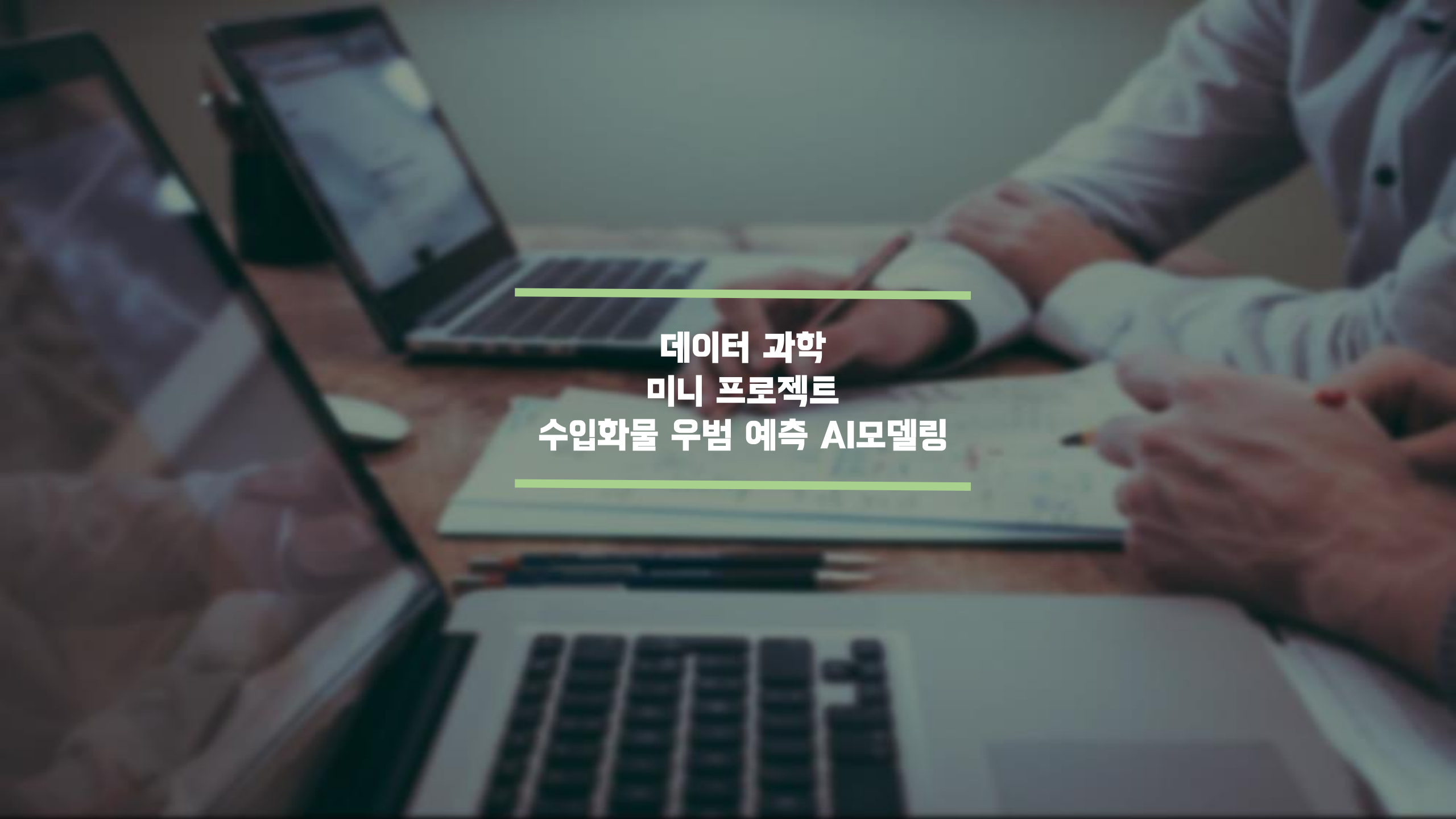


데이터 과학
미니 프로젝트
수입화물 우범 예측 AI모델링

2015021106 장석현

2016020148 이재빈

2017015197 정다윤

데이터 분석

데이터 셋
데이터 분석 결과

전처리

전처리 방법
Over Smapling

모델링

모델링 방법
결론

데이터 셋 구조

수치형 변수
Length : 5

신고중량(KG)
과세가격원화금액
관세율
우범여부
핵심적발

범주형 변수
Length : 19

신고번호
신고일자
통관지세관부호
신고인부호
수입자부호
검사결과코드

해외거래처부호
특송업체부호
수입통관계획
수입신고구분코드
수입거래구분코드
수입종류코드

징수형태코드
운송수단유형
반입보세구역부호
H10단위부호
적출국가코드
원산지국가코드
관세율구분코드

특송업체부호

```
[5]  df_org = pd.read_csv('/content/drive/MyDrive/train.csv', encoding=

[50] data_array = df_org['우범여부'].groupby(df_org['특송업체부호'])

     #data_array.mean()
     arr_s = pd.Series(data_array.mean())
     arr_s.sort_values(ascending=False)

     특송업체부호
     5FDIRV    0.714286
     A41WVG    0.709677
     ZVCGS6    0.688312
     47QGIA    0.666667
     5HIQAV    0.666667
               ...
     24BG4R    0.363937
     004TIW    0.346300
     TQ18AK    0.343943
     PR5UFJ    0.275348
     PAVJZL    0.270908
     Name: 우범여부, Length: 80, dtype: float64
```

√ 발견 사실

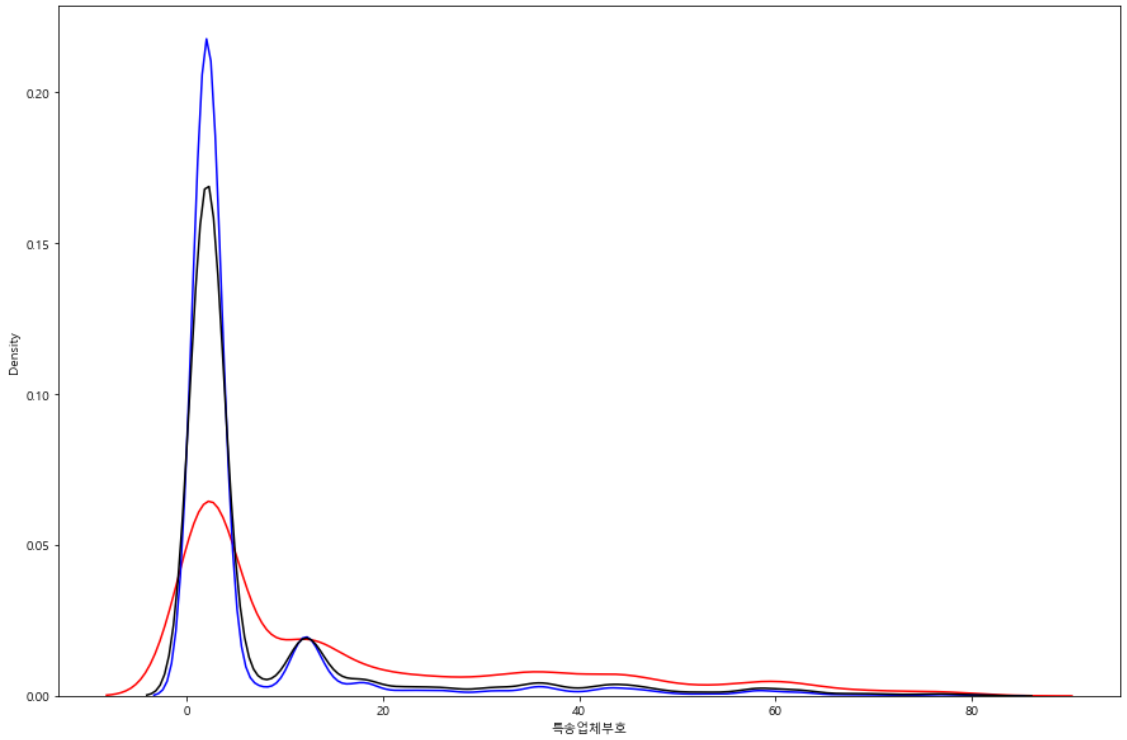
특송업체부호의
레이블 중
Count가 낮은 부호의
우범 비율이 다소 높게 측정이 됨.

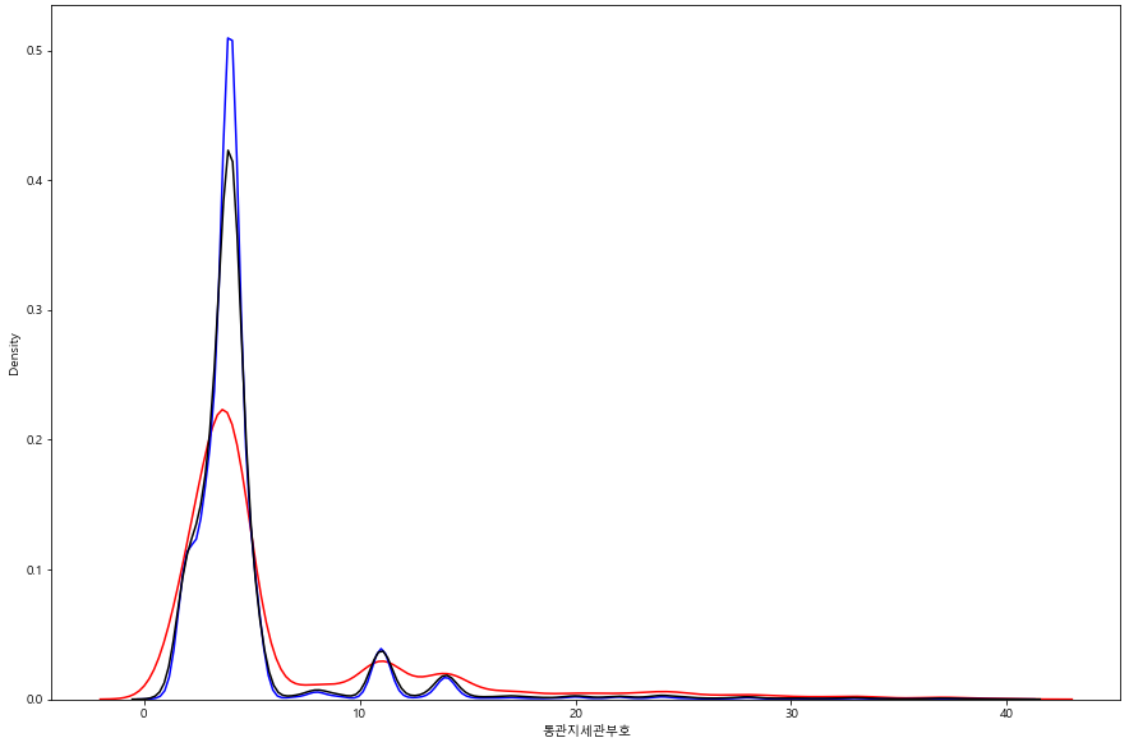
√ 새로운 가설

“특정 특송업체 중
소수의 값을 차지하는 경우
우범 가능성을 높임”

-> 특송업체 별 비율을
새로운변수로 추가

특송업체부호 & 통관지세관부호
Density 빨간색(우범), 파란색(우범 아님), 검은색(평균)



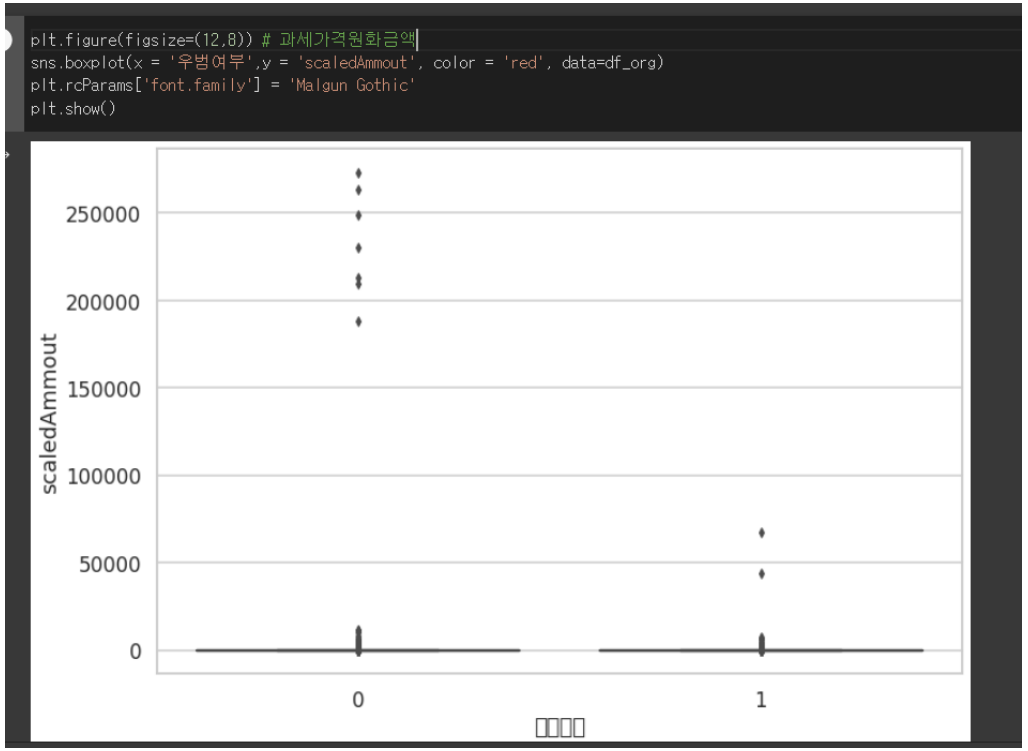


특송업체부호, 통관지세관 부호의 특정 레이블에 우범, 우범 아님 값의 분포가 몰려 있음.
다른 범주형 변수인 해외거래처부호, 운송수단 유형 코드도 마찬가지임.
-> 앞서 진행한 가설을 적용하여 모델의 정확도를 높이고자 함.

신고중량(KG), 과세가격원화금액 boxplot 결과

과세가격원화금액

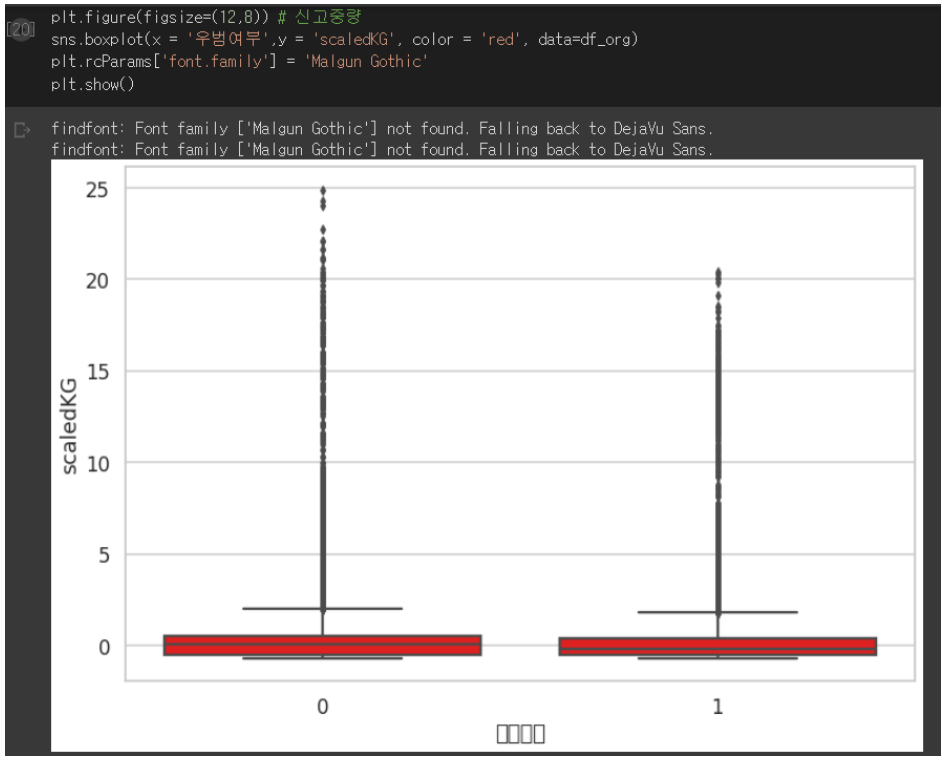
```
plt.figure(figsize=(12,8)) # 과세가격원화금액
sns.boxplot(x = '우범여부',y = 'scaledAmmout', color = 'red', data=df_org)
plt.rcParams['font.family'] = 'Malgun Gothic'
plt.show()
```



신고중량(KG)

```
plt.figure(figsize=(12,8)) # 신고중량
sns.boxplot(x = '우범여부',y = 'scaledKG', color = 'red', data=df_org)
plt.rcParams['font.family'] = 'Malgun Gothic'
plt.show()
```

```
findfont: Font family ['Malgun Gothic'] not found. Falling back to DejaVu Sans.
findfont: Font family ['Malgun Gothic'] not found. Falling back to DejaVu Sans.
```



-> 과세과격원화금액의 경우 우범인 경우에 가격이 낮은 경향을 보였다.

-> 신고 중량의 경우 거의 비슷하나 우범일 경우가 조금 더 낮았다.

새로운 변수들 생성

특송업체부호의 비율을 예시로 새로운 변수를 생성

	unique	rate	counts
0	□	0.155768	51731
1	PR5UFJ	0.379973	5030
2	PAVJZL	0.371569	2798
3	24BG4R	0.572172	1209
4	TQ18AK	0.541779	1131
5	OO4TIW	0.564586	1054
6	YACAWV	0.658845	956
7	QJOYGH	0.782369	647
8	6TFEP5	0.754438	593
9	6RJ8HT	0.863636	527

특송업체부호의 unique값 조사

0	없음	0.673256
1	PR5UFJ	0.065463
2	PAVJZL	0.036415
3	24BG4R	0.015735
4	TQ18AK	0.014719
...	...	...
76	PNE6WR	0.000325
77	ID1L6P	0.000286
78	M72HED	0.000247
79	MWL4TG	0.000221
80	5FDIRV	0.000182

전체 데이터에서 차지하는 비율 조사

특송업체 발송비율
0.014719
0.673256
0.673256
0.673256

조사결과를 새로운 수치형 변수 생성

새로운 변수들 생성

생성된 변수들 :

특송업체발송비율, 통관지세관비율,
해외거래처비율, 운송수단비율

특송업체 발송비율	통관지세 관비율	해외거래 처비율	운송수단 비율
0.014719	0.001783	0.342192	0.467158
0.673256	0.117782	0.342192	0.482658
0.673256	0.160782	0.000104	0.482658
0.673256	0.507659	0.342192	0.482658
0.673256	0.117782	0.000065	0.482658
0.673256	0.507659	0.001171	0.467158
0.036415	0.507659	0.001705	0.482658
0.673256	0.507659	0.001796	0.467158
0.673256	0.160782	0.002980	0.030532

기존 변수들 제거

제거된 변수들 :

신고번호, 신고일자, 검사결과코드,
해외거래처부호, 운송수단유형코드,
특송업체부호, 통관지세관부호

이유 :

해석하기가 어렵거나
우범여부를 판단하는데 영향력이 적기 때문에 제거

불균형 데이터 처리

```
smote = SMOTE(random_state=11, sampling_strategy='all')
df_train_over_x, df_train_over_y = smote.fit_resample(train_part_x, df_train_y)
df_train_over_x.info(), df_train_over_y
```

```
훈련  데이터 비우범건수: 41485, 훈련  데이터 우범건수: 12300
테스트데이터 비우범건수: 17780, 테스트데이터 우범건수: 5272
```



```
훈련  데이터 비우범건수: 41427, 훈련  데이터 우범건수: 41544
테스트데이터 비우범건수: 17838, 테스트데이터 우범건수: 17721
```

SMOTE를 이용해서
불균형 데이터를 해결

SMOTE 적용 후 우범 건 수 비교
12300 -> 41544

원핫인코딩

```
# 원핫인코더 로딩
from sklearn.preprocessing import  OneHotEncoder
# 원핫인코더 활성화
xgbohe = OneHotEncoder(categories="auto", handle_unknown='ignore')
# 훈련데이터: 범주형 변수만 분리하여 원핫인코딩 적용
df_train_cat = xgbohe.fit_transform(df_train[discrete_columns])
df_train_cat = pd.DataFrame(df_train_cat.toarray(),
                            columns = xgbohe.get_feature_names(discrete_columns))
# 수치형 변수 분리
df_train_num = df_train[numeric_columns]

# 원핫인코딩된 범주형 변수와 수치형 변수 재결합
df_train = pd.concat([df_train_num, df_train_cat], axis=1)
```

```
df_eval_cat = xgbohe.fit_transform(df_eval[discrete_columns])
df_eval_cat = pd.DataFrame(df_eval_cat.toarray(),
                           columns = xgbohe.get_feature_names(discrete_columns))
# 수치형 변수 분리
df_eval_num = df_train[numeric_columns]

# 원핫인코딩된 범주형 변수와 수치형 변수 재결합
df_eval_onehot = pd.concat([df_eval_cat, df_eval_num], axis=1)
```

-> 원핫인코딩을 통한 범주형 변수 처리

XG Boost 모델 구축


```
In [42]:
# XGBoost 모델 생성
from xgboost import XGBClassifier
from sklearn.metrics import f1_score,roc_auc_score
import warnings
warnings.filterwarnings("ignore")
```

Scikit learn 라이브러리의 Xgboost 모델인 XGBClassifier 사용

XG Boost 모델

```
In [43]: xgb_clf = XGBClassifier(n_estimators=50, max_depth=4, n_jobs=-1, reg_lambda=1)

         eval_set = [(df_train_over_x, df_train_over_y), (df_test, org_test_y)]

         xgb_clf.fit(df_train_over_x, df_train_over_y, eval_metric=["logloss"],
                     eval_set=eval_set, verbose=True)
```

라이브러리를 통해 불러온 모델의 Hyper Parameter를 조정한 후

전처리 과정과 샘플링을 거친 훈련 데이터와 테스트 데이터를

평가 데이터로 하여 학습을 진행

학습을 진행한 후 모델의 Status

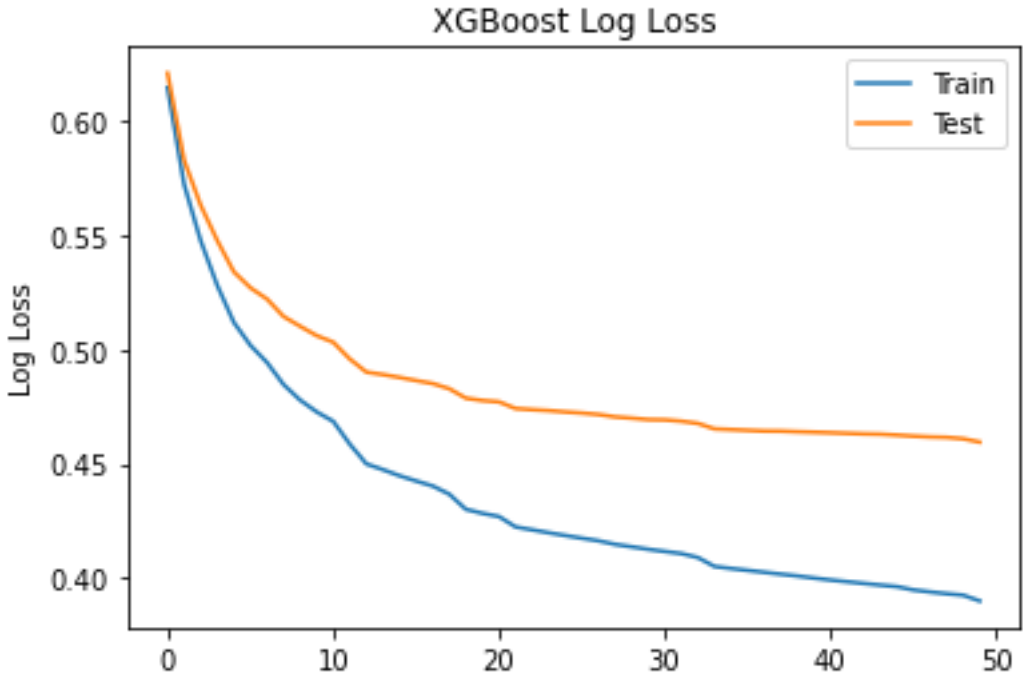
```
[0]    validation_0-logloss:0.61486    validation_1-logloss:0.62122
[1]    validation_0-logloss:0.57242    validation_1-logloss:0.58299
[2]    validation_0-logloss:0.54771    validation_1-logloss:0.56329
[3]    validation_0-logloss:0.52816    validation_1-logloss:0.54783
[4]    validation_0-logloss:0.51193    validation_1-logloss:0.53411
[5]    validation_0-logloss:0.50191    validation_1-logloss:0.52711
[6]    validation_0-logloss:0.49439    validation_1-logloss:0.52219
[7]    validation_0-logloss:0.48470    validation_1-logloss:0.51453
[8]    validation_0-logloss:0.47790    validation_1-logloss:0.51025
[9]    validation_0-logloss:0.47272    validation_1-logloss:0.50613
[10]   validation_0-logloss:0.46856    validation_1-logloss:0.50326
[11]   validation_0-logloss:0.45855    validation_1-logloss:0.49589
[12]   validation_0-logloss:0.44991    validation_1-logloss:0.49014
[13]   validation_0-logloss:0.44740    validation_1-logloss:0.48907
[14]   validation_0-logloss:0.44472    validation_1-logloss:0.48781
[15]   validation_0-logloss:0.44243    validation_1-logloss:0.48645
[16]   validation_0-logloss:0.44024    validation_1-logloss:0.48514
[17]   validation_0-logloss:0.43666    validation_1-logloss:0.48286
[18]   validation_0-logloss:0.43002    validation_1-logloss:0.47873
[19]   validation_0-logloss:0.42825    validation_1-logloss:0.47769
[20]   validation_0-logloss:0.42684    validation_1-logloss:0.47711
[21]   validation_0-logloss:0.42229    validation_1-logloss:0.47419
[22]   validation_0-logloss:0.42104    validation_1-logloss:0.47369
[23]   validation_0-logloss:0.41974    validation_1-logloss:0.47330
[24]   validation_0-logloss:0.41852    validation_1-logloss:0.47269
[25]   validation_0-logloss:0.41735    validation_1-logloss:0.47221
[26]   validation_0-logloss:0.41622    validation_1-logloss:0.47157
[27]   validation_0-logloss:0.41466    validation_1-logloss:0.47052
[28]   validation_0-logloss:0.41359    validation_1-logloss:0.46998
[29]   validation_0-logloss:0.41251    validation_1-logloss:0.46935
[30]   validation_0-logloss:0.41154    validation_1-logloss:0.46923
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=4,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=50, n_jobs=-1, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

모델의 성능 시각화

```
In [44]:
# 모델 성능 시각화

from matplotlib import pyplot
# retrieve performance metrics
results = xgb_clf.evals_result()
epochs = len(results['validation_0']['logloss'])
x_axis = range(0, epochs)
# plot log loss
fig, ax = pyplot.subplots()
ax.plot(x_axis, results['validation_0']['logloss'], label='Train')
ax.plot(x_axis, results['validation_1']['logloss'], label='Test')
ax.legend()
pyplot.ylabel('Log Loss')
pyplot.title('XGBoost Log Loss')
```



훈련과 시험 데이터의 log-loss값을 그래프로 나타냄

검증

```
In [45]:
# evaluate xgboost model
print("------Evaluating xgboost model------")
# Predict
test_pred = xgb_clf.predict_proba(df_test)[:,1]
# Calculate auc
xgb_auc = roc_auc_score(org_test_y, test_pred)
print(xgb_auc)
```

```
------Evaluating xgboost model------
0.822622715716112
```

학습한 모델의 테스트 데이터를 입력하여 예측 값을 계산하여 정확도를 평가

적중률, 적발률 결과

In [46]:
```
def inspection_performance(predicted_fraud, test_fraud):

    Inspect_Rate=[]
    Precision=[]
    Recall=[]

    for i in range(0,101,1):

        threshold = np.percentile(predicted_fraud, i)
        # Precision = number of frauds / number of inspection
        precision = np.mean(test_fraud[predicted_fraud >= threshold])
        # Recall = number of inspected frauds / number of frauds
        recall = sum(test_fraud[predicted_fraud >= threshold])/sum(test_fraud)
        # Save values
        Inspect_Rate.append(100-i)
        Precision.append(precision)
        Recall.append(recall)


    compiled_conf_matrix = pd.DataFrame({
        'Inspect_Rate':Inspect_Rate,
        'Precision':Precision,
        'Recall':Recall
    })

    return compiled_conf_matrix
```

In [47]:
```
basic_performance = inspection_performance(test_pred, org_test_y.astype(float))
```

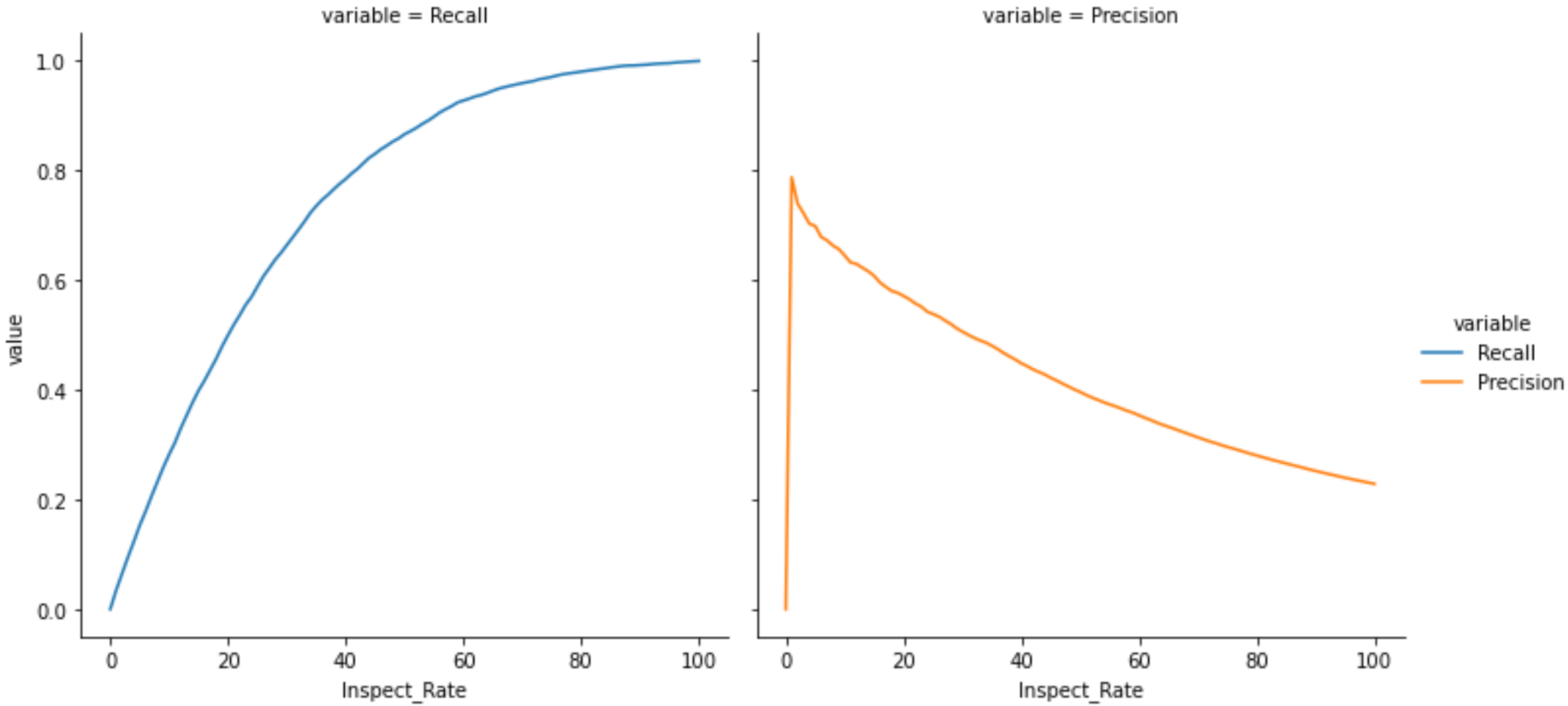
In [48]:
```
# 검사율 1~10% 지정 시 Precision 및 Recall 분석
# Precision (적중률) = (검사선별된 우범건수)/(검사선별 건수)
# Recall (적발률) = (검사선별된 우범건수) / (전체 우범건수)
basic_performance.iloc[range(99,89,-1),:]
```

	Inspect_Rate	Precision	Recall
99	1	0.787879	0.034522
98	2	0.740260	0.064871
97	3	0.722543	0.094841
96	4	0.703142	0.123103
95	5	0.699046	0.152883
94	6	0.679191	0.178300
93	7	0.672862	0.205994
92	8	0.663415	0.232170
91	9	0.656867	0.258536
90	10	0.645273	0.282246

In [49]:
```
data = pd.melt(basic_performance,
               id_vars = ['Inspect_Rate'],
               value_vars = ['Recall','Precision'])

sns.relplot(data=data,
            kind='line',
            x="Inspect_Rate",
            y="value",
            hue='variable',
            col="variable")
```

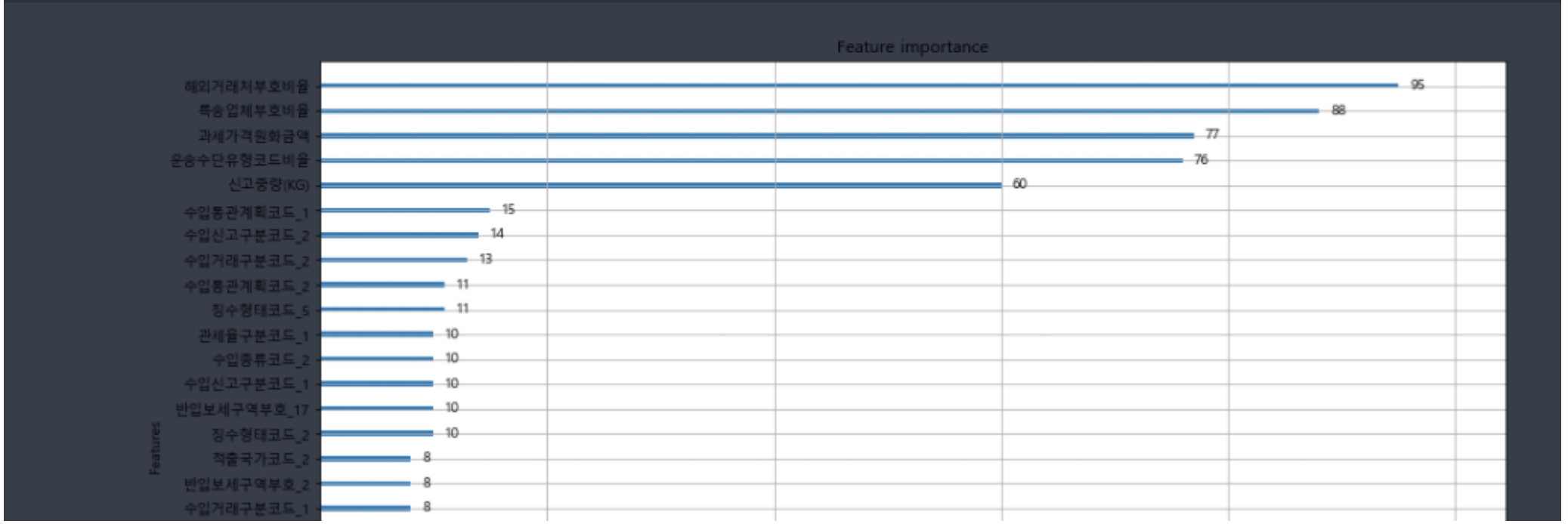
적중률 / 적발률 계산 결과 시각화



Feature Importance 결과

In [50]:

```
# 모델에 기여도가 높은 변수 시각화
from xgboost import plot_importance
plt.rcParams["font.family"] = 'Malgun Gothic'
plt.rcParams["figure.figsize"] = (15,10)
plot_importance(xgb_clf, max_num_features=30)
plt.show()
```



F1 Score & Recall & Precision

```
accuracy score for trained data 0.8303724237676269
accuracy score is 0.7729047371160854
Confusion matrix [[14256  3524]
 [ 1711  3561]]
Report                precision    recall  f1-score   support

           0       0.89      0.80      0.84     17780
           1       0.50      0.68      0.58      5272

    accuracy                           0.77     23052
   macro avg       0.70      0.74      0.71     23052
weighted avg       0.80      0.77      0.78     23052
```

-> 우범이 아닐 때 precision의 경우 높은 성능을 보이고 있으나, 우범일 경우 낮다.

-> recall의 경우 우범이 아닐 때 다소 낮아졌으나, 우범일 때 다소 높아진 것을 알 수 있다.

-> 전체적인 f1-socre 기준으로 평가하면 우리 모델은 우범이 아닐 때 정확도가 높다는 것을 알 수 있다.

√

모델 종합 평가

-> 범주형 변수가 class imbalance 상황에서 많이 사용될 경우 방해 요인이 됨.
해당 모델은 feature 하나, 하나 분석해보며 범주형 변수에서 유의미한 가설을 세웠고
실제로 해당 가설은 이전 모델보다 전체적인 정확도 향상에 도움이 됨.

-> 우범이 아닌 것을 가려내는 부분에 대한 precision이 다소 높기 때문에,
우범이 아니라고 판정 된 신고서의 경우 큰 노력을 드리지 않고 확인할 수 있다.

-> 우범일 경우 recall이 꽤 높기 때문에, 해당 모델을 믿고 우범으로 가려진
신고서 위주로 검토를 진행하면 효율적인 성과를 거둘 것으로 예상함.


Thank You For Listening