도에 따라 해당 변수가 채무 불이행에 큰 영향을 준 것



03 XAI - 반사실 설명 (1)

Counterfactual Explaination (반사실 설명)은 AI 모델의 예측을 이해하는 데 중요한 방법 중 하나야. 쉽게 말해, "만약에 이러지 않았다면, 모델의 예측은 어떻게 달라졌을까?"를 묻는 거야. 이를 통해 특정 예측에 영향을 미친 요인들을 더 잘 이해할 수 있어.

Counterfactual Explanation (반사실 설명) 뭐야? Counterfactual 설명은 모델이 특정 예측을 할 때, 그 예측이 바뀌려면 어떤 조건이 달라져야 하는지를 알려줘. 예를 들어, 어떤 데이터 포인트가 A라는 결과를 예측했을 때, 만약 B라는 결과가 나왔으려면 어떤 피처(특징)가 어떻게 변했어야 하는지를 설명하는 거야.

어떻게 작동해? 기본 데이터 포인트 선택: 먼저, 현재 모델의 예측을 보고 싶은 특정 데이터 포인트를 선택해. 목표 예측 설정: 이 데이터 포인트가 다른 예측 결과를 가지게 하려면 어떤 목표 예측값을 설정해야 할지 결정해. 피처 변형: 현재 데이터 포인트의 피처 값을 변경해서 목표 예측값을 얻을 수 있는 새로운 데이터 포인트를 찾아. 설명 제공: 원래 데이터 포인트와 새로운 데이터 포인트의 차이를 설명해. 이를 통해 예측이 바뀌기 위해 어떤 변화가 필요했는지 알 수 있어. 예시 예를 들어, 한 신용 대출 신청자가 대출 승인을 받지 못했어. 반사실 설명을 사용하면:

그 신청자의 현재 데이터를 선택해. 대출 승인을 목표 예측값으로 설정해. 모델이 대출을 승인하도록 하기 위해 소득이나 신용 점수와 같은 피처들을 변경해. 예를 들어, "신용 점수가 650에서 700으로 올라갔다면 대출이 승인되었을 것이다"라는 식으로 설명할 수 있어. 왜 중요해?
Counterfactual 설명은 사용자가 모델의 결정에 대해 더 명확히 이해하고, 어떤 조건이 결과에 중요한 영향을 미치는지 알 수 있게 해줘. 이를 통해 모델의 공정성을 평가하거나, 모델 개선의 방향을 찾는 데 유용해.

요약 Counterfactual Explanation은 모델의 예측을 이해하는 데 중요한 도구야. "만약에 이랬다면 결과가 어떻게 달라졌을까?"를 묻고, 그에 대한 답을 찾아줘. 이를 통해 모델의 예측에 어떤 피처가 중요한지, 그리고 어떤 변화를 통해 다른 결과를 얻을 수 있는지 알 수 있어.

03 XAI - 반사실 설명 (2)

```
100%|██████████| 1/1 [00:04<00:00,  4.24s/it]Query instance (original outcome : 1)
```

	actualdpd_943P	annuity_780A	annuitynextmonth_57A	applications30d_658L	avg_repayment_month	birth_259D	bureau_totalamount	clientscnt	clientscnt_157L	contaddr_smempladdr_334L	...	status_219L_P
0	0.0	2077.199951	3353.600098	0.0	24.0	35.945244	71419.398438	0.0	0.0	0	...	0

1 rows × 95 columns

atus_219L_R	status_219L_S	status_219L_T	type_25L_PHONE	type_25L_PRIMARY_EMAIL	type_25L_PRIMARY_MOBILE	type_25L_SECONDARY_MOBILE	target
0	0	0	0	0	1	0	1

Diverse Counterfactual set (new outcome: 0)

	actualdpd_943P	annuity_780A	annuitynextmonth_57A	applications30d_658L	avg_repayment_month	birth_259D	bureau_totalamount	clientscnt	clientscnt_157L	contaddr_smempladdr_334L	...	status_219L_P	target
0	-	-	-	-	22.8	-	-	-	-	-	...	-	0.0
1	-	-	-	4.0	-	-	-	-	-	-	...	-	0.0
2	-	-	-	-	-	-	-	-	-	-	...	-	0.0
3	-	1471.8	-	-	-	-	-	-	-	-	...	-	0.0
4	-	-	-	-	-	-	-	-	-	-	...	-	0.0

5 rows × 95 columns

마무리

1.overdue_proportion: 부채 기간 동안 연체된 기간의 비율
2.lastrejectreason_759M : 가장 최근에 거부된 신청서의 거절 이유.
3.debtoverdue_47A: 현재 고객의 기존 신용 계약에 연체된 금액.
4.pmtnum_254L: 고객이 대출 상환을 위해 지금까지 지불한 총 횟수
5.numrejects9m_859L: 지난 9개월 동안 거부된 신용 신청 수.
6.sex_7389L_M: 고객의 성별
7.education_1103M: 고객의 교육 수준
8.Days: 신용 기관 조회 횟수의 평균

이상으로 발표를 마칩니다.
들어주셔서 감사합니다 !