In [1]:
```python
import pandas as pd
import numpy as np
from collections import Counter
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from scipy.stats import skew
import scipy.stats as stats
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.ensemble import GradientBoostingClassifier
from lightgbm import LGBMClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score,balanced_accuracy_score,precision_score,reca
from imblearn.metrics import geometric_mean_score
import warnings
warnings.filterwarnings(action='ignore')
```

In [2]:
```python
df = pd.read_csv("Kaggle_Company_Bankruptcy_Prediction.csv")
```

In [3]:
```python
df.isna().sum()
```

Out[3]:
```
Bankrupt                                                   0
 ROA(C) before interest and depreciation before interest    0
 ROA(A) before interest and % after tax                     0
 ROA(B) before interest and depreciation after tax          0
 Operating Gross Margin                                     0
                                                           ..
 Liability to Equity                                        0
 Degree of Financial Leverage (DFL)                         0
 Interest Coverage Ratio (Interest expense to EBIT)         0
 Net Income Flag                                            0
 Equity to Liability                                        0
Length: 96, dtype: int64
```

In [4]:
```python
df.duplicated().sum()
```

Out[4]:
```
0
```

In [5]:
```python
df.shape
```

Out[5]:
```
(6819, 96)
```

In [6]:
```python
df[' Net Income Flag'].value_counts()
```

Out[6]:
```
1    6819
Name:  Net Income Flag, dtype: int64
```

In [7]:
```python
df = df.drop([' Net Income Flag'],axis=1) #단일값 변수 우선 삭제
```

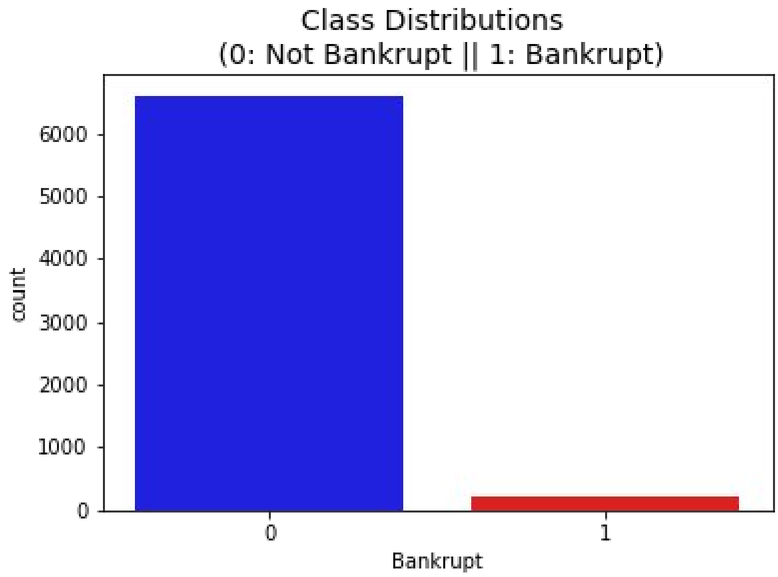
In [8]:
```python
# 클래스 비율
print('Class Count','\n',df['Bankrupt'].value_counts(),'\n')
```

```
print('Stable', round(df['Bankrupt'].value_counts()[0]/len(df) * 100,2), '% of the data
print('Unstable', round(df['Bankrupt'].value_counts()[1]/len(df) * 100,2), '% of the da
plt.title('Class Distributions \n (0: Not Bankrupt || 1: Bankrupt)', fontsize=14)
colors = ["blue", "red"]
sns.countplot(df['Bankrupt'],palette=colors)
```

```
Class Count
 0    6599
1     220
Name: Bankrupt, dtype: int64

Stable 96.77 % of the dataset
Unstable 3.23 % of the dataset
```

Out[8]:

```
<AxesSubplot:title={'center':'Class Distributions \n (0: Not Bankrupt || 1: Bankrupt)'},
xlabel='Bankrupt', ylabel='count'>
```



In [9]:

```
#df.info()
```

In [10]:

```
df[' Liability-Assets Flag'].value_counts()
```

Out[10]:

```
0    6811
1       8
Name:  Liability-Assets Flag, dtype: int64
```

In [11]:

```
df.groupby('Bankrupt')[' Liability-Assets Flag'].value_counts() #변수 선택법에서 유용하지
```

Out[11]:

```
Bankrupt   Liability-Assets Flag
0          0                         6597
           1                            2
1          0                          214
           1                            6
Name:  Liability-Assets Flag, dtype: int64
```

In [12]:

```
df = df.drop([' Liability-Assets Flag'],axis=1,inplace=False)
```

Feature Selection

Lasso Penalty(L1) --> 로지스틱 회귀에 접합하기 위한 데이터 사전에 표준화

In [13]:

```
X = df.drop(['Bankrupt'],axis=1,inplace=False)
```

```
y = df['Bankrupt']
```

In [14]:

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled_data = pd.DataFrame(X_scaled, index=X.index, columns=X.columns)
```

In [15]:

```
X_train,X_test,y_train,y_test = train_test_split(X_scaled_data,y,test_size=0.3,random_s
```

In [16]:

```
#param = {'C': [10**-2,10**-1,10**0,10**1,10**2]} # Best C : 0.1

param = {'C': [0.1]}
lr_model = LogisticRegression(penalty='l1', solver='liblinear')
gs_model = GridSearchCV(estimator=lr_model, param_grid=param)
gs_model.fit(X_train, y_train)

# Train a LR model with best parameters
model = LogisticRegression(**gs_model.best_params_, penalty='l1', solver='liblinear')
model.fit(X_train, y_train)
```

Out[16]:

```
LogisticRegression(C=0.1, penalty='l1', solver='liblinear')
```

In [17]:

```
coef = model.coef_[0]
red_features = pd.Series(X.columns)[list(coef==0)]
imp_features = pd.Series(X.columns)[list(coef!=0)]
```

In [18]:

```
print('Important Features Count:',sum(coef!=0))
print('Important Features Name:',list(imp_features))
```

```
Important Features Count: 34
Important Features Name: [' ROA(C) before interest and depreciation before interest', '
ROA(B) before interest and depreciation after tax', ' Pre-tax net Interest Rate', ' Oper
ating Expense Rate', ' Net Value Per Share (B)', ' Persistent EPS in the Last Four Seaso
ns', ' Regular Net Profit Growth Rate', ' Total Asset Growth Rate', ' Cash Reinvestment
%', ' Quick Ratio', ' Total debt/Total net worth', ' Debt ratio %', ' Net worth/Assets',
' Borrowing dependency', ' Inventory and accounts receivable/Net value', ' Total Asset T
urnover', ' Accounts Receivable Turnover', ' Fixed Assets Turnover Frequency', ' Net Wor
th Turnover Rate (times)', ' Revenue per person', ' Operating profit per person', ' Cas
h/Total Assets', ' Cash/Current Liability', ' Current Liability to Assets', ' Inventory/
Working Capital', ' Working Capital/Equity', ' Total expense/Assets', ' Quick Asset Turn
over Rate', ' Cash Turnover Rate', ' Fixed Assets to Assets', ' Net Income to Total Asse
ts', ' Total assets to GNP price', ' No-credit Interval', ' Degree of Financial Leverage
(DFL)']
```

In [19]:

```
print('Redundant Features Count:',sum(coef==0))
#print('Redundant Features Name:',list(red_features))
```

```
Redundant Features Count: 59
```

In [20]:

```
data_df = df.drop(df[red_features],axis=1,inplace=False)
```

In [21]:

```
# Selected Features
X = data_df.drop(['Bankrupt'],axis=1,inplace=False)
y = data_df['Bankrupt']
```

In [22]:

```python
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=2021,st
```

시각화 & 요약통계량 --> 34개 변수

In [23]:

```python
pd.set_option('display.max_columns', None)
data_df.groupby('Bankrupt').describe()
```

Out[23]:

	ROA(C) before interest and depreciation before interest									
	count	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**	**count**	**mean**
Bankrupt										
0	6599.0	0.508069	0.057694	0.000000	0.478623	0.504314	0.537074	1.000000	6599.0	0.556659
1	220.0	0.418503	0.081068	0.024277	0.391703	0.441330	0.469276	0.576951	220.0	0.461483

In [24]:

```python
correlation_matrix = data_df.corr()
correlation_matrix.style.background_gradient(sns.light_palette('blue', as_cmap= True))
```

Out[24]:

	Bankrupt	**ROA(C) before interest and depreciation before interest**	**ROA(B) before interest and depreciation after tax**	**Pre-tax net Interest Rate**	**Operating Expense Rate**	**Net Value Per Share (B)**	**Persistent EPS in the Last Four Seasons**	
Bankrupt	1.000000	-0.260807	-0.273051	-0.008517	-0.006083	-0.165399	-0.219560	-(
ROA(C) before interest and depreciation before interest	-0.260807	1.000000	0.986849	0.053419	0.066869	0.505580	0.775006	(
ROA(B) before interest and depreciation after tax	-0.273051	0.986849	1.000000	0.053726	0.065602	0.502052	0.764597	(
Pre-tax net Interest Rate	-0.008517	0.053419	0.053726	1.000000	0.014247	0.033034	0.033726	(
Operating Expense Rate	-0.006083	0.066869	0.065602	0.014247	1.000000	0.090519	0.080969	(
Net Value Per Share (B)	-0.165399	0.505580	0.502052	0.033034	0.090519	1.000000	0.755568	(
Persistent EPS in the Last Four Seasons	-0.219560	0.775006	0.764597	0.033726	0.080969	0.755568	1.000000	(
Regular Net Profit Growth Rate	-0.036820	0.115040	0.117042	0.034914	0.009511	0.056518	0.086083	1
Total Asset Growth Rate	-0.044431	0.019635	0.022104	0.037633	0.014168	-0.018871	-0.036743	(
Cash Reinvestment %	-0.051345	0.296158	0.292008	0.017801	-0.003016	0.090651	0.165224	(

Out[24]:

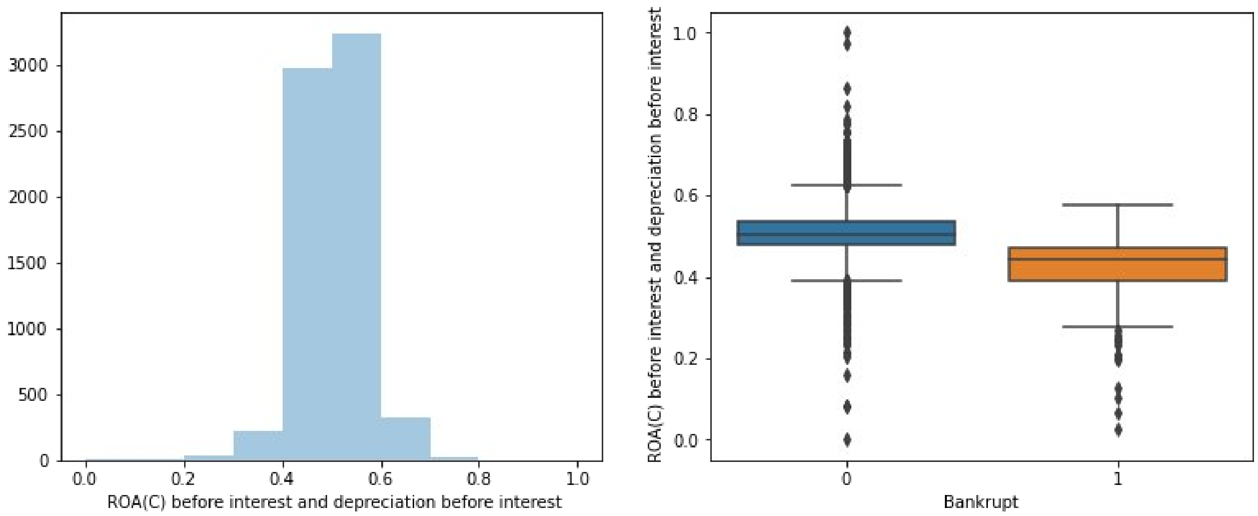
	Bankrupt	ROA(C) before interest and depreciation before interest	ROA(B) before interest and depreciation after tax	Pre-tax net Interest Rate	Operating Expense Rate	Net Value Per Share (B)	Persistent EPS in the Last Four Seasons	
Quick Ratio	0.025058	-0.026336	-0.024232	-0.017376	0.017687	-0.002909	-0.004244	0
Total debt/Total net worth	0.012314	-0.022208	-0.021161	-0.001964	-0.016164	0.008546	-0.011383	0
Debt ratio %	0.250161	-0.261427	-0.264734	-0.003906	0.143833	-0.249146	-0.177429	-0
Net worth/Assets	-0.250161	0.261427	0.264734	0.003906	-0.143833	0.249146	0.177429	0
Borrowing dependency	0.176543	-0.161671	-0.158618	-0.004654	0.023977	-0.123991	-0.144138	-0
Inventory and accounts receivable/Net value	0.075278	-0.109888	-0.109501	0.009042	0.079747	-0.089396	-0.037986	-0
Total Asset Turnover	-0.067915	0.210622	0.194810	0.029667	0.195063	0.082026	0.214710	0
Accounts Receivable Turnover	-0.004754	-0.033947	-0.033768	0.089576	-0.028331	-0.018647	-0.019997	-0
Fixed Assets Turnover Frequency	0.072818	-0.065919	-0.061046	0.003581	-0.055160	-0.080971	-0.129457	-0
Net Worth Turnover Rate (times)	0.021089	0.022896	0.012763	0.015513	0.165135	-0.032829	0.066033	0
Revenue per person	0.039718	-0.014834	-0.014545	-0.144956	-0.010492	-0.017799	-0.011412	-0
Operating profit per person	-0.092842	0.301996	0.304522	0.020271	0.126869	0.261330	0.351589	0
Cash/Total Assets	-0.100130	0.235314	0.227144	0.017136	-0.110605	0.185621	0.240956	0
Cash/Current Liability	0.077921	-0.046009	-0.041296	-0.001404	0.024258	-0.033078	-0.034404	-0
Current Liability to Assets	0.194494	-0.210256	-0.217186	0.001632	0.135256	-0.198546	-0.097689	-0
Inventory/Working Capital	-0.001906	-0.004447	-0.001616	0.010231	-0.008018	-0.005685	0.000283	0
Working Capital/Equity	-0.147221	0.103819	0.101962	0.014933	0.007324	0.070112	0.121854	0
Total expense/Assets	0.139049	-0.296019	-0.322223	-0.004525	-0.249426	-0.235573	-0.177996	-0
Quick Asset Turnover Rate	0.025814	-0.027280	-0.029928	0.012206	0.153936	-0.043038	-0.029352	0
Cash Turnover Rate	-0.018035	-0.029477	-0.030410	0.015581	0.040730	-0.054775	-0.034256	0

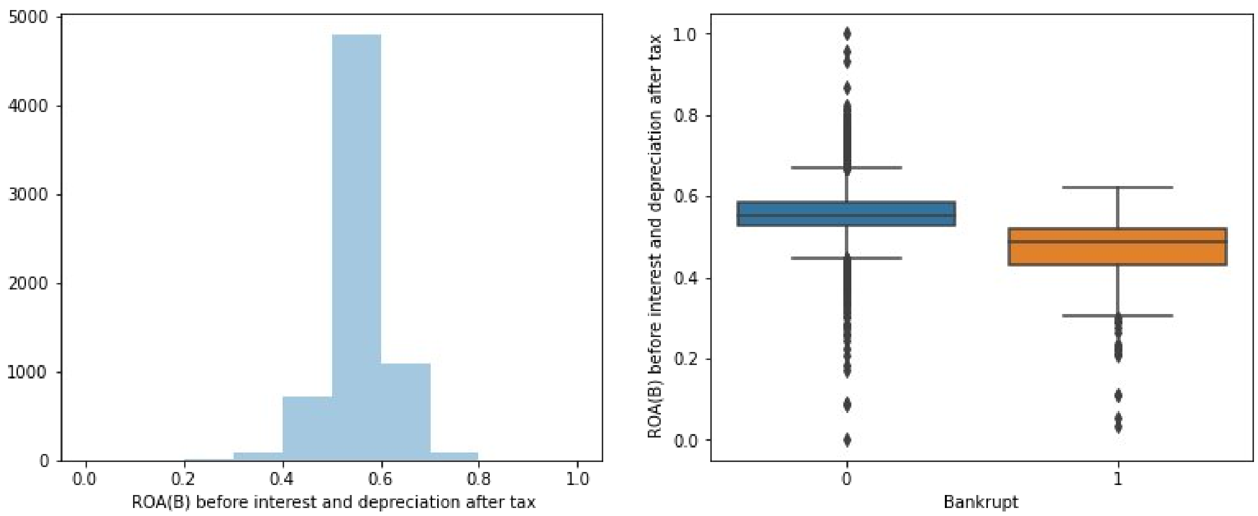
Out[24]:

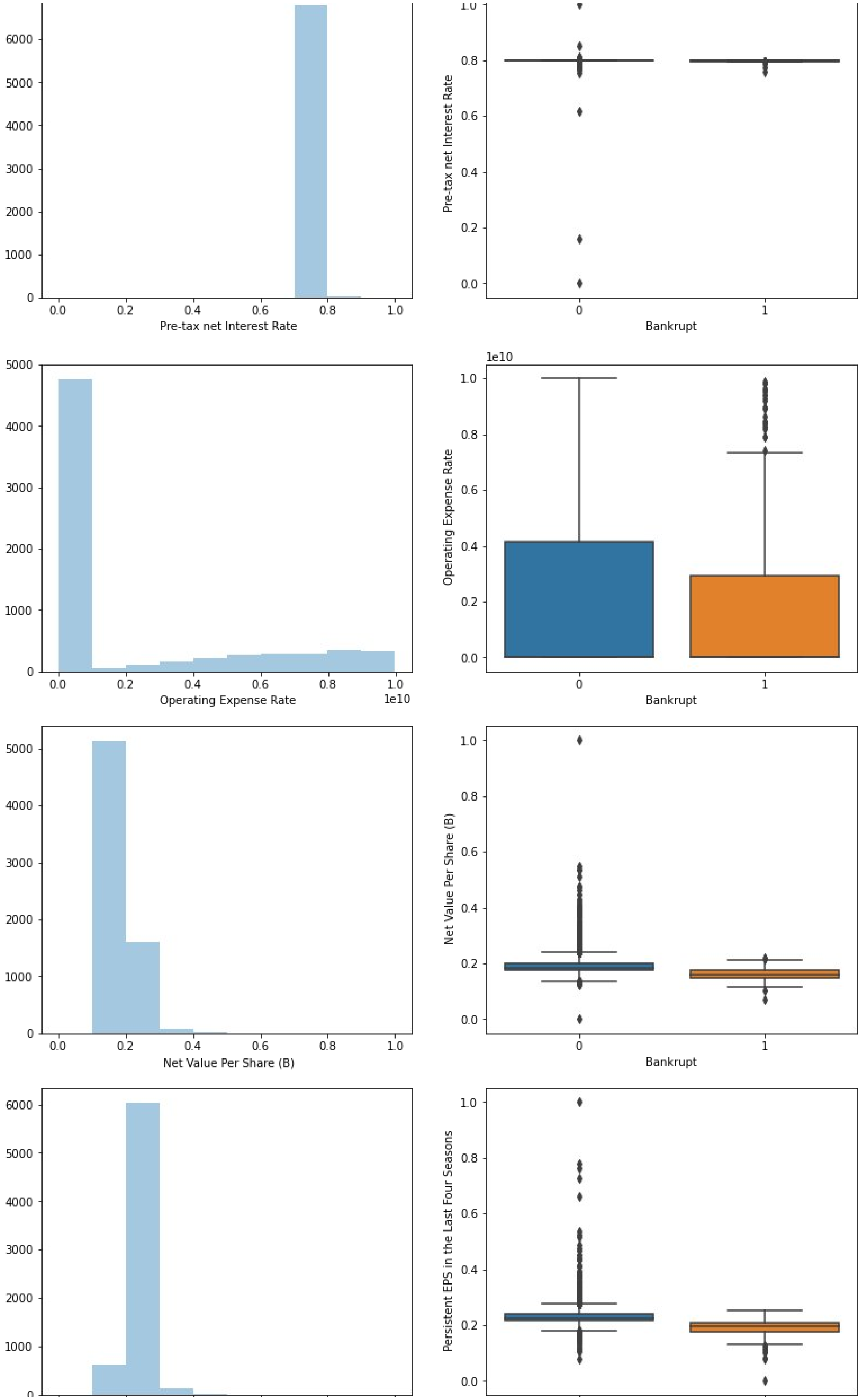
	Bankrupt	ROA(C) before interest and depreciation before interest	ROA(B) before interest and depreciation after tax	Pre-tax net Interest Rate	Operating Expense Rate	Net Value Per Share (B)	Persistent EPS in the Last Four Seasons
Fixed Assets to Assets	0.066328	-0.009192	-0.008364	-0.000047	-0.007464	-0.003112	-0.006477
Net Income to Total Assets	-0.315457	0.887670	0.912040	0.048587	0.071365	0.493776	0.691152
Total assets to GNP price	0.035104	-0.071725	-0.089088	-0.004243	-0.025524	-0.059970	-0.033509
No-credit Interval	-0.005547	0.008135	0.007523	-0.000075	0.006497	0.014303	0.003791
Degree of Financial Leverage (DFL)	0.010508	-0.016575	-0.014663	0.000855	0.013577	-0.021860	-0.018829

In [25]:

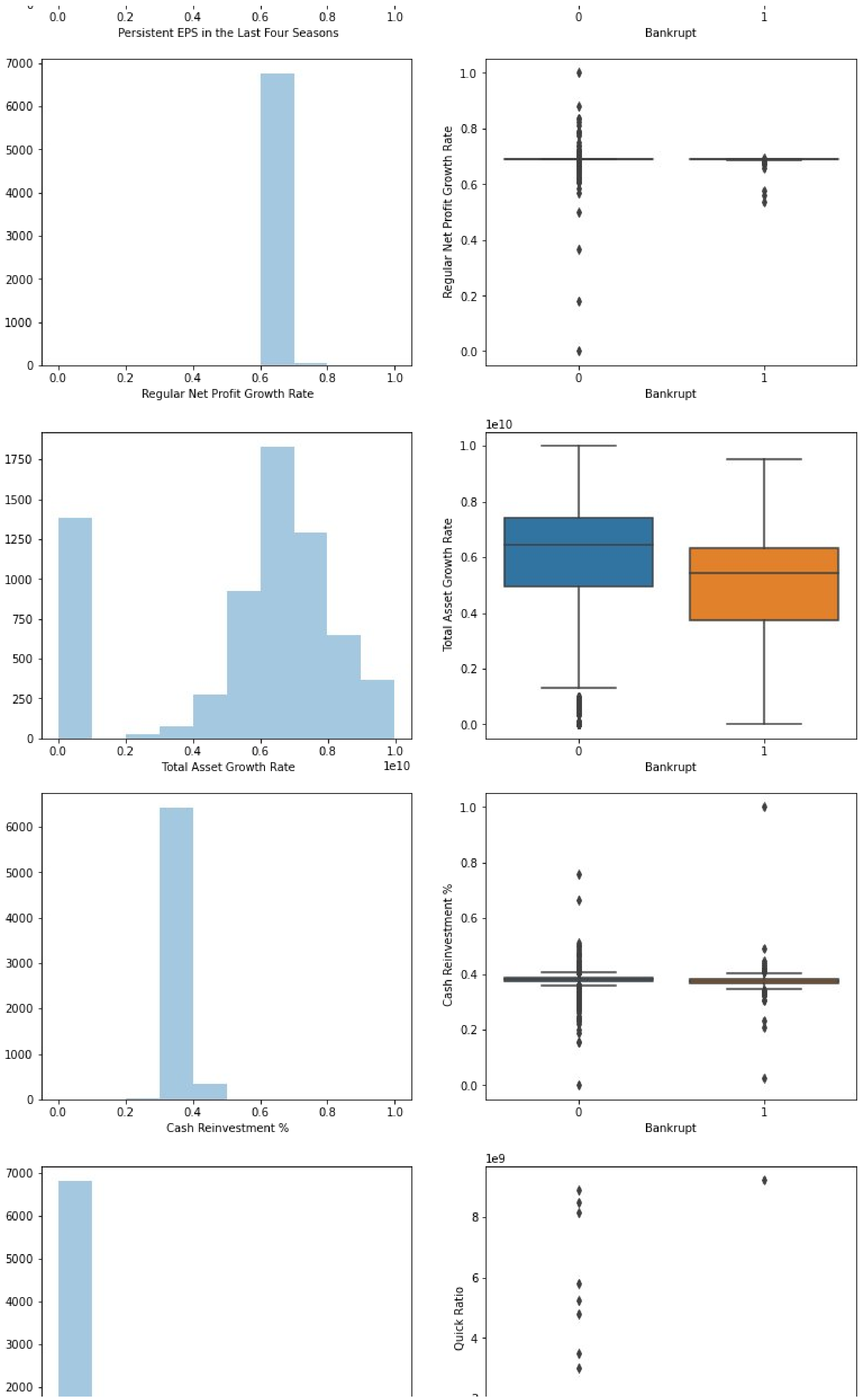
```python
cols = data_df.drop(['Bankrupt'], axis=1).columns

for feature in cols:
    fig, ax = plt.subplots(ncols=2,figsize=(13,5))
    sns.distplot(data_df[feature], bins=10, kde=False,ax=ax[0])
    sns.boxplot('Bankrupt', data_df[feature], data=data_df,ax=ax[1])
```



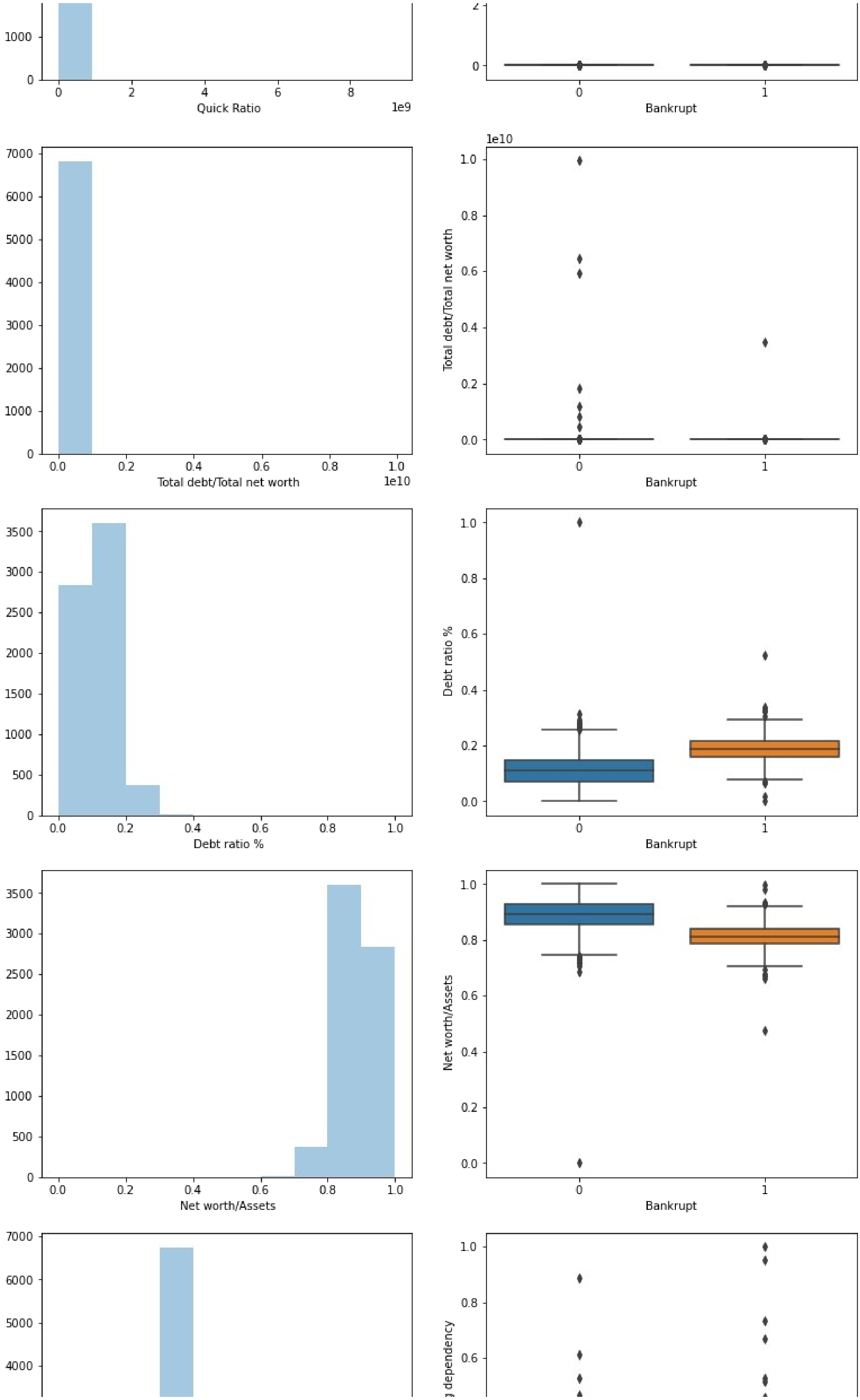




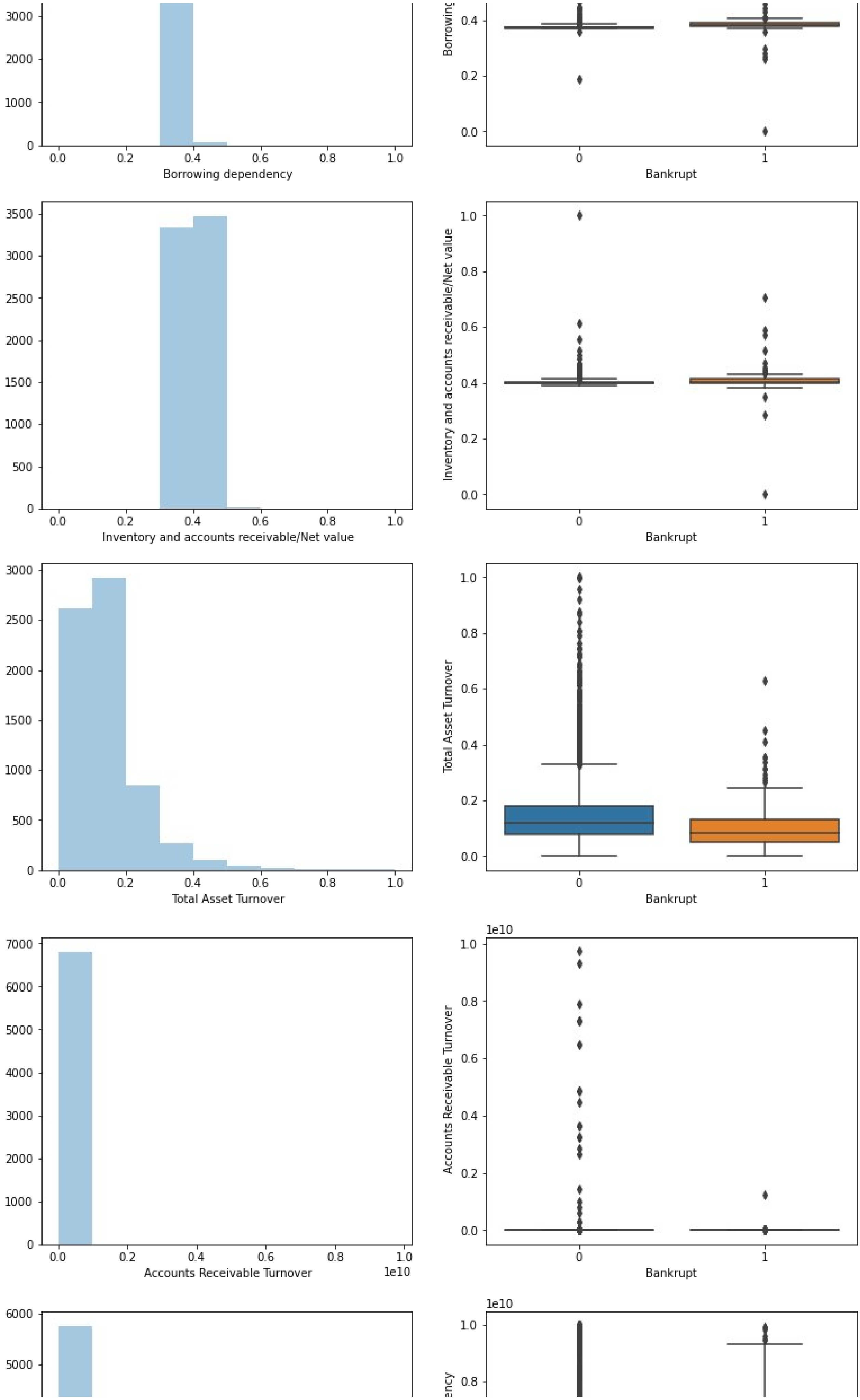
Pre-tax net Interest Rate
Bankrupt
Pre-tax net Interest Rate
Operating Expense Rate
1e10
Bankrupt
Operating Expense Rate
1e10
Net Value Per Share (B)
Bankrupt
Net Value Per Share (B)
Persistent EPS in the Last Four Seasons



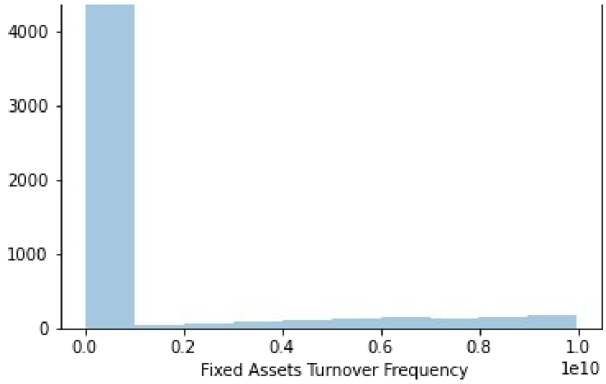
Persistent EPS in the Last Four Seasons
Bankrupt
Regular Net Profit Growth Rate
Bankrupt
Total Asset Growth Rate
1e10
Bankrupt
Cash Reinvestment %
Bankrupt
Quick Ratio
1e9

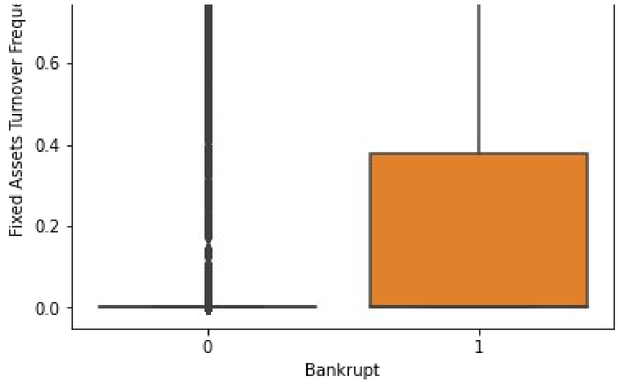


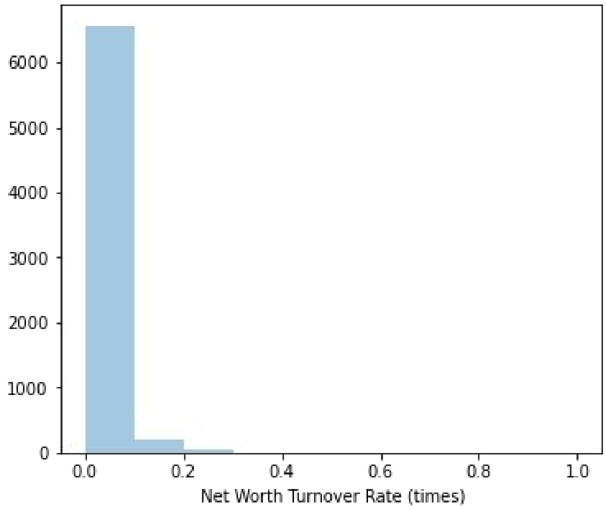
Quick Ratio
1e9
Bankrupt
1e10
Total debt/Total net worth
Bankrupt
Debt ratio %
Bankrupt
Net worth/Assets
Bankrupt
g dependency

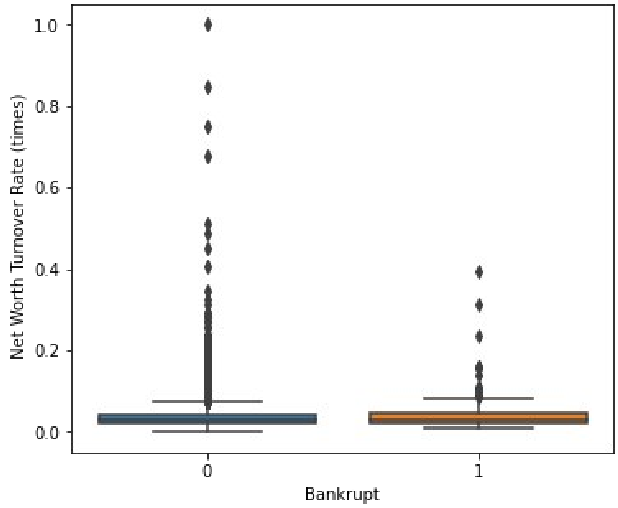


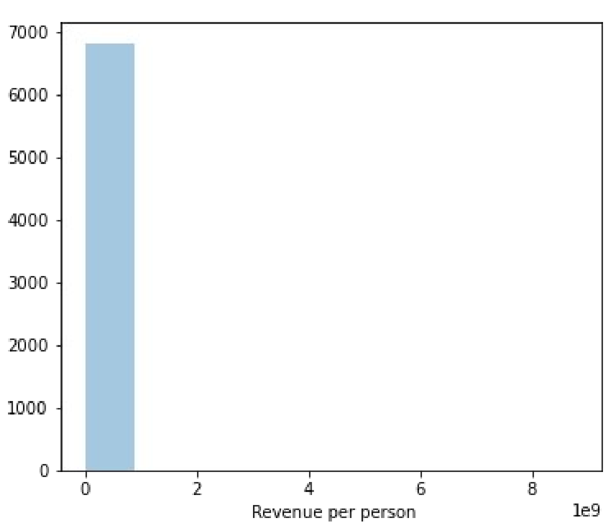
Borrowing dependency
Bankrupt
Inventory and accounts receivable/Net value
Bankrupt
Total Asset Turnover
Bankrupt
Accounts Receivable Turnover
1e10
Bankrupt
1e10

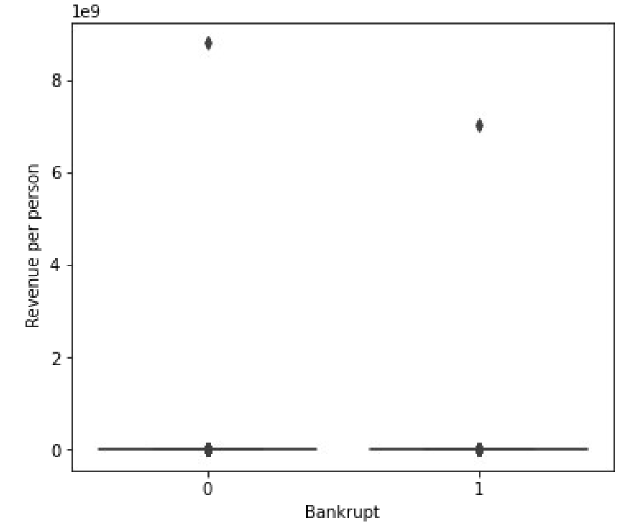

Fixed Assets Turnover Frequency
1e10

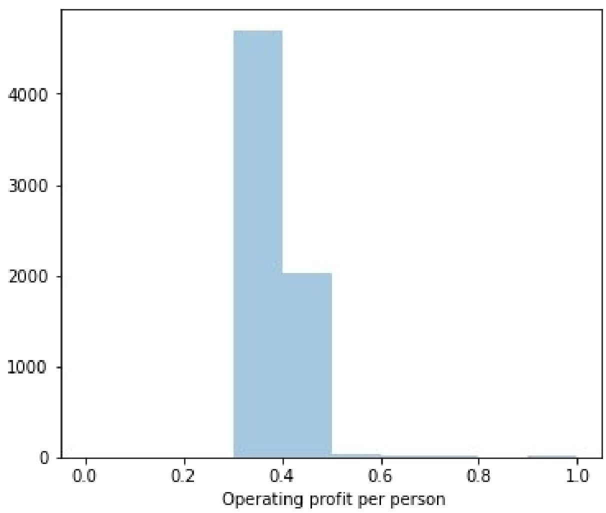

Fixed Assets Turnover Freque
Bankrupt

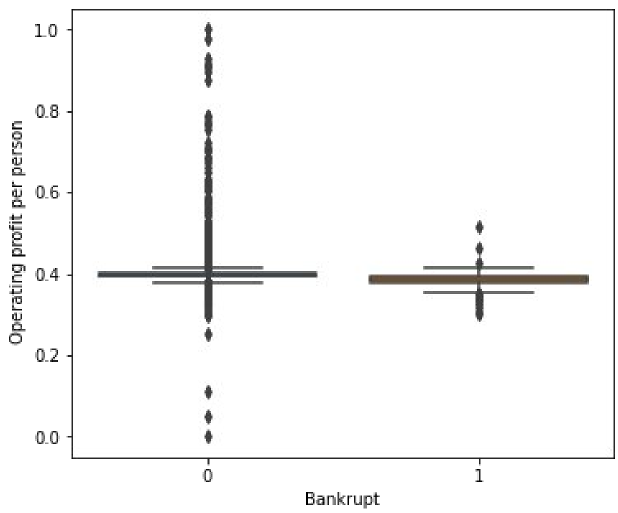

Net Worth Turnover Rate (times)

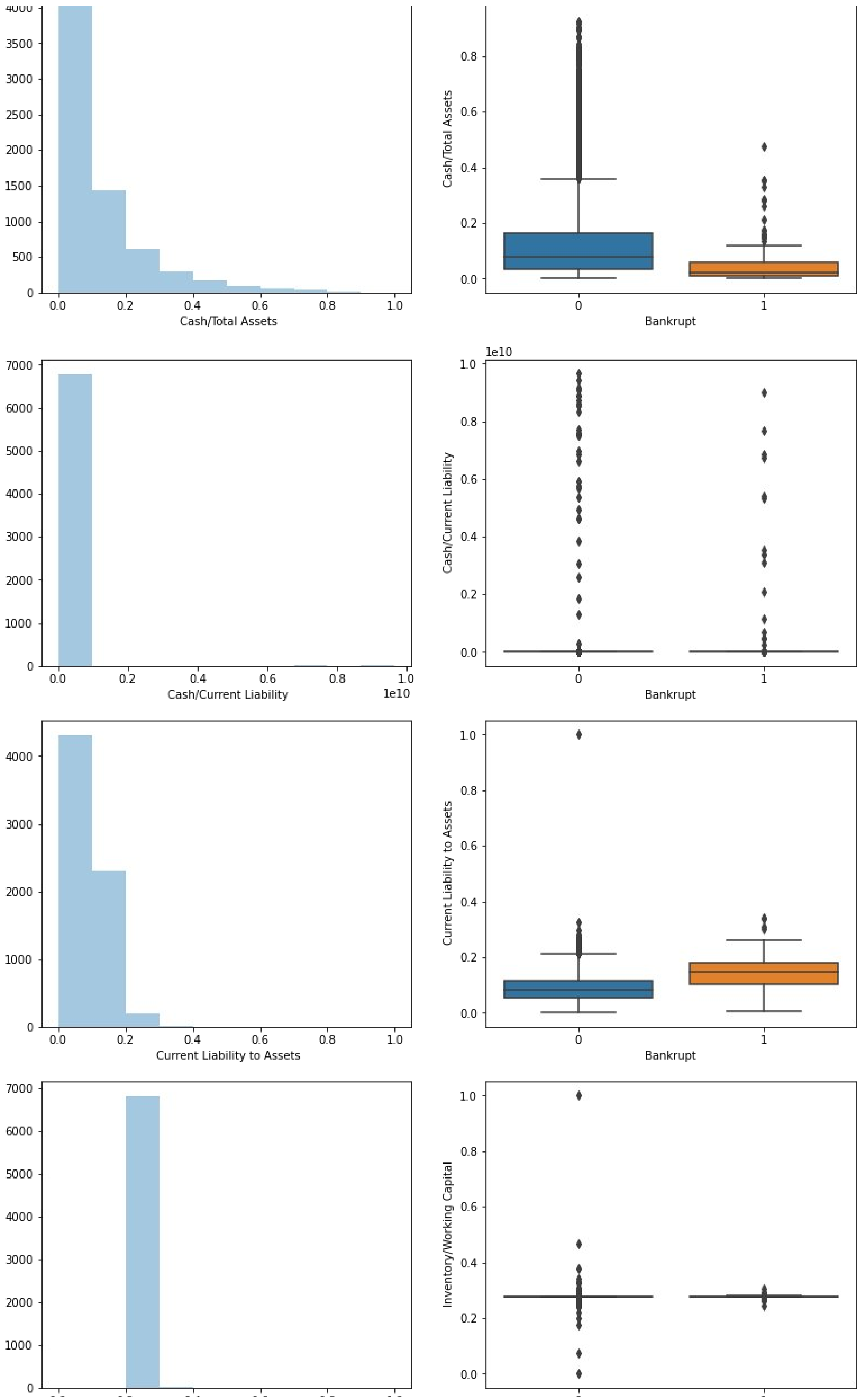

Net Worth Turnover Rate (times)
Bankrupt


Revenue per person
1e9

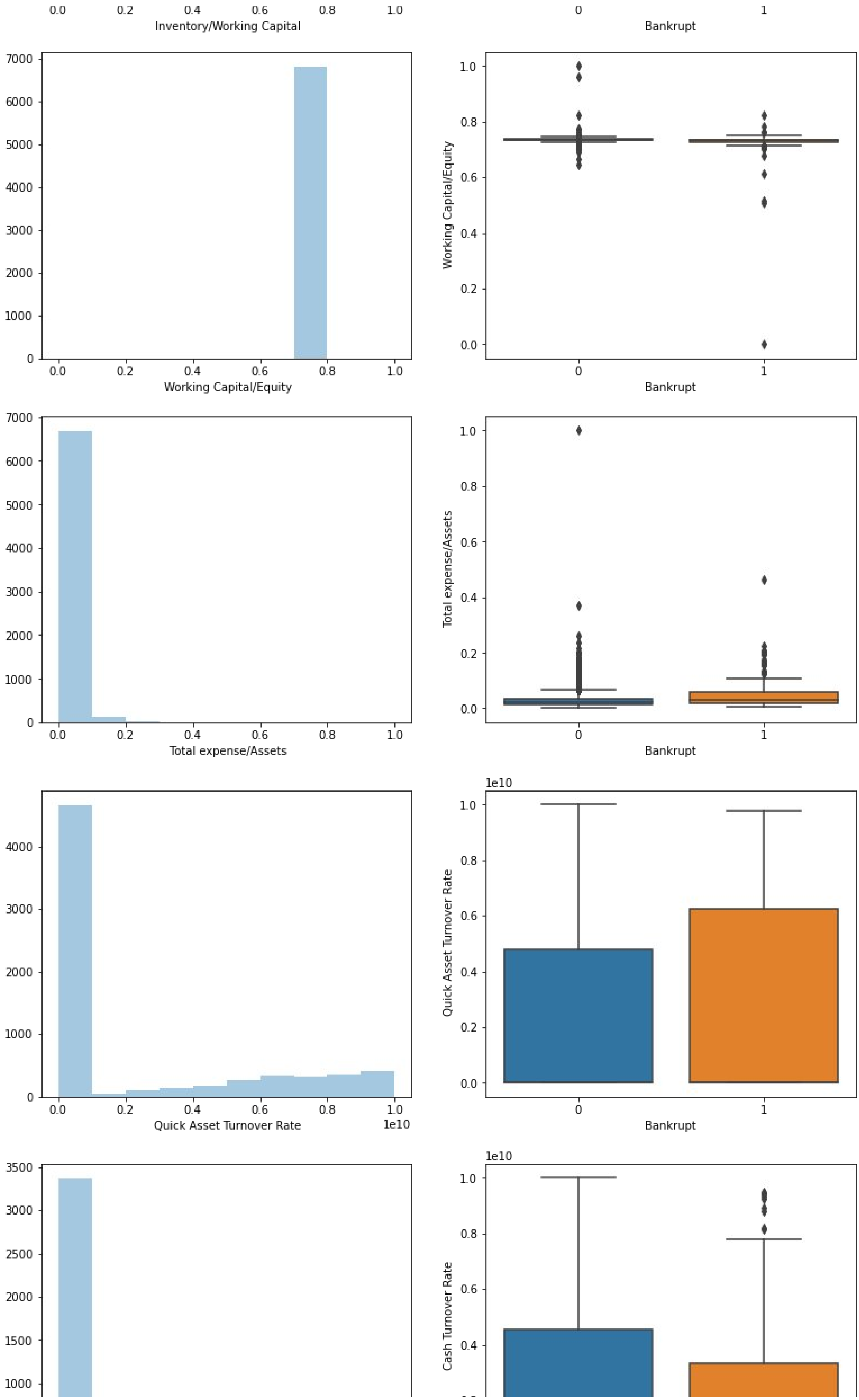

1e9
Revenue per person
Bankrupt


Operating profit per person

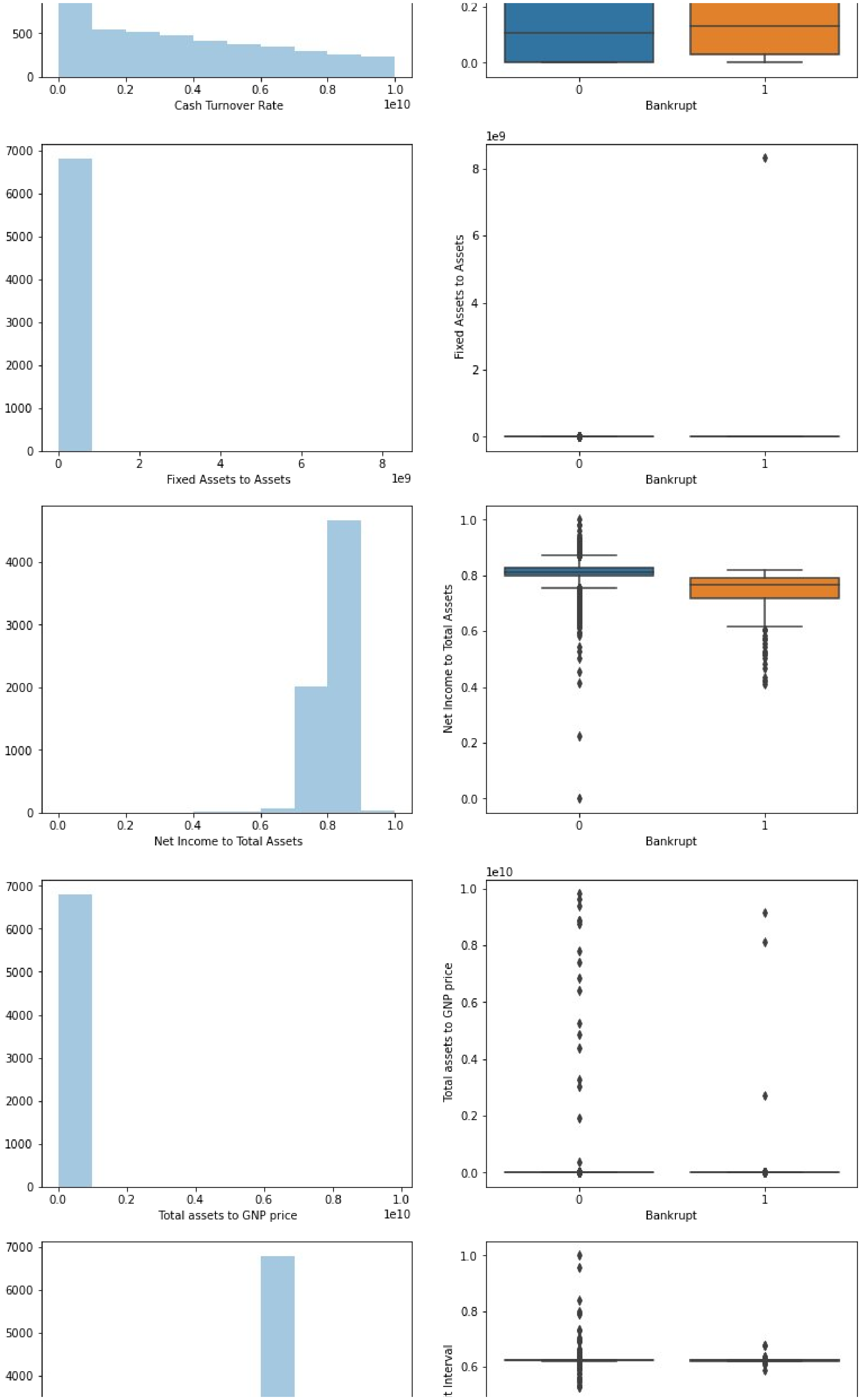

Operating profit per person
Bankrupt

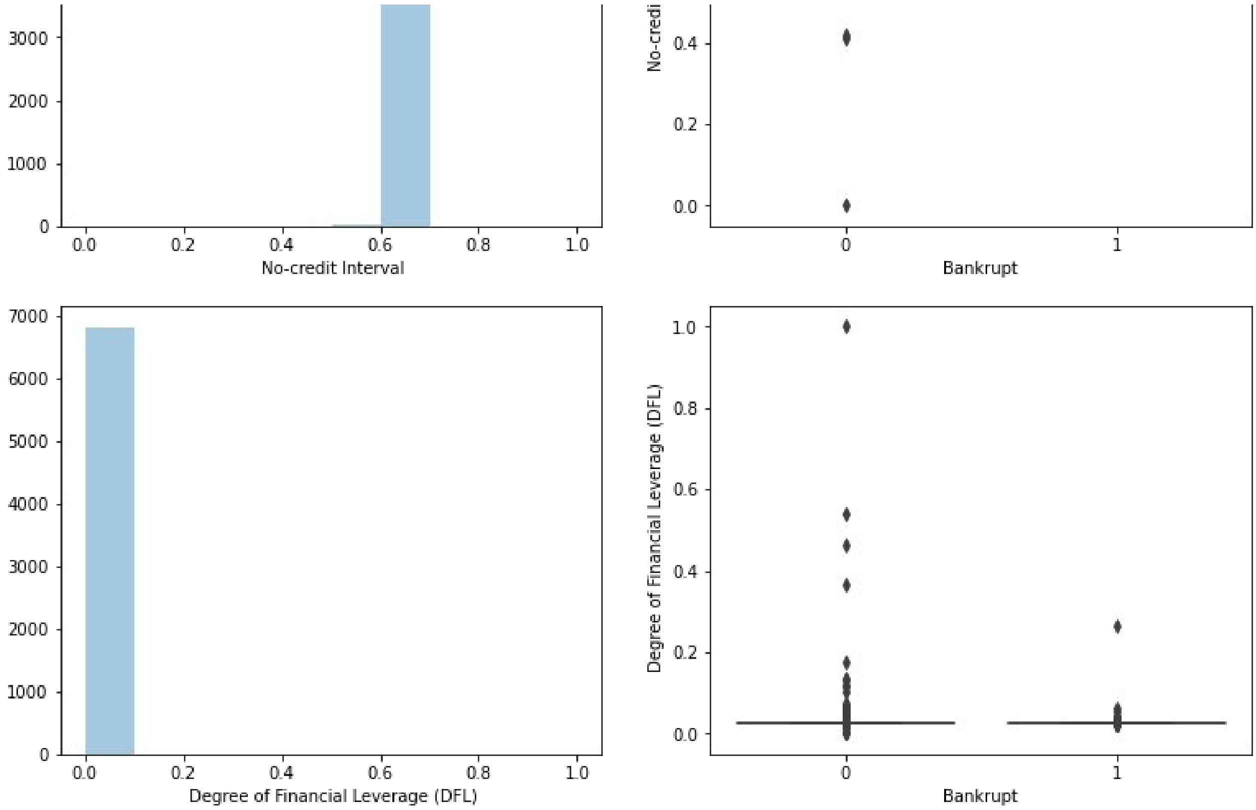


Cash/Total Assets
Bankrupt
Cash/Current Liability
1e10
Current Liability to Assets
Inventory/Working Capital



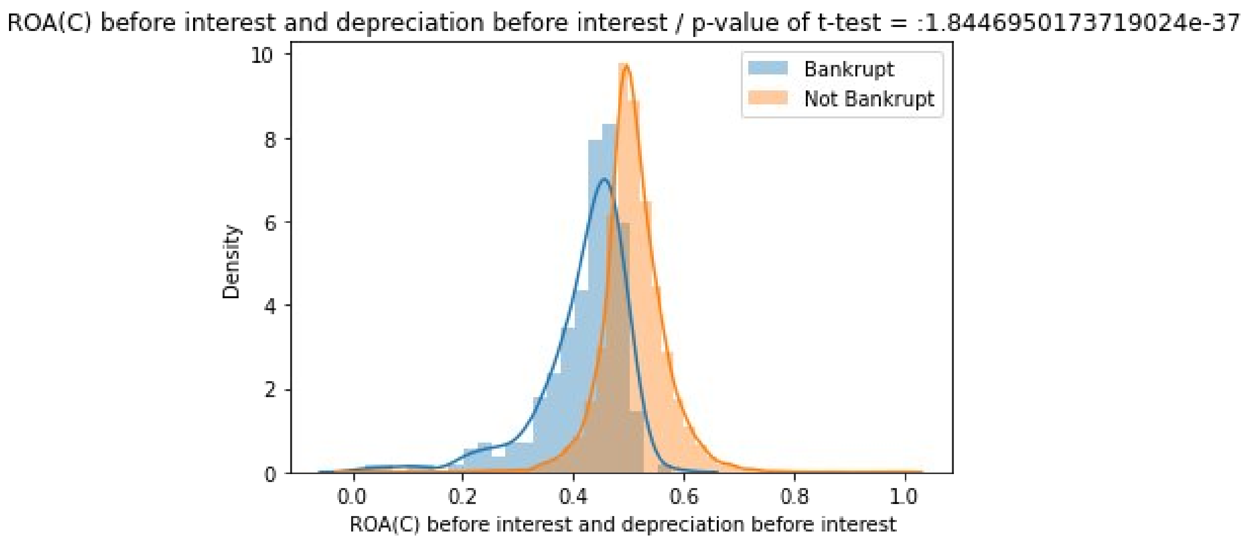
Inventory/Working Capital
Bankrupt
Working Capital/Equity
Bankrupt
Working Capital/Equity
Total expense/Assets
Bankrupt
Total expense/Assets
Quick Asset Turnover Rate
1e10
Bankrupt
Quick Asset Turnover Rate
1e10
Cash Turnover Rate
1e10

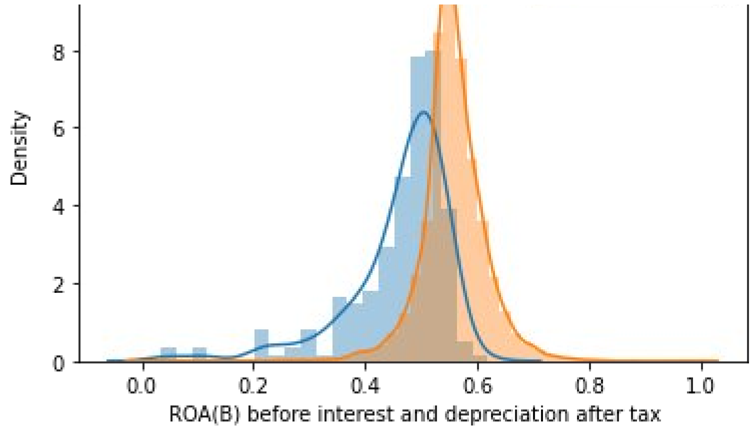

Cash Turnover Rate
1e10
Bankrupt
Fixed Assets to Assets
1e9
Bankrupt
Net Income to Total Assets
Bankrupt
Total assets to GNP price
1e10
Bankrupt
t Interval



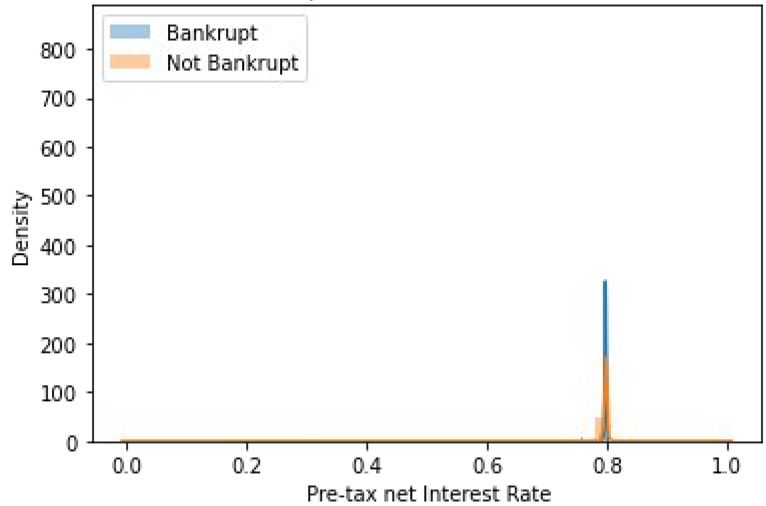
In [26]:

```
bankrupt = data_df[data_df['Bankrupt']==1]
not_bankrupt = data_df[data_df['Bankrupt']==0]
cols = data_df.drop(['Bankrupt'], axis=1).columns

for feature in cols:
    a = bankrupt[feature]
    b = not_bankrupt[feature]
    b = b.sample(n=len(a), random_state=2021)
    test = stats.ttest_ind(a,b)
    plt.figure()
    sns.distplot(bankrupt[feature], kde= True, label="Bankrupt")
    sns.distplot(not_bankrupt[feature], kde= True, label="Not Bankrupt")
    plt.title("{} / p-value of t-test = :{}".format(feature, test[1]))
    plt.legend()
```

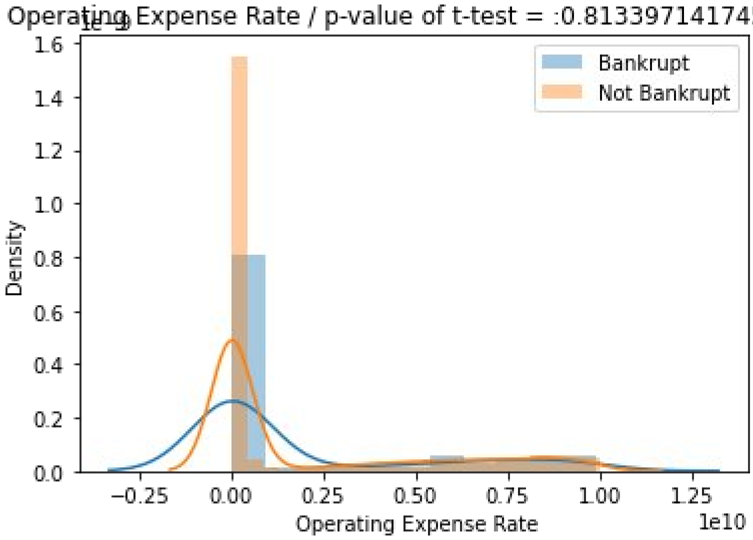




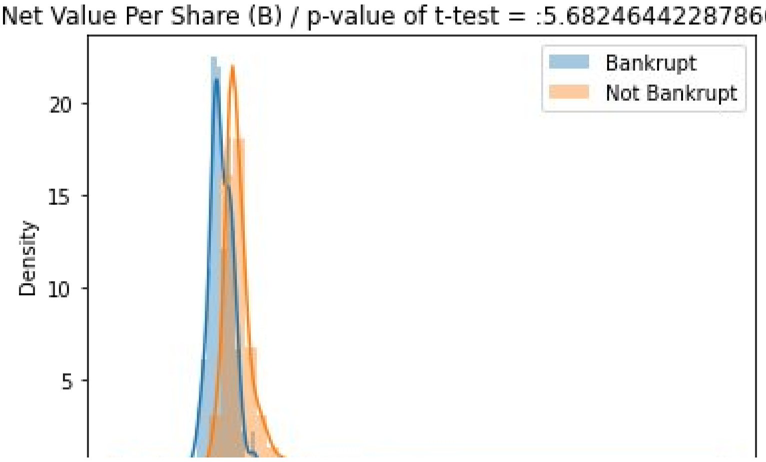
Pre-tax net Interest Rate / p-value of t-test = :0.00010762626717843968

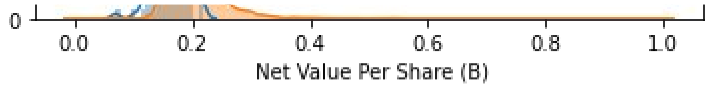


Operating Expense Rate / p-value of t-test = :0.8133971417453854

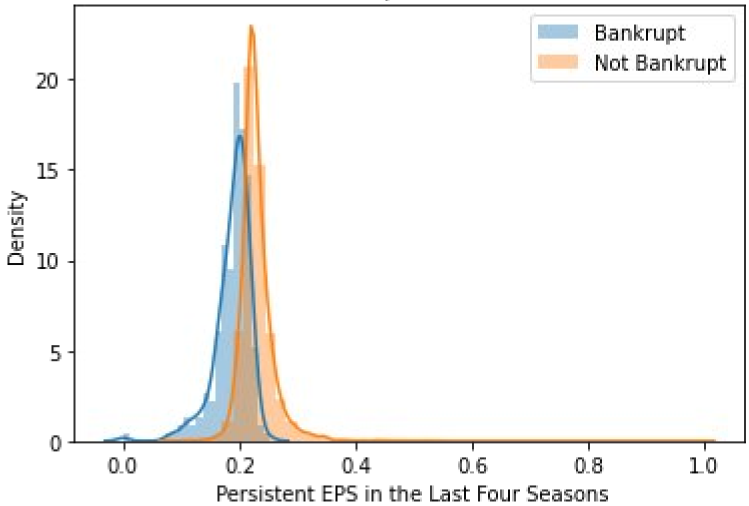


Net Value Per Share (B) / p-value of t-test = :5.682464422878665e-34

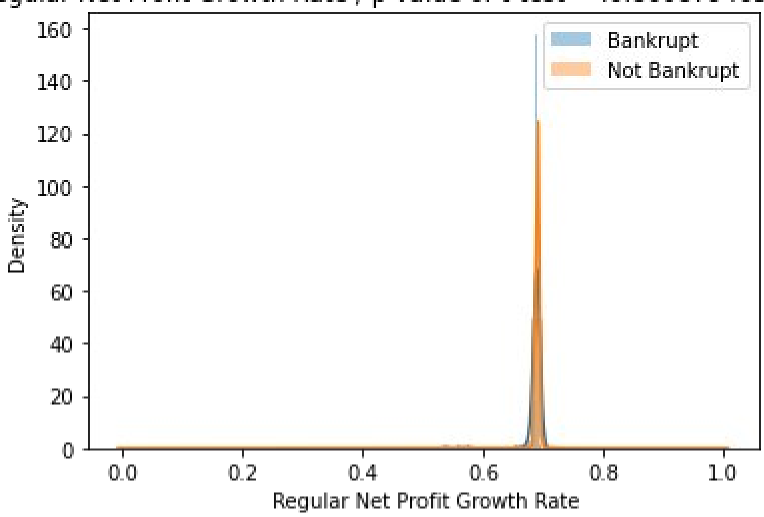




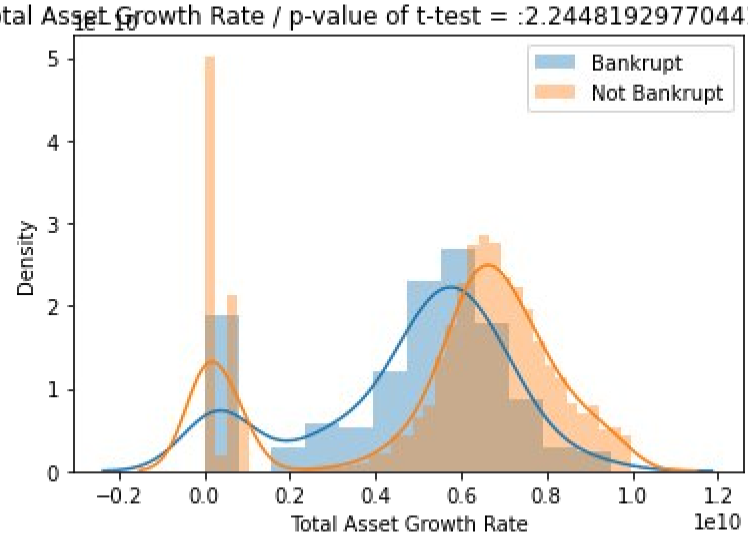
Persistent EPS in the Last Four Seasons / p-value of t-test = :1.1434723432433157e-42



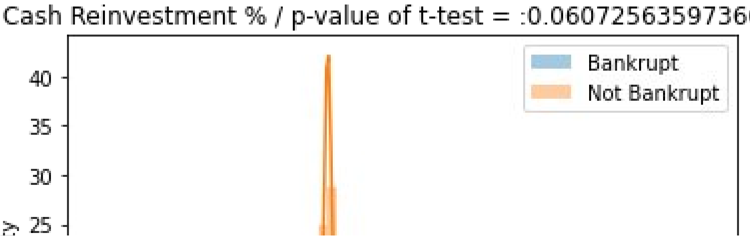
Regular Net Profit Growth Rate / p-value of t-test = :0.8608794655590564

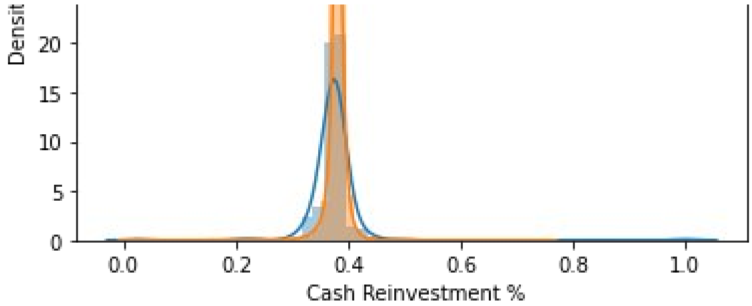


Total Asset Growth Rate / p-value of t-test = :2.2448192977044141e-07

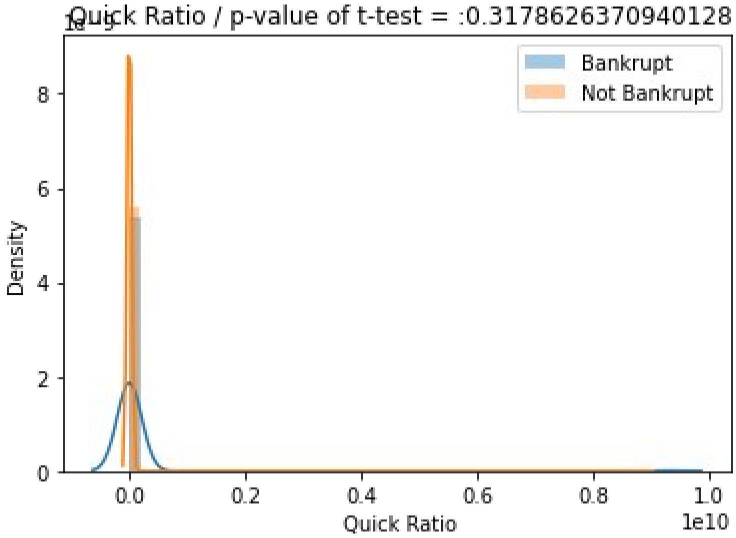


Cash Reinvestment % / p-value of t-test = :0.06072563597366525

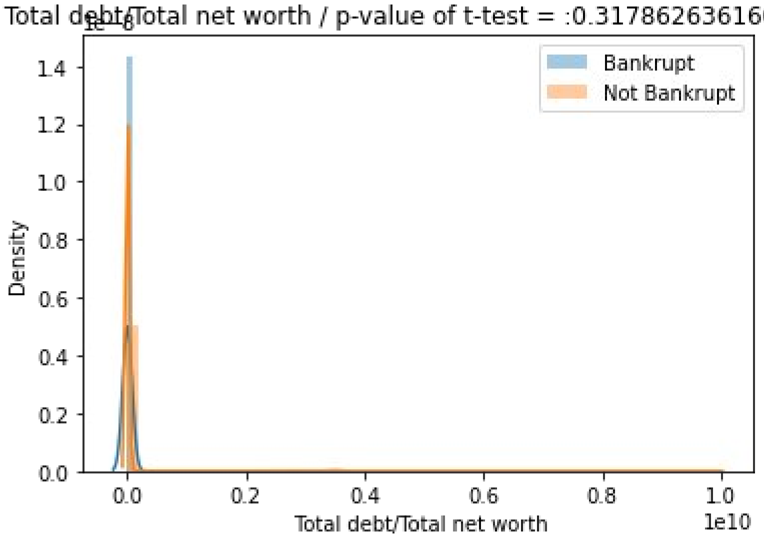




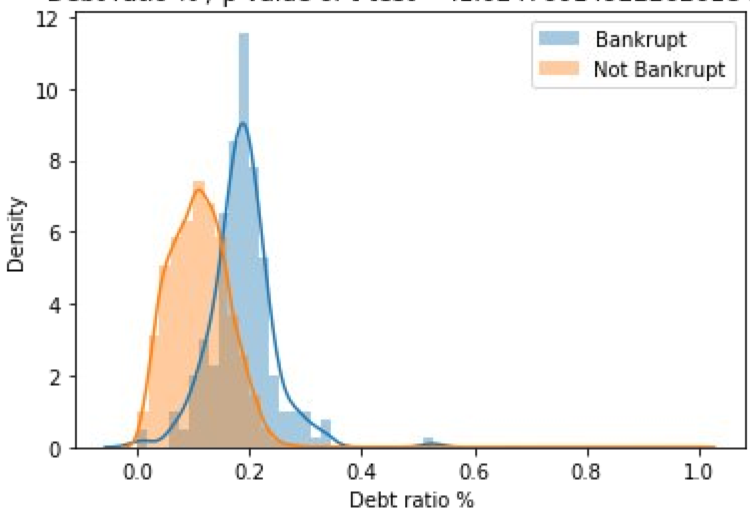
Quick Ratio / p-value of t-test = :0.3178626370940128



Total debt/Total net worth / p-value of t-test = :0.3178626361603897

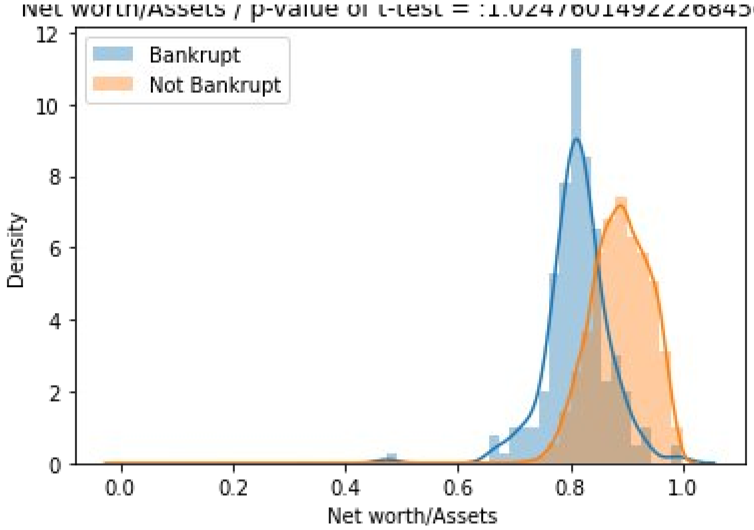


Debt ratio % / p-value of t-test = :1.0247601492226261e-50

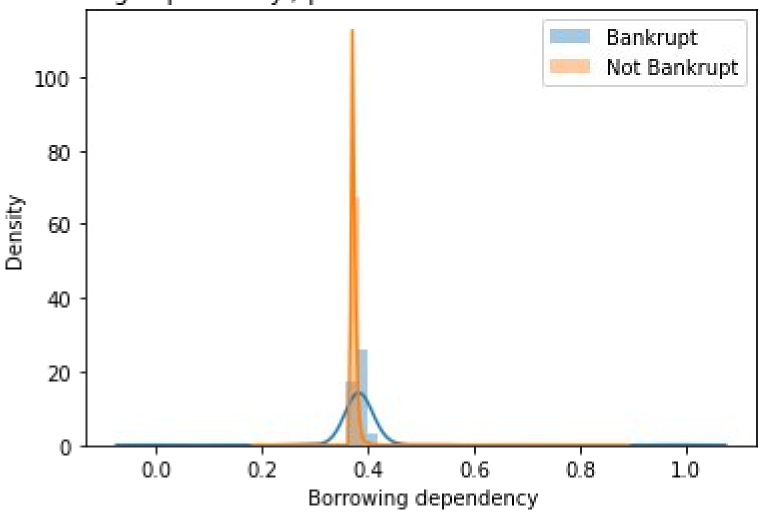


Net worth/Assets / p-value of t-test = :1.0247601492226845e-50

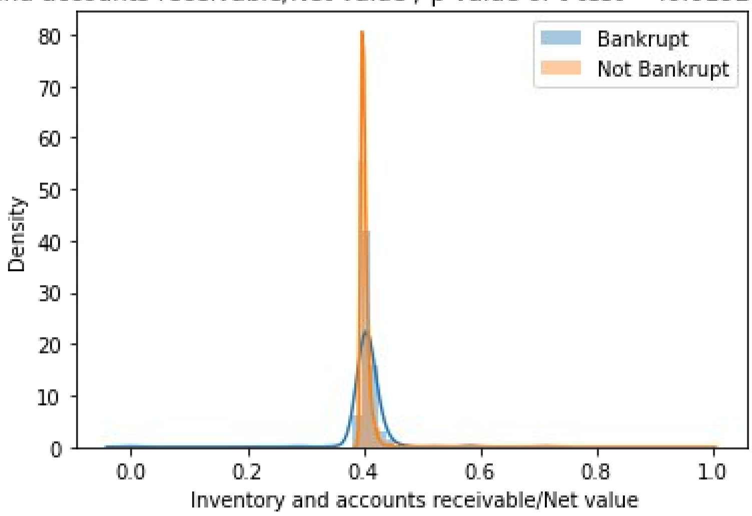
Net worth/Assets / p-value of t-test = :1.0247601492226845e-50



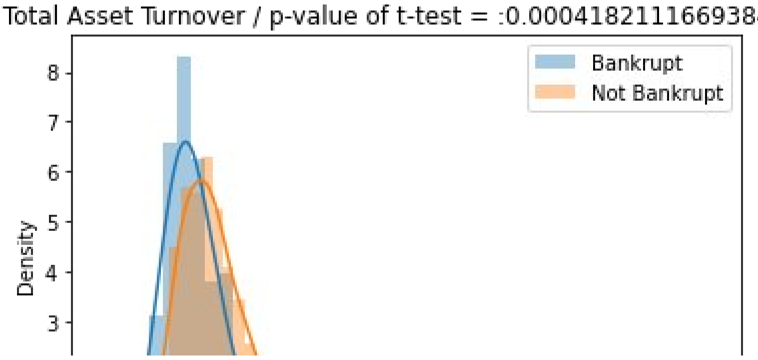
Borrowing dependency / p-value of t-test = :0.000615479123515144

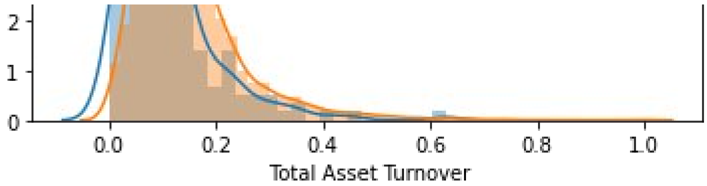


Inventory and accounts receivable/Net value / p-value of t-test = :0.019263143637869164

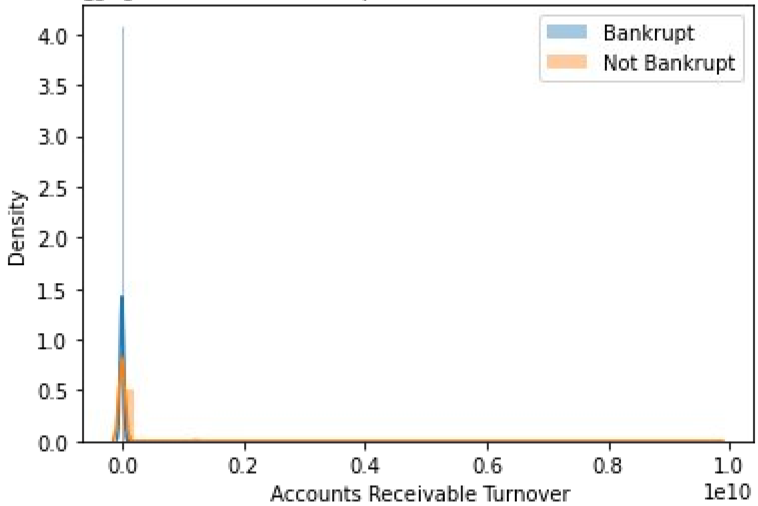


Total Asset Turnover / p-value of t-test = :0.0004182111669384241

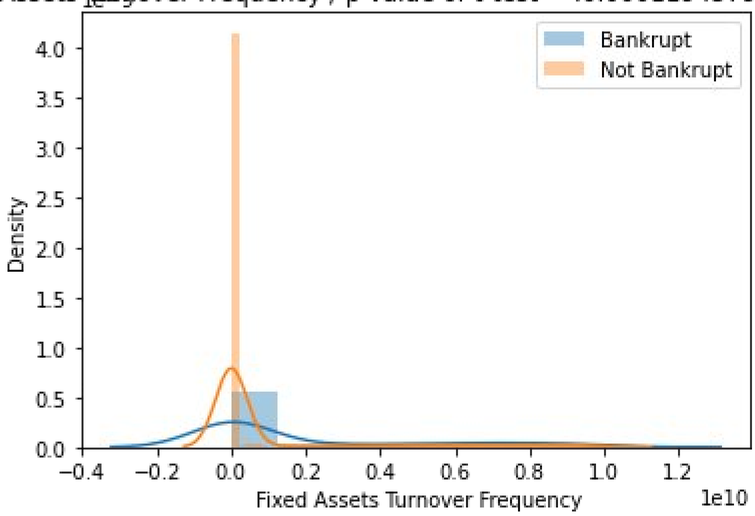




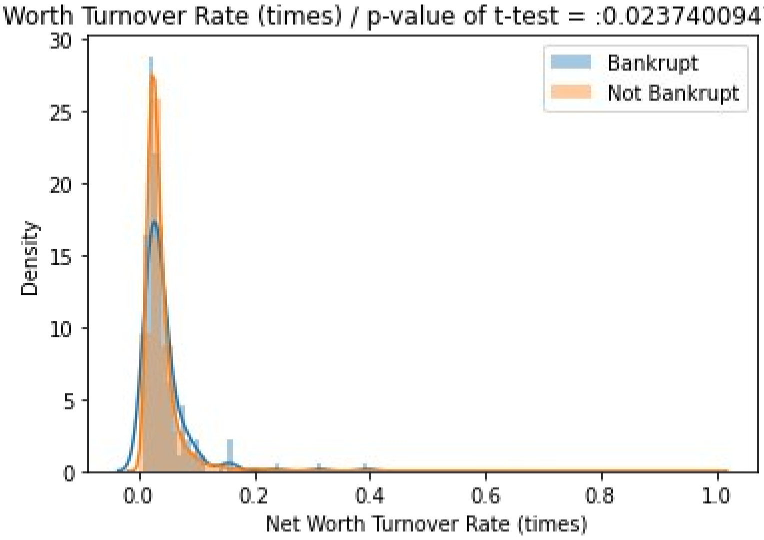
Accounts Receivable Turnover / p-value of t-test = :0.3178626368064024



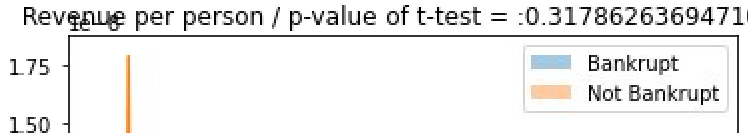
Fixed Assets Turnover Frequency / p-value of t-test = :0.00011945795866096175

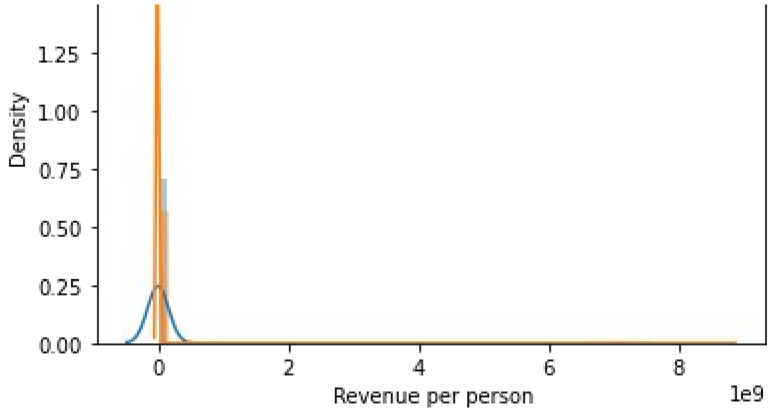


Net Worth Turnover Rate (times) / p-value of t-test = :0.023740094764209036

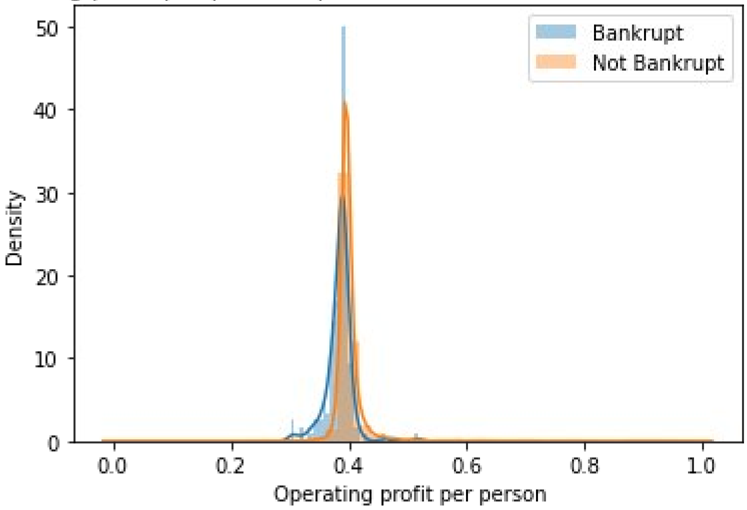


Revenue per person / p-value of t-test = :0.3178626369471087

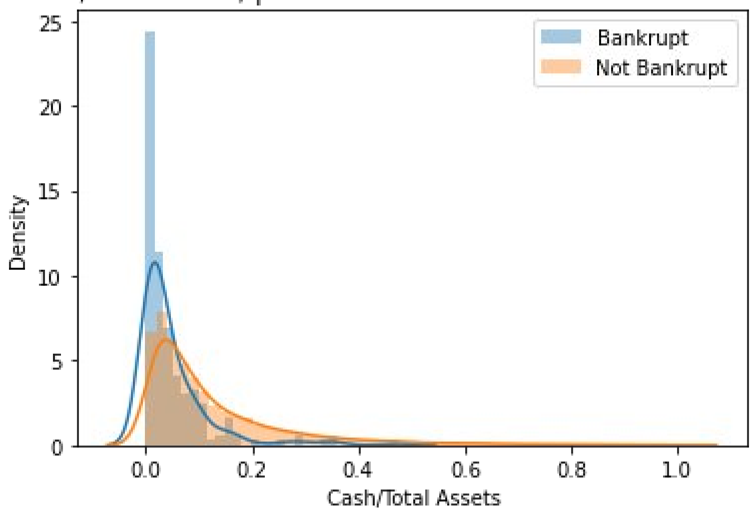




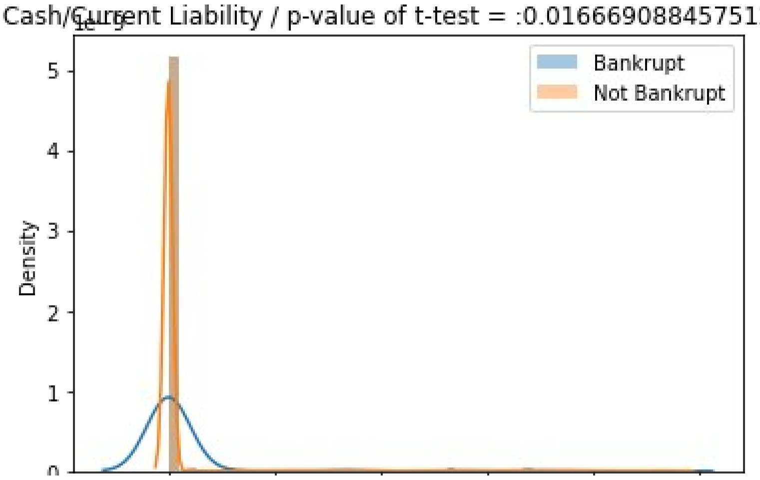
Operating profit per person / p-value of t-test = :9.548554118511362e-13



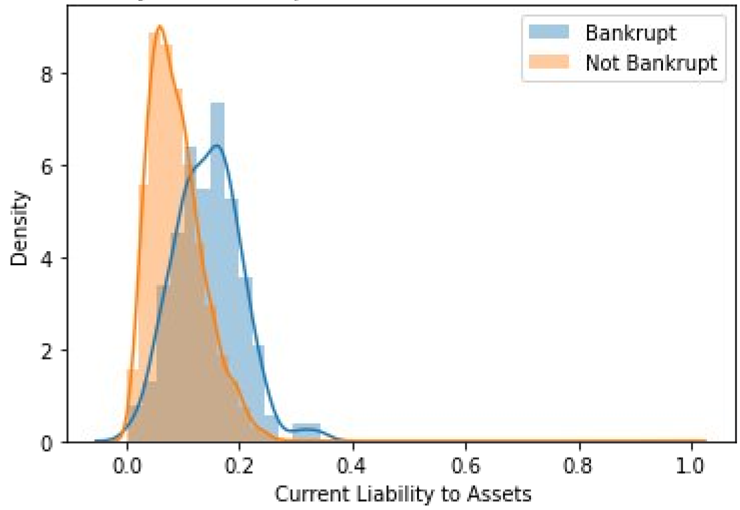
Cash/Total Assets / p-value of t-test = :1.3759217274175401e-14



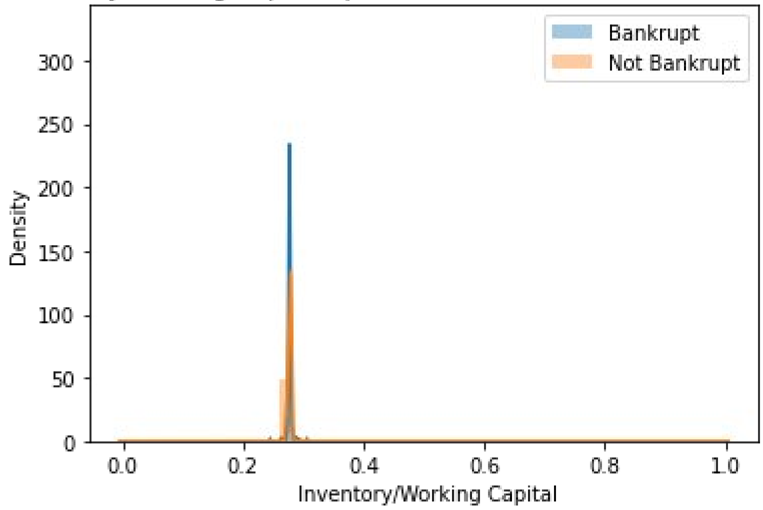
Cash/Current Liability / p-value of t-test = :0.016669088457512565



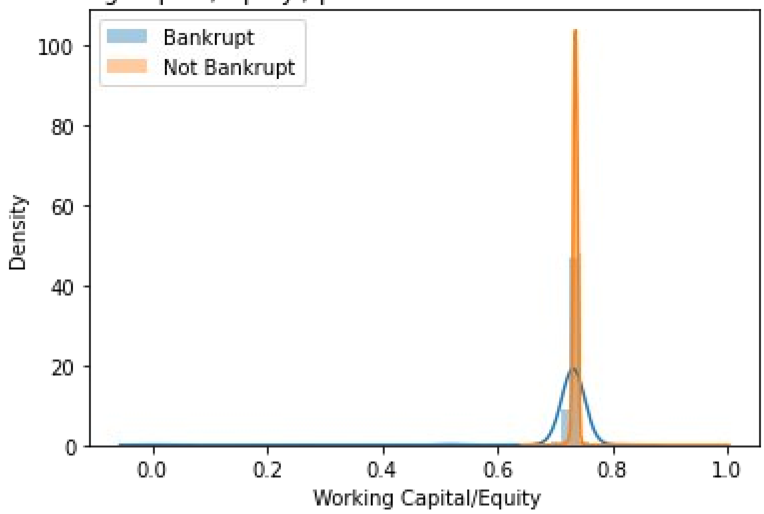
Current Liability to Assets / p-value of t-test = :1.8510860226217706e-33



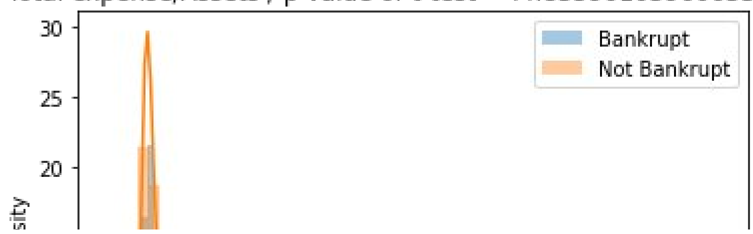
Inventory/Working Capital / p-value of t-test = :0.7986781908375753

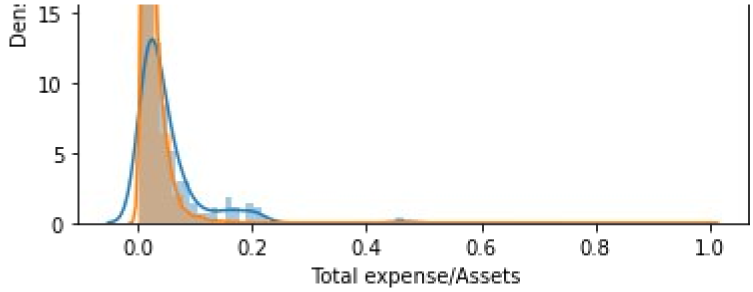


Working Capital/Equity / p-value of t-test = :0.00993529531703589

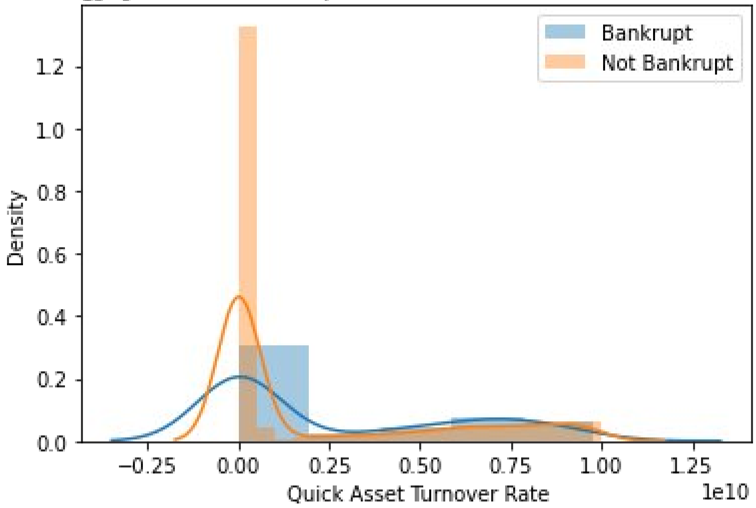


Total expense/Assets / p-value of t-test = :4.835901039606331e-08

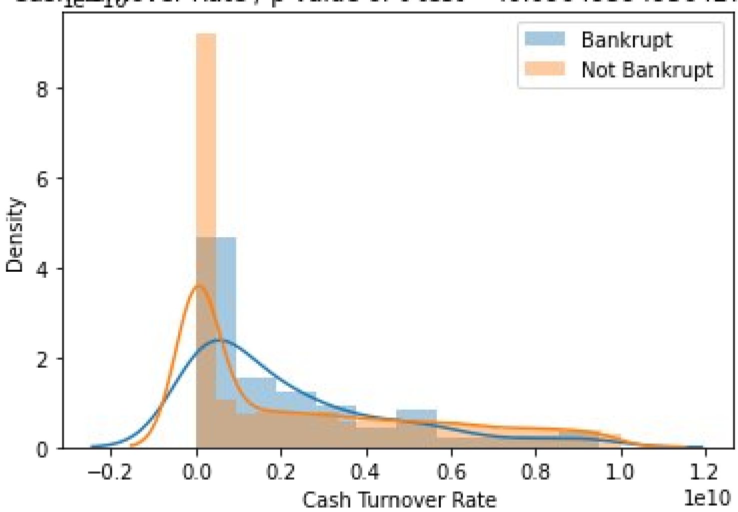




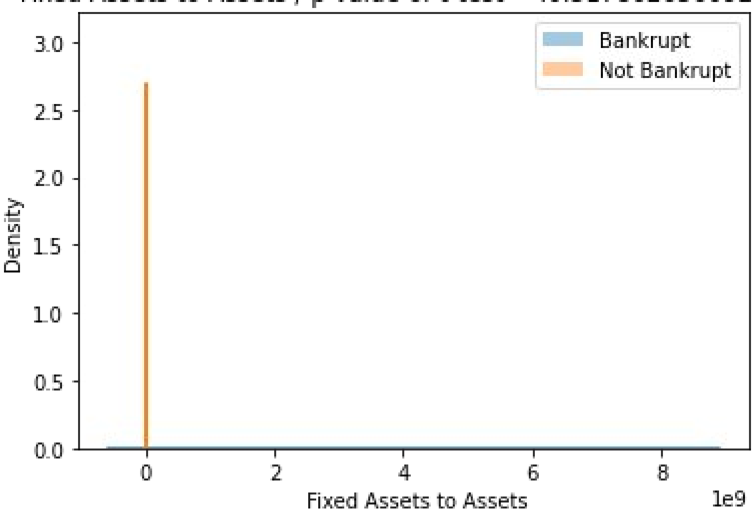
Quick Asset Turnover Rate / p-value of t-test = :0.010266065172380429



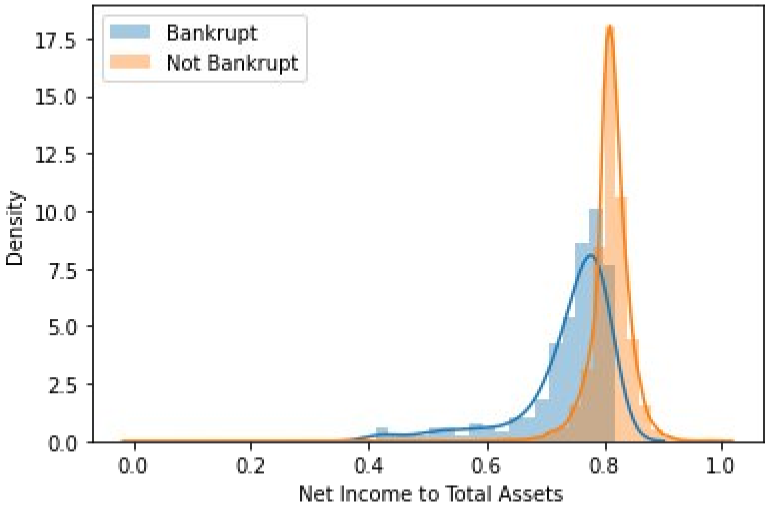
Cash Turnover Rate / p-value of t-test = :0.09049594936427773



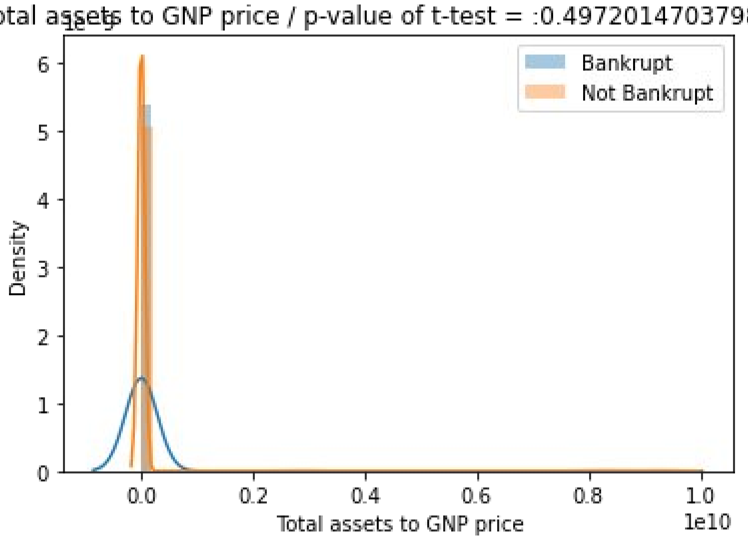
Fixed Assets to Assets / p-value of t-test = :0.3178626360923286



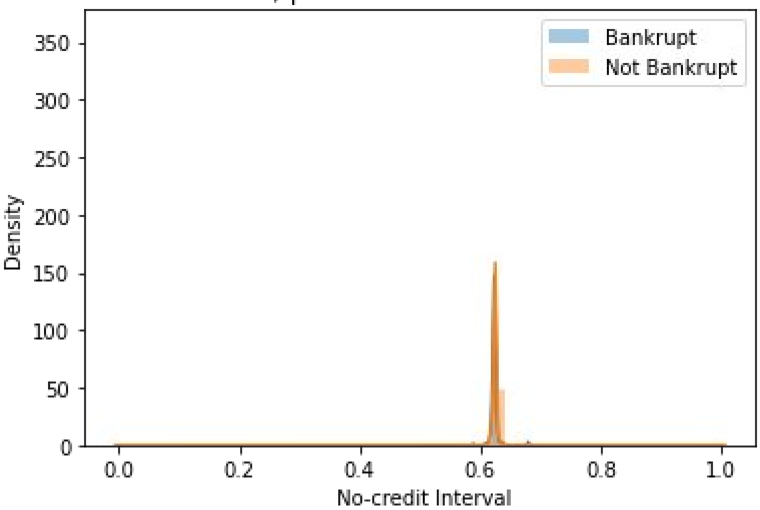
Net Income to Total Assets / p-value of t-test = :7.054312994701923e-30



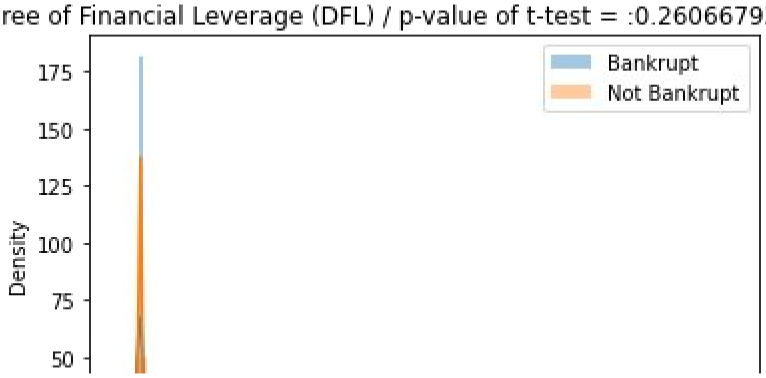
Total assets to GNP price / p-value of t-test = :0.49720147037982343

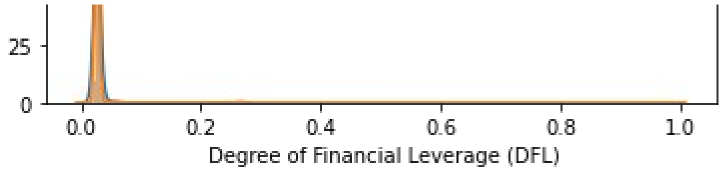


No-credit Interval / p-value of t-test = :0.1144804951610577



Degree of Financial Leverage (DFL) / p-value of t-test = :0.2606679399394551





#

Metrics

In [27]:

```
def get_clf_eval(y_test,pred=None,pred_proba=None):
    confusion = confusion_matrix(y_test,pred)
    accuracy = accuracy_score(y_test,pred)
    precision = precision_score(y_test,pred)
    recall = recall_score(y_test,pred)
    roc_auc = roc_auc_score(y_test,pred_proba)
    f1 = f1_score(y_test,pred)
    f2 = fbeta_score(y_test,pred,beta=2)
    balanced_acc = balanced_accuracy_score(y_test,pred)
    G_Mean = geometric_mean_score(y_test,pred)
    MCC = matthews_corrcoef(y_test,pred)
    print(confusion)
    print('정확도:',accuracy.round(3),'정밀도:', precision.round(3),'재현율:',recall.rour
```

Baseline Model은 #1번 노트에 표시

Resampling 1.) SMOTE

In [28]:

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

SMOTE GBM 0.7~1.0

In [29]:

```
smote1 = SMOTE(sampling_strategy=0.7)
X_train_smote1,y_train_smote1 = smote1.fit_resample(X_train,y_train)

gbm_smote_1 = GradientBoostingClassifier(random_state=2021)
gbm_smote_1.fit(X_train_smote1,y_train_smote1)
pred = gbm_smote_1.predict(X_test)
pred_proba = gbm_smote_1.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote2 = SMOTE(sampling_strategy=0.75)
X_train_smote2,y_train_smote2 = smote2.fit_resample(X_train,y_train)

gbm_smote_2 = GradientBoostingClassifier(random_state=2021)
gbm_smote_2.fit(X_train_smote2,y_train_smote2)
pred = gbm_smote_2.predict(X_test)
pred_proba = gbm_smote_2.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote3 = SMOTE(sampling_strategy=0.8)
X_train_smote3,y_train_smote3 = smote3.fit_resample(X_train,y_train)

gbm_smote_3 = GradientBoostingClassifier(random_state=2021)
gbm_smote_3.fit(X_train_smote3,y_train_smote3)
```

```python
pred = gbm_smote_3.predict(X_test)
pred_proba = gbm_smote_3.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote4 = SMOTE(sampling_strategy=0.85)
X_train_smote4,y_train_smote4 = smote4.fit_resample(X_train,y_train)

gbm_smote_4 = GradientBoostingClassifier(random_state=2021)
gbm_smote_4.fit(X_train_smote4,y_train_smote4)
pred = gbm_smote_4.predict(X_test)
pred_proba = gbm_smote_4.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote5 = SMOTE(sampling_strategy=0.9)
X_train_smote5,y_train_smote5 = smote5.fit_resample(X_train,y_train)

gbm_smote_5 = GradientBoostingClassifier(random_state=2021)
gbm_smote_5.fit(X_train_smote5,y_train_smote5)
pred = gbm_smote_5.predict(X_test)
pred_proba = gbm_smote_5.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote6 = SMOTE(sampling_strategy=0.95)
X_train_smote6,y_train_smote6 = smote6.fit_resample(X_train,y_train)

gbm_smote_6 = GradientBoostingClassifier(random_state=2021)
gbm_smote_6.fit(X_train_smote6,y_train_smote6)
pred = gbm_smote_6.predict(X_test)
pred_proba = gbm_smote_6.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote7 = SMOTE()
X_train_smote7,y_train_smote7 = smote7.fit_resample(X_train,y_train)

gbm_smote_7 = GradientBoostingClassifier(random_state=2021)
gbm_smote_7.fit(X_train_smote7,y_train_smote7)
pred = gbm_smote_7.predict(X_test)
pred_proba = gbm_smote_7.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
```

```
[[1860  120]
 [  21   45]]
정확도: 0.931 정밀도: 0.273 재현율: 0.682 AUC: 0.925 F1: 0.39 F2: 0.524 Balanced_Accurac
y: 0.811 G-Mean: 0.8
 matthews_corrcoef: 0.403


[[1853  127]
 [  17   49]]
정확도: 0.93 정밀도: 0.278 재현율: 0.742 AUC: 0.922 F1: 0.405 F2: 0.557 Balanced_Accurac
y: 0.839 G-Mean: 0.834
 matthews_corrcoef: 0.427


[[1852  128]
 [  20   46]]
정확도: 0.928 정밀도: 0.264 재현율: 0.697 AUC: 0.928 F1: 0.383 F2: 0.525 Balanced_Accurac
y: 0.816 G-Mean: 0.807
 matthews_corrcoef: 0.401
```

```
[[1852  128]
 [  21   45]]
정확도: 0.927 정밀도: 0.26 재현율: 0.682 AUC: 0.925 F1: 0.377 F2: 0.515 Balanced_Accurac
y: 0.809 G-Mean: 0.799
 matthews_corrcoef: 0.392


[[1842  138]
 [  21   45]]
정확도: 0.922 정밀도: 0.246 재현율: 0.682 AUC: 0.923 F1: 0.361 F2: 0.503 Balanced_Accurac
y: 0.806 G-Mean: 0.796
 matthews_corrcoef: 0.379


[[1841  139]
 [  18   48]]
정확도: 0.923 정밀도: 0.257 재현율: 0.727 AUC: 0.926 F1: 0.379 F2: 0.532 Balanced_Accurac
y: 0.829 G-Mean: 0.822
 matthews_corrcoef: 0.403


[[1828  152]
 [  21   45]]
정확도: 0.915 정밀도: 0.228 재현율: 0.682 AUC: 0.92 F1: 0.342 F2: 0.488 Balanced_Accurac
y: 0.803 G-Mean: 0.793
 matthews_corrcoef: 0.362
```

SMOTE LGBM 0.7~1.0

In [30]:

```
smote1 = SMOTE(sampling_strategy=0.7)
X_train_smote1,y_train_smote1 = smote1.fit_resample(X_train,y_train)

lgbm_smote_1 = LGBMClassifier(random_state=2021)
lgbm_smote_1.fit(X_train_smote1,y_train_smote1)
pred = lgbm_smote_1.predict(X_test)
pred_proba = lgbm_smote_1.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote2 = SMOTE(sampling_strategy=0.75)
X_train_smote2,y_train_smote2 = smote2.fit_resample(X_train,y_train)

lgbm_smote_2 = LGBMClassifier(random_state=2021)
lgbm_smote_2.fit(X_train_smote2,y_train_smote2)
pred = lgbm_smote_2.predict(X_test)
pred_proba = lgbm_smote_2.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote3 = SMOTE(sampling_strategy=0.8)
X_train_smote3,y_train_smote3 = smote3.fit_resample(X_train,y_train)

lgbm_smote_3 = LGBMClassifier(random_state=2021)
lgbm_smote_3.fit(X_train_smote3,y_train_smote3)
pred = lgbm_smote_3.predict(X_test)
pred_proba = lgbm_smote_3.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote4 = SMOTE(sampling_strategy=0.85)
X_train_smote4,y_train_smote4 = smote4.fit_resample(X_train,y_train)

lgbm_smote_4 = LGBMClassifier(random_state=2021)
```

```
lgbm_smote_4.fit(X_train_smote4,y_train_smote4)
pred = lgbm_smote_4.predict(X_test)
pred_proba = lgbm_smote_4.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote5 = SMOTE(sampling_strategy=0.9)
X_train_smote5,y_train_smote5 = smote5.fit_resample(X_train,y_train)

lgbm_smote_5 = LGBMClassifier(random_state=2021)
lgbm_smote_5.fit(X_train_smote5,y_train_smote5)
pred = lgbm_smote_5.predict(X_test)
pred_proba = lgbm_smote_5.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote6 = SMOTE(sampling_strategy=0.95)
X_train_smote6,y_train_smote6 = smote6.fit_resample(X_train,y_train)

lgbm_smote_6 = LGBMClassifier(random_state=2021)
lgbm_smote_6.fit(X_train_smote6,y_train_smote6)
pred = lgbm_smote_6.predict(X_test)
pred_proba = lgbm_smote_6.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote7 = SMOTE()
X_train_smote7,y_train_smote7 = smote7.fit_resample(X_train,y_train)

lgbm_smote_7 = LGBMClassifier(random_state=2021)
lgbm_smote_7.fit(X_train_smote7,y_train_smote7)
pred = lgbm_smote_7.predict(X_test)
pred_proba = lgbm_smote_7.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
```

```
[[1922   58]
 [  27   39]]
정확도: 0.958 정밀도: 0.402 재현율: 0.591 AUC: 0.927 F1: 0.479 F2: 0.54 Balanced_Accurac
y: 0.781 G-Mean: 0.757
 matthews_corrcoef: 0.467


[[1923   57]
 [  29   37]]
정확도: 0.958 정밀도: 0.394 재현율: 0.561 AUC: 0.938 F1: 0.462 F2: 0.517 Balanced_Accurac
y: 0.766 G-Mean: 0.738
 matthews_corrcoef: 0.449


[[1918   62]
 [  25   41]]
정확도: 0.957 정밀도: 0.398 재현율: 0.621 AUC: 0.933 F1: 0.485 F2: 0.559 Balanced_Accurac
y: 0.795 G-Mean: 0.776
 matthews_corrcoef: 0.477


[[1918   62]
 [  26   40]]
정확도: 0.957 정밀도: 0.392 재현율: 0.606 AUC: 0.929 F1: 0.476 F2: 0.546 Balanced_Accurac
y: 0.787 G-Mean: 0.766
 matthews_corrcoef: 0.467


[[1923   57]
 [  24   42]]
```

```
정확도: 0.96 정밀도: 0.424 재현율: 0.636 AUC: 0.937 F1: 0.509 F2: 0.579 Balanced_Accurac
y: 0.804 G-Mean: 0.786
 matthews_corrcoef: 0.5


[[1918   62]
 [  24   42]]
정확도: 0.958 정밀도: 0.404 재현율: 0.636 AUC: 0.933 F1: 0.494 F2: 0.571 Balanced_Accurac
y: 0.803 G-Mean: 0.785
 matthews_corrcoef: 0.487


[[1919   61]
 [  30   36]]
정확도: 0.956 정밀도: 0.371 재현율: 0.545 AUC: 0.926 F1: 0.442 F2: 0.499 Balanced_Accurac
y: 0.757 G-Mean: 0.727
 matthews_corrcoef: 0.428
```

SMOTE XGBoost 0.7~1.0

In [31]:

```
smote1 = SMOTE(sampling_strategy=0.7)
X_train_smote1,y_train_smote1 = smote1.fit_resample(X_train,y_train)

xgb_smote_1 = XGBClassifier(random_state=2021,subsample=0.8)
xgb_smote_1.fit(X_train_smote1,y_train_smote1)
pred = xgb_smote_1.predict(X_test)
pred_proba = xgb_smote_1.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote2 = SMOTE(sampling_strategy=0.75)
X_train_smote2,y_train_smote2 = smote2.fit_resample(X_train,y_train)

xgb_smote_2 = XGBClassifier(random_state=2021,subsample=0.8)
xgb_smote_2.fit(X_train_smote2,y_train_smote2)
pred = xgb_smote_2.predict(X_test)
pred_proba = xgb_smote_2.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote3 = SMOTE(sampling_strategy=0.8)
X_train_smote3,y_train_smote3 = smote3.fit_resample(X_train,y_train)

xgb_smote_3 = XGBClassifier(random_state=2021,subsample=0.8)
xgb_smote_3.fit(X_train_smote3,y_train_smote3)
pred = xgb_smote_3.predict(X_test)
pred_proba = xgb_smote_3.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote4 = SMOTE(sampling_strategy=0.85)
X_train_smote4,y_train_smote4 = smote4.fit_resample(X_train,y_train)

xgb_smote_4 = XGBClassifier(random_state=2021,subsample=0.8)
xgb_smote_4.fit(X_train_smote4,y_train_smote4)
pred = xgb_smote_4.predict(X_test)
pred_proba = xgb_smote_4.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote5 = SMOTE(sampling_strategy=0.9)
X_train_smote5,y_train_smote5 = smote5.fit_resample(X_train,y_train)

xgb_smote_5 = XGBClassifier(random_state=2021,subsample=0.8)
```

```python
xgb_smote_5.fit(X_train_smote5,y_train_smote5)
pred = xgb_smote_5.predict(X_test)
pred_proba = xgb_smote_5.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote6 = SMOTE(sampling_strategy=0.95)
X_train_smote6,y_train_smote6 = smote6.fit_resample(X_train,y_train)

xgb_smote_6 = XGBClassifier(random_state=2021,subsample=0.8)
xgb_smote_6.fit(X_train_smote6,y_train_smote6 )
pred = xgb_smote_6.predict(X_test)
pred_proba = xgb_smote_6.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote7 = SMOTE()
X_train_smote7,y_train_smote7 = smote7.fit_resample(X_train,y_train)

xgb_smote_7 = XGBClassifier(random_state=2021,subsample=0.8)
xgb_smote_7.fit(X_train_smote7,y_train_smote7 )
pred = xgb_smote_7.predict(X_test)
pred_proba = xgb_smote_7.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
```

```
[17:18:24] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1917   63]
 [  24   42]]
정확도: 0.957 정밀도: 0.4 재현율: 0.636 AUC: 0.936 F1: 0.491 F2: 0.569 Balanced_Accuracy:
0.802 G-Mean: 0.785
 matthews_corrcoef: 0.484


[17:18:27] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1923   57]
 [  30   36]]
정확도: 0.957 정밀도: 0.387 재현율: 0.545 AUC: 0.93 F1: 0.453 F2: 0.504 Balanced_Accurac
y: 0.758 G-Mean: 0.728
 matthews_corrcoef: 0.438


[17:18:29] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1916   64]
 [  31   35]]
정확도: 0.954 정밀도: 0.354 재현율: 0.53 AUC: 0.917 F1: 0.424 F2: 0.482 Balanced_Accurac
y: 0.749 G-Mean: 0.716
 matthews_corrcoef: 0.41


[17:18:31] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1924   56]
 [  31   35]]
정확도: 0.957 정밀도: 0.385 재현율: 0.53 AUC: 0.927 F1: 0.446 F2: 0.493 Balanced_Accurac
```

```
y: 0.751 G-Mean: 0.718
 matthews_corrcoef: 0.43


[17:18:34] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1923   57]
 [  29   37]]
정확도: 0.958 정밀도: 0.394 재현율: 0.561 AUC: 0.92 F1: 0.462 F2: 0.517 Balanced_Accurac
y: 0.766 G-Mean: 0.738
 matthews_corrcoef: 0.449


[17:18:37] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1917   63]
 [  27   39]]
정확도: 0.956 정밀도: 0.382 재현율: 0.591 AUC: 0.935 F1: 0.464 F2: 0.533 Balanced_Accurac
y: 0.78 G-Mean: 0.756
 matthews_corrcoef: 0.454


[17:18:40] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1924   56]
 [  24   42]]
정확도: 0.961 정밀도: 0.429 재현율: 0.636 AUC: 0.933 F1: 0.512 F2: 0.58 Balanced_Accurac
y: 0.804 G-Mean: 0.786
 matthews_corrcoef: 0.503
```

Resampling 2.) ROSE

In [32]:

```python
from imblearn.over_sampling import RandomOverSampler
```

ROSE GBM

In [33]:

```python
rose1 = RandomOverSampler(sampling_strategy=0.7)
X_train_rose1,y_train_rose1 = rose1.fit_resample(X_train,y_train)

gbm_rose_1 = GradientBoostingClassifier(random_state=2021)
gbm_rose_1.fit(X_train_rose1,y_train_rose1)
pred = gbm_rose_1.predict(X_test)
pred_proba = gbm_rose_1.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

rose2 = RandomOverSampler(sampling_strategy=0.75)
X_train_rose2,y_train_rose2 = rose2.fit_resample(X_train,y_train)

gbm_rose_2 = GradientBoostingClassifier(random_state=2021)
gbm_rose_2.fit(X_train_rose2,y_train_rose2)
pred = gbm_rose_2.predict(X_test)
pred_proba = gbm_rose_2.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

rose3 = RandomOverSampler(sampling_strategy=0.8)
```

```
X_train_rose3,y_train_rose3 = rose3.fit_resample(X_train,y_train)

gbm_rose_3 = GradientBoostingClassifier(random_state=2021)
gbm_rose_3.fit(X_train_rose3,y_train_rose3)
pred = gbm_rose_3.predict(X_test)
pred_proba = gbm_rose_3.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

rose4 = RandomOverSampler(sampling_strategy=0.85)
X_train_rose4,y_train_rose4 = rose4.fit_resample(X_train,y_train)

gbm_rose_4 = GradientBoostingClassifier(random_state=2021)
gbm_rose_4.fit(X_train_rose4,y_train_rose4)
pred = gbm_rose_4.predict(X_test)
pred_proba = gbm_rose_4.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

rose5 = RandomOverSampler(sampling_strategy=0.9)
X_train_rose5,y_train_rose5 = rose5.fit_resample(X_train,y_train)

gbm_rose_5 = GradientBoostingClassifier(random_state=2021)
gbm_rose_5.fit(X_train_rose5,y_train_rose5)
pred = gbm_rose_5.predict(X_test)
pred_proba = gbm_rose_5.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

rose6 = RandomOverSampler(sampling_strategy=0.95)
X_train_rose6,y_train_rose6 = rose6.fit_resample(X_train,y_train)

gbm_rose_6 = GradientBoostingClassifier(random_state=2021)
gbm_rose_6.fit(X_train_rose6,y_train_rose6)
pred = gbm_rose_6.predict(X_test)
pred_proba = gbm_rose_6.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

rose7 = RandomOverSampler()
X_train_rose7,y_train_rose7 = rose7.fit_resample(X_train,y_train)

gbm_rose_7 = GradientBoostingClassifier(random_state=2021)
gbm_rose_7.fit(X_train_rose7,y_train_rose7)
pred = gbm_rose_7.predict(X_test)
pred_proba = gbm_rose_7.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
```

```
[[1875  105]
 [  21   45]]
정확도: 0.938 정밀도: 0.3 재현율: 0.682 AUC: 0.932 F1: 0.417 F2: 0.543 Balanced_Accuracy:
0.814 G-Mean: 0.804
 matthews_corrcoef: 0.426



[[1869  111]
 [  19   47]]
정확도: 0.936 정밀도: 0.297 재현율: 0.712 AUC: 0.929 F1: 0.42 F2: 0.557 Balanced_Accurac
y: 0.828 G-Mean: 0.82
 matthews_corrcoef: 0.434



[[1877  103]
 [  21   45]]
```

```
정확도: 0.939 정밀도: 0.304 재현율: 0.682 AUC: 0.928 F1: 0.421 F2: 0.546 Balanced_Accurac
y: 0.815 G-Mean: 0.804
 matthews_corrcoef: 0.43


[[1866  114]
 [  18   48]]
정확도: 0.935 정밀도: 0.296 재현율: 0.727 AUC: 0.932 F1: 0.421 F2: 0.563 Balanced_Accurac
y: 0.835 G-Mean: 0.828
 matthews_corrcoef: 0.438


[[1856  124]
 [  19   47]]
정확도: 0.93 정밀도: 0.275 재현율: 0.712 AUC: 0.928 F1: 0.397 F2: 0.54 Balanced_Accuracy:
0.825 G-Mean: 0.817
 matthews_corrcoef: 0.415


[[1854  126]
 [  17   49]]
정확도: 0.93 정밀도: 0.28 재현율: 0.742 AUC: 0.927 F1: 0.407 F2: 0.558 Balanced_Accuracy:
0.839 G-Mean: 0.834
 matthews_corrcoef: 0.429


[[1861  119]
 [  18   48]]
정확도: 0.933 정밀도: 0.287 재현율: 0.727 AUC: 0.93 F1: 0.412 F2: 0.557 Balanced_Accurac
y: 0.834 G-Mean: 0.827
 matthews_corrcoef: 0.431
```

ROSE LGBM

In [34]:

```
rose1 = RandomOverSampler(sampling_strategy=0.7)
X_train_rose1,y_train_rose1 = rose1.fit_resample(X_train,y_train)

lgbm_rose_1 = LGBMClassifier(random_state=2021)
lgbm_rose_1.fit(X_train_rose1,y_train_rose1)
pred = lgbm_rose_1.predict(X_test)
pred_proba = lgbm_rose_1.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

rose2 = RandomOverSampler(sampling_strategy=0.75)
X_train_rose2,y_train_rose2 = rose2.fit_resample(X_train,y_train)

lgbm_rose_2 = LGBMClassifier(random_state=2021)
lgbm_rose_2.fit(X_train_rose2,y_train_rose2)
pred = lgbm_rose_2.predict(X_test)
pred_proba = lgbm_rose_2.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

rose3 = RandomOverSampler(sampling_strategy=0.8)
X_train_rose3,y_train_rose3 = rose3.fit_resample(X_train,y_train)

lgbm_rose_3 = LGBMClassifier(random_state=2021)
lgbm_rose_3.fit(X_train_rose3,y_train_rose3)
pred = lgbm_rose_3.predict(X_test)
pred_proba = lgbm_rose_3.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

rose4 = RandomOverSampler(sampling_strategy=0.85)
```

```python
X_train_rose4,y_train_rose4 = rose4.fit_resample(X_train,y_train)

lgbm_rose_4 = LGBMClassifier(random_state=2021)
lgbm_rose_4.fit(X_train_rose4,y_train_rose4)
pred = lgbm_rose_4.predict(X_test)
pred_proba = lgbm_rose_4.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

rose5 = RandomOverSampler(sampling_strategy=0.9)
X_train_rose5,y_train_rose5 = rose5.fit_resample(X_train,y_train)

lgbm_rose_5 = LGBMClassifier(random_state=2021)
lgbm_rose_5.fit(X_train_rose5,y_train_rose5)
pred = lgbm_rose_5.predict(X_test)
pred_proba = lgbm_rose_5.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

rose6 = RandomOverSampler(sampling_strategy=0.95)
X_train_rose6,y_train_rose6 = rose6.fit_resample(X_train,y_train)

lgbm_rose_6 = LGBMClassifier(random_state=2021)
lgbm_rose_6.fit(X_train_rose6,y_train_rose6)
pred = lgbm_rose_6.predict(X_test)
pred_proba = lgbm_rose_6.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

rose7 = RandomOverSampler()
X_train_rose7,y_train_rose7 = rose7.fit_resample(X_train,y_train)

lgbm_rose_7 = LGBMClassifier(random_state=2021)
lgbm_rose_7.fit(X_train_rose7,y_train_rose7)
pred = lgbm_rose_7.predict(X_test)
pred_proba = lgbm_rose_7.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
```

```
[[1952   28]
 [  42   24]]
정확도: 0.966 정밀도: 0.462 재현율: 0.364 AUC: 0.927 F1: 0.407 F2: 0.38 Balanced_Accurac
y: 0.675 G-Mean: 0.599
 matthews_corrcoef: 0.392


[[1949   31]
 [  39   27]]
정확도: 0.966 정밀도: 0.466 재현율: 0.409 AUC: 0.922 F1: 0.435 F2: 0.419 Balanced_Accurac
y: 0.697 G-Mean: 0.635
 matthews_corrcoef: 0.419


[[1952   28]
 [  40   26]]
정확도: 0.967 정밀도: 0.481 재현율: 0.394 AUC: 0.926 F1: 0.433 F2: 0.409 Balanced_Accurac
y: 0.69 G-Mean: 0.623
 matthews_corrcoef: 0.419


[[1953   27]
 [  38   28]]
정확도: 0.968 정밀도: 0.509 재현율: 0.424 AUC: 0.93 F1: 0.463 F2: 0.439 Balanced_Accurac
y: 0.705 G-Mean: 0.647
 matthews_corrcoef: 0.449
```

```
[[1947   33]
 [  39   27]]
정확도: 0.965 정밀도: 0.45 재현율: 0.409 AUC: 0.928 F1: 0.429 F2: 0.417 Balanced_Accurac
y: 0.696 G-Mean: 0.634
 matthews_corrcoef: 0.411


[[1947   33]
 [  39   27]]
정확도: 0.965 정밀도: 0.45 재현율: 0.409 AUC: 0.929 F1: 0.429 F2: 0.417 Balanced_Accurac
y: 0.696 G-Mean: 0.634
 matthews_corrcoef: 0.411


[[1949   31]
 [  37   29]]
정확도: 0.967 정밀도: 0.483 재현율: 0.439 AUC: 0.934 F1: 0.46 F2: 0.448 Balanced_Accurac
y: 0.712 G-Mean: 0.658
 matthews_corrcoef: 0.444
```

ROSE XGB

In [35]:

```
rose1 = RandomOverSampler(sampling_strategy=0.7)
X_train_rose1,y_train_rose1 = rose1.fit_resample(X_train,y_train)

xgb_rose_1 = XGBClassifier(random_state=2021)
xgb_rose_1.fit(X_train_rose1,y_train_rose1)
pred = xgb_rose_1.predict(X_test)
pred_proba = xgb_rose_1.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

rose2 = RandomOverSampler(sampling_strategy=0.75)
X_train_rose2,y_train_rose2 = rose2.fit_resample(X_train,y_train)

xgb_rose_2 = XGBClassifier(random_state=2021)
xgb_rose_2.fit(X_train_rose2,y_train_rose2)
pred = xgb_rose_2.predict(X_test)
pred_proba = xgb_rose_2.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

rose3 = RandomOverSampler(sampling_strategy=0.8)
X_train_rose3,y_train_rose3 = rose3.fit_resample(X_train,y_train)

xgb_rose_3 = XGBClassifier(random_state=2021)
xgb_rose_3.fit(X_train_rose3,y_train_rose3)
pred = xgb_rose_3.predict(X_test)
pred_proba = xgb_rose_3.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

rose4 = RandomOverSampler(sampling_strategy=0.85)
X_train_rose4,y_train_rose4 = rose4.fit_resample(X_train,y_train)

xgb_rose_4 = XGBClassifier(random_state=2021)
xgb_rose_4.fit(X_train_rose4,y_train_rose4)
pred = xgb_rose_4.predict(X_test)
pred_proba = xgb_rose_4.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

rose5 = RandomOverSampler(sampling_strategy=0.9)
```

```
X_train_rose5,y_train_rose5 = rose5.fit_resample(X_train,y_train)

xgb_rose_5 = XGBClassifier(random_state=2021)
xgb_rose_5.fit(X_train_rose5,y_train_rose5)
pred = xgb_rose_5.predict(X_test)
pred_proba = xgb_rose_5.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

rose6 = RandomOverSampler(sampling_strategy=0.95)
X_train_rose6,y_train_rose6 = rose6.fit_resample(X_train,y_train)

xgb_rose_6 = XGBClassifier(random_state=2021)
xgb_rose_6.fit(X_train_rose6,y_train_rose6)
pred = xgb_rose_6.predict(X_test)
pred_proba = xgb_rose_6.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

rose7 = RandomOverSampler()
X_train_rose7,y_train_rose7 = rose7.fit_resample(X_train,y_train)

xgb_rose_7 = XGBClassifier(random_state=2021)
xgb_rose_7.fit(X_train_rose7,y_train_rose7)
pred = xgb_rose_7.predict(X_test)
pred_proba = xgb_rose_7.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
```

```
[17:19:41] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1951   29]
 [  42   24]]
정확도: 0.965 정밀도: 0.453 재현율: 0.364 AUC: 0.918 F1: 0.403 F2: 0.379 Balanced_Accurac
y: 0.674 G-Mean: 0.599
 matthews_corrcoef: 0.388


[17:19:43] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1953   27]
 [  42   24]]
정확도: 0.966 정밀도: 0.471 재현율: 0.364 AUC: 0.923 F1: 0.41 F2: 0.381 Balanced_Accurac
y: 0.675 G-Mean: 0.599
 matthews_corrcoef: 0.397


[17:19:44] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1950   30]
 [  44   22]]
정확도: 0.964 정밀도: 0.423 재현율: 0.333 AUC: 0.918 F1: 0.373 F2: 0.348 Balanced_Accurac
y: 0.659 G-Mean: 0.573
 matthews_corrcoef: 0.357


[17:19:46] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
```

```
[[1951   29]
 [  40   26]]
정확도: 0.966 정밀도: 0.473 재현율: 0.394 AUC: 0.907 F1: 0.43 F2: 0.408 Balanced_Accurac
y: 0.69 G-Mean: 0.623
 matthews_corrcoef: 0.414


[17:19:48] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1954   26]
 [  40   26]]
정확도: 0.968 정밀도: 0.5 재현율: 0.394 AUC: 0.914 F1: 0.441 F2: 0.411 Balanced_Accuracy:
0.69 G-Mean: 0.624
 matthews_corrcoef: 0.428


[17:19:50] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1945   35]
 [  44   22]]
정확도: 0.961 정밀도: 0.386 재현율: 0.333 AUC: 0.912 F1: 0.358 F2: 0.343 Balanced_Accurac
y: 0.658 G-Mean: 0.572
 matthews_corrcoef: 0.339


[17:19:51] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1951   29]
 [  42   24]]
정확도: 0.965 정밀도: 0.453 재현율: 0.364 AUC: 0.922 F1: 0.403 F2: 0.379 Balanced_Accurac
y: 0.674 G-Mean: 0.599
 matthews_corrcoef: 0.388
```

Resampling 3.) ADASYN

In [36]:

```
from imblearn.over_sampling import ADASYN
```

ADASYN GBM

In [37]:

```
ads1 = ADASYN(sampling_strategy=0.7)
X_train_ads1,y_train_ads1 = ads1.fit_resample(X_train,y_train)

gbm_ads1 = GradientBoostingClassifier(random_state=2021)
gbm_ads1.fit(X_train_ads1,y_train_ads1)
pred = gbm_ads1.predict(X_test)
pred_proba = gbm_ads1.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

ads2 = ADASYN(sampling_strategy=0.75)
X_train_ads2,y_train_ads2 = ads2.fit_resample(X_train,y_train)

gbm_ads2 = GradientBoostingClassifier(random_state=2021)
gbm_ads2.fit(X_train_ads2,y_train_ads2)
pred = gbm_ads2.predict(X_test)
pred_proba = gbm_ads2.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
```

```python
print('\n')

ads3 = ADASYN(sampling_strategy=0.8)
X_train_ads3,y_train_ads3 = ads3.fit_resample(X_train,y_train)

gbm_ads3 = GradientBoostingClassifier(random_state=2021)
gbm_ads3.fit(X_train_ads3,y_train_ads3)
pred = gbm_ads3.predict(X_test)
pred_proba = gbm_ads3.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

ads4 = ADASYN(sampling_strategy=0.85)
X_train_ads4,y_train_ads4 = ads4.fit_resample(X_train,y_train)

gbm_ads4 = GradientBoostingClassifier(random_state=2021)
gbm_ads4.fit(X_train_ads4,y_train_ads4)
pred = gbm_ads4.predict(X_test)
pred_proba = gbm_ads4.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

ads5 = ADASYN(sampling_strategy=0.9)
X_train_ads5,y_train_ads5 = ads5.fit_resample(X_train,y_train)

gbm_ads5 = GradientBoostingClassifier(random_state=2021)
gbm_ads5.fit(X_train_ads5,y_train_ads5)
pred = gbm_ads5.predict(X_test)
pred_proba = gbm_ads5.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

ads6 = ADASYN(sampling_strategy=0.95)
X_train_ads6,y_train_ads6 = ads6.fit_resample(X_train,y_train)

gbm_ads6 = GradientBoostingClassifier(random_state=2021)
gbm_ads6.fit(X_train_ads6,y_train_ads6)
pred = gbm_ads6.predict(X_test)
pred_proba = gbm_ads6.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

ads7 = ADASYN()
X_train_ads7,y_train_ads7 = ads7.fit_resample(X_train,y_train)

gbm_ads7 = GradientBoostingClassifier(random_state=2021)
gbm_ads7.fit(X_train_ads7,y_train_ads7)
pred = gbm_ads7.predict(X_test)
pred_proba = gbm_ads7.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
```

```
[[1856  124]
 [  22   44]]
정확도: 0.929 정밀도: 0.262 재현율: 0.667 AUC: 0.927 F1: 0.376 F2: 0.509 Balanced_Accurac
y: 0.802 G-Mean: 0.791
 matthews_corrcoef: 0.389


[[1851  129]
 [  19   47]]
정확도: 0.928 정밀도: 0.267 재현율: 0.712 AUC: 0.929 F1: 0.388 F2: 0.534 Balanced_Accurac
y: 0.823 G-Mean: 0.816
 matthews_corrcoef: 0.408
```

```
[[1853  127]
 [  18   48]]
정확도: 0.929 정밀도: 0.274 재현율: 0.727 AUC: 0.928 F1: 0.398 F2: 0.547 Balanced_Accurac
y: 0.832 G-Mean: 0.825
 matthews_corrcoef: 0.419


[[1841  139]
 [  20   46]]
정확도: 0.922 정밀도: 0.249 재현율: 0.697 AUC: 0.922 F1: 0.367 F2: 0.512 Balanced_Accurac
y: 0.813 G-Mean: 0.805
 matthews_corrcoef: 0.386


[[1837  143]
 [  19   47]]
정확도: 0.921 정밀도: 0.247 재현율: 0.712 AUC: 0.924 F1: 0.367 F2: 0.518 Balanced_Accurac
y: 0.82 G-Mean: 0.813
 matthews_corrcoef: 0.39


[[1833  147]
 [  19   47]]
정확도: 0.919 정밀도: 0.242 재현율: 0.712 AUC: 0.927 F1: 0.362 F2: 0.513 Balanced_Accurac
y: 0.819 G-Mean: 0.812
 matthews_corrcoef: 0.385


[[1837  143]
 [  19   47]]
정확도: 0.921 정밀도: 0.247 재현율: 0.712 AUC: 0.927 F1: 0.367 F2: 0.518 Balanced_Accurac
y: 0.82 G-Mean: 0.813
 matthews_corrcoef: 0.39
```

ADASYN LGBM

In [38]:

```
ads1 = ADASYN(sampling_strategy=0.7)
X_train_ads1,y_train_ads1 = ads1.fit_resample(X_train,y_train)

lgbm_ads1 = LGBMClassifier(random_state=2021)
lgbm_ads1.fit(X_train_ads1,y_train_ads1)
pred = lgbm_ads1.predict(X_test)
pred_proba = lgbm_ads1.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

ads2 = ADASYN(sampling_strategy=0.75)
X_train_ads2,y_train_ads2 = ads2.fit_resample(X_train,y_train)

lgbm_ads2 = LGBMClassifier(random_state=2021)
lgbm_ads2.fit(X_train_ads2,y_train_ads2)
pred = lgbm_ads2.predict(X_test)
pred_proba = lgbm_ads2.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

ads3 = ADASYN(sampling_strategy=0.8)
X_train_ads3,y_train_ads3 = ads3.fit_resample(X_train,y_train)

lgbm_ads3 = LGBMClassifier(random_state=2021)
lgbm_ads3.fit(X_train_ads3,y_train_ads3)
pred = lgbm_ads3.predict(X_test)
pred_proba = lgbm_ads3.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
```

```
print('\n')

ads4 = ADASYN(sampling_strategy=0.85)
X_train_ads4,y_train_ads4 = ads4.fit_resample(X_train,y_train)

lgbm_ads4 = LGBMClassifier(random_state=2021)
lgbm_ads4.fit(X_train_ads4,y_train_ads4)
pred = lgbm_ads4.predict(X_test)
pred_proba = lgbm_ads4.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

ads5 = ADASYN(sampling_strategy=0.9)
X_train_ads5,y_train_ads5 = ads5.fit_resample(X_train,y_train)

lgbm_ads5 = LGBMClassifier(random_state=2021)
lgbm_ads5.fit(X_train_ads5,y_train_ads5)
pred = lgbm_ads5.predict(X_test)
pred_proba = lgbm_ads5.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

ads6 = ADASYN(sampling_strategy=0.95)
X_train_ads6,y_train_ads6 = ads6.fit_resample(X_train,y_train)

lgbm_ads6 = LGBMClassifier(random_state=2021)
lgbm_ads6.fit(X_train_ads6,y_train_ads6)
pred = lgbm_ads6.predict(X_test)
pred_proba = lgbm_ads6.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

ads7 = ADASYN()
X_train_ads7,y_train_ads7 = ads7.fit_resample(X_train,y_train)

lgbm_ads7 = LGBMClassifier(random_state=2021)
lgbm_ads7.fit(X_train_ads7,y_train_ads7)
pred = lgbm_ads7.predict(X_test)
pred_proba = lgbm_ads7.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
```

```
[[1919   61]
 [  24   42]]
정확도: 0.958 정밀도: 0.408 재현율: 0.636 AUC: 0.935 F1: 0.497 F2: 0.572 Balanced_Accurac
y: 0.803 G-Mean: 0.785
 matthews_corrcoef: 0.489


[[1923   57]
 [  25   41]]
정확도: 0.96 정밀도: 0.418 재현율: 0.621 AUC: 0.932 F1: 0.5 F2: 0.566 Balanced_Accuracy:
0.796 G-Mean: 0.777
 matthews_corrcoef: 0.49


[[1924   56]
 [  26   40]]
정확도: 0.96 정밀도: 0.417 재현율: 0.606 AUC: 0.925 F1: 0.494 F2: 0.556 Balanced_Accurac
y: 0.789 G-Mean: 0.767
 matthews_corrcoef: 0.483


[[1919   61]
 [  26   40]]
```

```
정확도: 0.957 정밀도: 0.396 재현율: 0.606 AUC: 0.933 F1: 0.479 F2: 0.548 Balanced_Accurac
y: 0.788 G-Mean: 0.766
 matthews_corrcoef: 0.469


[[1920   60]
 [  25   41]]
정확도: 0.958 정밀도: 0.406 재현율: 0.621 AUC: 0.932 F1: 0.491 F2: 0.562 Balanced_Accurac
y: 0.795 G-Mean: 0.776
 matthews_corrcoef: 0.482


[[1915   65]
 [  26   40]]
정확도: 0.956 정밀도: 0.381 재현율: 0.606 AUC: 0.937 F1: 0.468 F2: 0.542 Balanced_Accurac
y: 0.787 G-Mean: 0.766
 matthews_corrcoef: 0.459


[[1915   65]
 [  26   40]]
정확도: 0.956 정밀도: 0.381 재현율: 0.606 AUC: 0.926 F1: 0.468 F2: 0.542 Balanced_Accurac
y: 0.787 G-Mean: 0.766
 matthews_corrcoef: 0.459
```

ADASYN XGB

In [39]:

```
ads1 = ADASYN(sampling_strategy=0.7)
X_train_ads1,y_train_ads1 = ads1.fit_resample(X_train,y_train)

xgb_ads1 = XGBClassifier(random_state=2021)
xgb_ads1.fit(X_train_ads1,y_train_ads1)
pred = xgb_ads1.predict(X_test)
pred_proba = xgb_ads1.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

ads2 = ADASYN(sampling_strategy=0.75)
X_train_ads2,y_train_ads2 = ads2.fit_resample(X_train,y_train)

xgb_ads2 = XGBClassifier(random_state=2021)
xgb_ads2.fit(X_train_ads2,y_train_ads2)
pred = xgb_ads2.predict(X_test)
pred_proba = xgb_ads2.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

ads3 = ADASYN(sampling_strategy=0.8)
X_train_ads3,y_train_ads3 = ads3.fit_resample(X_train,y_train)

xgb_ads3 = XGBClassifier(random_state=2021)
xgb_ads3.fit(X_train_ads3,y_train_ads3)
pred = xgb_ads3.predict(X_test)
pred_proba = xgb_ads3.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

ads4 = ADASYN(sampling_strategy=0.85)
X_train_ads4,y_train_ads4 = ads4.fit_resample(X_train,y_train)

xgb_ads4 = XGBClassifier(random_state=2021)
xgb_ads4.fit(X_train_ads4,y_train_ads4)
pred = xgb_ads4.predict(X_test)
pred_proba = xgb_ads4.predict_proba(X_test)[:,1]
```

```python
get_clf_eval(y_test,pred,pred_proba)
print('\n')

ads5 = ADASYN(sampling_strategy=0.9)
X_train_ads5,y_train_ads5 = ads5.fit_resample(X_train,y_train)

xgb_ads5 = XGBClassifier(random_state=2021)
xgb_ads5.fit(X_train_ads5,y_train_ads5)
pred = xgb_ads5.predict(X_test)
pred_proba = xgb_ads5.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

ads6 = ADASYN(sampling_strategy=0.95)
X_train_ads6,y_train_ads6 = ads6.fit_resample(X_train,y_train)

xgb_ads6 = XGBClassifier(random_state=2021)
xgb_ads6.fit(X_train_ads6,y_train_ads6)
pred = xgb_ads6.predict(X_test)
pred_proba = xgb_ads6.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

ads7 = ADASYN()
X_train_ads7,y_train_ads7 = ads7.fit_resample(X_train,y_train)

xgb_ads7 = XGBClassifier(random_state=2021)
xgb_ads7.fit(X_train_ads7,y_train_ads7)
pred = xgb_ads7.predict(X_test)
pred_proba = xgb_ads7.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
```

```
[17:21:04] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1925   55]
 [  24   42]]
정확도: 0.961 정밀도: 0.433 재현율: 0.636 AUC: 0.926 F1: 0.515 F2: 0.582 Balanced_Accurac
y: 0.804 G-Mean: 0.787
 matthews_corrcoef: 0.506


[17:21:07] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1920   60]
 [  26   40]]
정확도: 0.958 정밀도: 0.4 재현율: 0.606 AUC: 0.922 F1: 0.482 F2: 0.549 Balanced_Accuracy:
0.788 G-Mean: 0.767
 matthews_corrcoef: 0.472


[17:21:09] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1919   61]
 [  27   39]]
정확도: 0.957 정밀도: 0.39 재현율: 0.591 AUC: 0.924 F1: 0.47 F2: 0.536 Balanced_Accuracy:
0.78 G-Mean: 0.757
 matthews_corrcoef: 0.459
```

```
[17:21:12] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1922   58]
 [  26   40]]
정확도: 0.959 정밀도: 0.408 재현율: 0.606 AUC: 0.931 F1: 0.488 F2: 0.552 Balanced_Accurac
y: 0.788 G-Mean: 0.767
 matthews_corrcoef: 0.477


[17:21:15] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1920   60]
 [  26   40]]
정확도: 0.958 정밀도: 0.4 재현율: 0.606 AUC: 0.919 F1: 0.482 F2: 0.549 Balanced_Accuracy:
0.788 G-Mean: 0.767
 matthews_corrcoef: 0.472


[17:21:18] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1919   61]
 [  25   41]]
정확도: 0.958 정밀도: 0.402 재현율: 0.621 AUC: 0.933 F1: 0.488 F2: 0.56 Balanced_Accurac
y: 0.795 G-Mean: 0.776
 matthews_corrcoef: 0.479


[17:21:21] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1917   63]
 [  23   43]]
정확도: 0.958 정밀도: 0.406 재현율: 0.652 AUC: 0.93 F1: 0.5 F2: 0.581 Balanced_Accuracy:
0.81 G-Mean: 0.794
 matthews_corrcoef: 0.494
```

Resampling 4.) Borderline-SMOTE

In [40]:

```
from imblearn.over_sampling import BorderlineSMOTE
```

Borderline-SMOTE GBM

In [41]:

```
bd_smote1 = BorderlineSMOTE(sampling_strategy=0.7)
X_train_bd_smote1,y_train_bd_smote1 = bd_smote1.fit_resample(X_train,y_train)

gbm_bd_smote1 = GradientBoostingClassifier(random_state=2021)
gbm_bd_smote1.fit(X_train_bd_smote1,y_train_bd_smote1)
pred = gbm_bd_smote1.predict(X_test)
pred_proba = gbm_bd_smote1.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

bd_smote2 = BorderlineSMOTE(sampling_strategy=0.75)
X_train_bd_smote2,y_train_bd_smote2 = bd_smote2.fit_resample(X_train,y_train)

gbm_bd_smote2 = GradientBoostingClassifier(random_state=2021)
```

```
gbm_bd_smote2.fit(X_train_bd_smote2,y_train_bd_smote2)
pred = gbm_bd_smote2.predict(X_test)
pred_proba = gbm_bd_smote2.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

bd_smote3 = BorderlineSMOTE(sampling_strategy=0.8)
X_train_bd_smote3,y_train_bd_smote3 = bd_smote3.fit_resample(X_train,y_train)

gbm_bd_smote3 = GradientBoostingClassifier(random_state=2021)
gbm_bd_smote3.fit(X_train_bd_smote3,y_train_bd_smote3)
pred = gbm_bd_smote3.predict(X_test)
pred_proba = gbm_bd_smote3.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

bd_smote4 =  BorderlineSMOTE(sampling_strategy=0.85)
X_train_bd_smote4,y_train_bd_smote4 = bd_smote4.fit_resample(X_train,y_train)

gbm_bd_smote4 = GradientBoostingClassifier(random_state=2021)
gbm_bd_smote4.fit(X_train_bd_smote4,y_train_bd_smote4)
pred = gbm_bd_smote4.predict(X_test)
pred_proba = gbm_bd_smote4.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

bd_smote5 =  BorderlineSMOTE(sampling_strategy=0.9)
X_train_bd_smote5,y_train_bd_smote5 = bd_smote5.fit_resample(X_train,y_train)

gbm_bd_smote5 = GradientBoostingClassifier(random_state=2021)
gbm_bd_smote5.fit(X_train_bd_smote5,y_train_bd_smote5)
pred = gbm_bd_smote5.predict(X_test)
pred_proba = gbm_bd_smote5.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

bd_smote6 =  BorderlineSMOTE(sampling_strategy=0.95)
X_train_bd_smote6,y_train_bd_smote6 = bd_smote6.fit_resample(X_train,y_train)

gbm_bd_smote6 = GradientBoostingClassifier(random_state=2021)
gbm_bd_smote6.fit(X_train_bd_smote6,y_train_bd_smote6)
pred = gbm_bd_smote6.predict(X_test)
pred_proba = gbm_bd_smote6.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

bd_smote7 = BorderlineSMOTE()
X_train_bd_smote7,y_train_bd_smote7 = bd_smote7.fit_resample(X_train,y_train)

gbm_bd_smote7 = GradientBoostingClassifier(random_state=2021)
gbm_bd_smote7.fit(X_train_bd_smote7,y_train_bd_smote7)
pred = gbm_bd_smote7.predict(X_test)
pred_proba = gbm_bd_smote7.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
```

```
[[1888   92]
 [  26   40]]
정확도: 0.942 정밀도: 0.303 재현율: 0.606 AUC: 0.919 F1: 0.404 F2: 0.505 Balanced_Accurac
y: 0.78 G-Mean: 0.76
 matthews_corrcoef: 0.402


[[1885   95]
```

```
 [  24   42]]
정확도: 0.942 정밀도: 0.307 재현율: 0.636 AUC: 0.917 F1: 0.414 F2: 0.524 Balanced_Accurac
y: 0.794 G-Mean: 0.778
 matthews_corrcoef: 0.416


[[1879  101]
 [  24   42]]
정확도: 0.939 정밀도: 0.294 재현율: 0.636 AUC: 0.92 F1: 0.402 F2: 0.516 Balanced_Accurac
y: 0.793 G-Mean: 0.777
 matthews_corrcoef: 0.406


[[1875  105]
 [  24   42]]
정확도: 0.937 정밀도: 0.286 재현율: 0.636 AUC: 0.922 F1: 0.394 F2: 0.511 Balanced_Accurac
y: 0.792 G-Mean: 0.776
 matthews_corrcoef: 0.399


[[1877  103]
 [  25   41]]
정확도: 0.937 정밀도: 0.285 재현율: 0.621 AUC: 0.916 F1: 0.39 F2: 0.502 Balanced_Accurac
y: 0.785 G-Mean: 0.767
 matthews_corrcoef: 0.393


[[1873  107]
 [  24   42]]
정확도: 0.936 정밀도: 0.282 재현율: 0.636 AUC: 0.92 F1: 0.391 F2: 0.508 Balanced_Accurac
y: 0.791 G-Mean: 0.776
 matthews_corrcoef: 0.396


[[1873  107]
 [  26   40]]
정확도: 0.935 정밀도: 0.272 재현율: 0.606 AUC: 0.919 F1: 0.376 F2: 0.487 Balanced_Accurac
y: 0.776 G-Mean: 0.757
 matthews_corrcoef: 0.378
```

Borderline-SMOTE LGBM

In [42]:

```python
bd_smote1 = BorderlineSMOTE(sampling_strategy=0.7)
X_train_bd_smote1,y_train_bd_smote1 = bd_smote1.fit_resample(X_train,y_train)

lgbm_bd_smote1 = LGBMClassifier(random_state=2021)
lgbm_bd_smote1.fit(X_train_bd_smote1,y_train_bd_smote1)
pred = lgbm_bd_smote1.predict(X_test)
pred_proba = lgbm_bd_smote1.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

bd_smote2 = BorderlineSMOTE(sampling_strategy=0.75)
X_train_bd_smote2,y_train_bd_smote2 = bd_smote2.fit_resample(X_train,y_train)

lgbm_bd_smote2 = LGBMClassifier(random_state=2021)
lgbm_bd_smote2.fit(X_train_bd_smote2,y_train_bd_smote2)
pred = lgbm_bd_smote2.predict(X_test)
pred_proba = lgbm_bd_smote2.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

bd_smote3 = BorderlineSMOTE(sampling_strategy=0.8)
X_train_bd_smote3,y_train_bd_smote3 = bd_smote3.fit_resample(X_train,y_train)
```

```python
lgbm_bd_smote3 = LGBMClassifier(random_state=2021)
lgbm_bd_smote3.fit(X_train_bd_smote3,y_train_bd_smote3)
pred = lgbm_bd_smote3.predict(X_test)
pred_proba = lgbm_bd_smote3.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

bd_smote4 =  BorderlineSMOTE(sampling_strategy=0.85)
X_train_bd_smote4,y_train_bd_smote4 = bd_smote4.fit_resample(X_train,y_train)

lgbm_bd_smote4 = LGBMClassifier(random_state=2021)
lgbm_bd_smote4.fit(X_train_bd_smote4,y_train_bd_smote4)
pred = lgbm_bd_smote4.predict(X_test)
pred_proba = lgbm_bd_smote4.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

bd_smote5 =  BorderlineSMOTE(sampling_strategy=0.9)
X_train_bd_smote5,y_train_bd_smote5 = bd_smote5.fit_resample(X_train,y_train)

lgbm_bd_smote5 = LGBMClassifier(random_state=2021)
lgbm_bd_smote5.fit(X_train_bd_smote5,y_train_bd_smote5)
pred = lgbm_bd_smote5.predict(X_test)
pred_proba = lgbm_bd_smote5.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

bd_smote6 =  BorderlineSMOTE(sampling_strategy=0.95)
X_train_bd_smote6,y_train_bd_smote6 = bd_smote6.fit_resample(X_train,y_train)

lgbm_bd_smote6 = LGBMClassifier(random_state=2021)
lgbm_bd_smote6.fit(X_train_bd_smote6,y_train_bd_smote6)
pred = lgbm_bd_smote6.predict(X_test)
pred_proba = lgbm_bd_smote6.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

bd_smote7 = BorderlineSMOTE()
X_train_bd_smote7,y_train_bd_smote7 = bd_smote7.fit_resample(X_train,y_train)

lgbm_bd_smote7 = LGBMClassifier(random_state=2021)
lgbm_bd_smote7.fit(X_train_bd_smote7,y_train_bd_smote7)
pred = lgbm_bd_smote7.predict(X_test)
pred_proba = lgbm_bd_smote7.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
```

```
[[1935   45]
 [  33   33]]
정확도: 0.962 정밀도: 0.423 재현율: 0.5 AUC: 0.937 F1: 0.458 F2: 0.482 Balanced_Accuracy:
0.739 G-Mean: 0.699
 matthews_corrcoef: 0.44


[[1937   43]
 [  34   32]]
정확도: 0.962 정밀도: 0.427 재현율: 0.485 AUC: 0.926 F1: 0.454 F2: 0.472 Balanced_Accurac
y: 0.732 G-Mean: 0.689
 matthews_corrcoef: 0.435


[[1932   48]
 [  31   35]]
정확도: 0.961 정밀도: 0.422 재현율: 0.53 AUC: 0.929 F1: 0.47 F2: 0.504 Balanced_Accuracy:
0.753 G-Mean: 0.719
```

```
matthews_corrcoef: 0.453


[[1936   44]
 [  31   35]]
정확도: 0.963 정밀도: 0.443 재현율: 0.53 AUC: 0.931 F1: 0.483 F2: 0.51 Balanced_Accuracy: 0.754 G-Mean: 0.72
 matthews_corrcoef: 0.466


[[1938   42]
 [  33   33]]
정확도: 0.963 정밀도: 0.44 재현율: 0.5 AUC: 0.928 F1: 0.468 F2: 0.487 Balanced_Accuracy: 0.739 G-Mean: 0.7
 matthews_corrcoef: 0.45


[[1938   42]
 [  31   35]]
정확도: 0.964 정밀도: 0.455 재현율: 0.53 AUC: 0.931 F1: 0.49 F2: 0.513 Balanced_Accuracy: 0.755 G-Mean: 0.72
 matthews_corrcoef: 0.473


[[1932   48]
 [  31   35]]
정확도: 0.961 정밀도: 0.422 재현율: 0.53 AUC: 0.932 F1: 0.47 F2: 0.504 Balanced_Accuracy: 0.753 G-Mean: 0.719
 matthews_corrcoef: 0.453
```

Borderline-SMOTE XGB

In [43]:

```
bd_smote1 = BorderlineSMOTE(sampling_strategy=0.7)
X_train_bd_smote1,y_train_bd_smote1 = bd_smote1.fit_resample(X_train,y_train)

xgb_bd_smote1 = XGBClassifier(random_state=2021)
xgb_bd_smote1.fit(X_train_bd_smote1,y_train_bd_smote1)
pred = xgb_bd_smote1.predict(X_test)
pred_proba = xgb_bd_smote1.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

bd_smote2 = BorderlineSMOTE(sampling_strategy=0.75)
X_train_bd_smote2,y_train_bd_smote2 = bd_smote2.fit_resample(X_train,y_train)

xgb_bd_smote2 = XGBClassifier(random_state=2021)
xgb_bd_smote2.fit(X_train_bd_smote2,y_train_bd_smote2)
pred = xgb_bd_smote2.predict(X_test)
pred_proba = xgb_bd_smote2.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

bd_smote3 = BorderlineSMOTE(sampling_strategy=0.8)
X_train_bd_smote3,y_train_bd_smote3 = bd_smote3.fit_resample(X_train,y_train)

xgb_bd_smote3 = XGBClassifier(random_state=2021)
xgb_bd_smote3.fit(X_train_bd_smote3,y_train_bd_smote3)
pred = xgb_bd_smote3.predict(X_test)
pred_proba = xgb_bd_smote3.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

bd_smote4 =  BorderlineSMOTE(sampling_strategy=0.85)
X_train_bd_smote4,y_train_bd_smote4 = bd_smote4.fit_resample(X_train,y_train)
```

```
xgb_bd_smote4 = XGBClassifier(random_state=2021)
xgb_bd_smote4.fit(X_train_bd_smote4,y_train_bd_smote4)
pred = xgb_bd_smote4.predict(X_test)
pred_proba = xgb_bd_smote4.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

bd_smote5 =  BorderlineSMOTE(sampling_strategy=0.9)
X_train_bd_smote5,y_train_bd_smote5 = bd_smote5.fit_resample(X_train,y_train)

xgb_bd_smote5 = XGBClassifier(random_state=2021)
xgb_bd_smote5.fit(X_train_bd_smote5,y_train_bd_smote5)
pred = xgb_bd_smote5.predict(X_test)
pred_proba = xgb_bd_smote5.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

bd_smote6 =  BorderlineSMOTE(sampling_strategy=0.95)
X_train_bd_smote6,y_train_bd_smote6 = bd_smote6.fit_resample(X_train,y_train)

xgb_bd_smote6 = XGBClassifier(random_state=2021)
xgb_bd_smote6.fit(X_train_bd_smote6,y_train_bd_smote6)
pred = xgb_bd_smote6.predict(X_test)
pred_proba = xgb_bd_smote6.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

bd_smote7 = BorderlineSMOTE()
X_train_bd_smote7,y_train_bd_smote7 = bd_smote7.fit_resample(X_train,y_train)

xgb_bd_smote7 = XGBClassifier(random_state=2021)
xgb_bd_smote7.fit(X_train_bd_smote7,y_train_bd_smote7)
pred = xgb_bd_smote7.predict(X_test)
pred_proba = xgb_bd_smote7.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
```

```
[17:22:35] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1939   41]
 [  31   35]]
정확도: 0.965 정밀도: 0.461 재현율: 0.53 AUC: 0.921 F1: 0.493 F2: 0.515 Balanced_Accurac
y: 0.755 G-Mean: 0.721
 matthews_corrcoef: 0.476


[17:22:37] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1932   48]
 [  31   35]]
정확도: 0.961 정밀도: 0.422 재현율: 0.53 AUC: 0.929 F1: 0.47 F2: 0.504 Balanced_Accuracy:
0.753 G-Mean: 0.719
 matthews_corrcoef: 0.453


[17:22:40] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1925   55]
 [  29   37]]
```

```
정확도: 0.959 정밀도: 0.402 재현율: 0.561 AUC: 0.929 F1: 0.468 F2: 0.52 Balanced_Accurac
y: 0.766 G-Mean: 0.738
 matthews_corrcoef: 0.454


[17:22:42] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1930   50]
 [  34   32]]
정확도: 0.959 정밀도: 0.39 재현율: 0.485 AUC: 0.917 F1: 0.432 F2: 0.462 Balanced_Accurac
y: 0.73 G-Mean: 0.687
 matthews_corrcoef: 0.414


[17:22:45] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1926   54]
 [  30   36]]
정확도: 0.959 정밀도: 0.4 재현율: 0.545 AUC: 0.923 F1: 0.462 F2: 0.508 Balanced_Accuracy:
0.759 G-Mean: 0.728
 matthews_corrcoef: 0.446


[17:22:48] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1931   49]
 [  34   32]]
정확도: 0.959 정밀도: 0.395 재현율: 0.485 AUC: 0.925 F1: 0.435 F2: 0.464 Balanced_Accurac
y: 0.73 G-Mean: 0.688
 matthews_corrcoef: 0.417


[17:22:51] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1924   56]
 [  33   33]]
정확도: 0.957 정밀도: 0.371 재현율: 0.5 AUC: 0.928 F1: 0.426 F2: 0.467 Balanced_Accuracy:
0.736 G-Mean: 0.697
 matthews_corrcoef: 0.409
```

Resampling 5.) SVM SMOTE

In [44]:

```
from imblearn.over_sampling import SVMSMOTE
```

SVM SMOTE GBM

In [45]:

```
svm_smote1 = SVMSMOTE(sampling_strategy=0.7)
X_train_svm_smote1,y_train_svm_smote1 = svm_smote1.fit_resample(X_train,y_train)

gbm_svm_smote1 = GradientBoostingClassifier(random_state=2021)
gbm_svm_smote1.fit(X_train_svm_smote1,y_train_svm_smote1)
pred = gbm_svm_smote1.predict(X_test)
pred_proba = gbm_svm_smote1.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')
```

```python
svm_smote2 = SVMSMOTE(sampling_strategy=0.75)
X_train_svm_smote2,y_train_svm_smote2 = svm_smote2.fit_resample(X_train,y_train)

gbm_svm_smote2 = GradientBoostingClassifier(random_state=2021)
gbm_svm_smote2.fit(X_train_svm_smote2,y_train_svm_smote2)
pred = gbm_svm_smote2.predict(X_test)
pred_proba = gbm_svm_smote2.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

svm_smote3 = SVMSMOTE(sampling_strategy=0.8)
X_train_svm_smote3,y_train_svm_smote3 = svm_smote3.fit_resample(X_train,y_train)

gbm_svm_smote3 = GradientBoostingClassifier(random_state=2021)
gbm_svm_smote3.fit(X_train_svm_smote3,y_train_svm_smote3)
pred = gbm_svm_smote3.predict(X_test)
pred_proba = gbm_svm_smote3.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

svm_smote4 =  SVMSMOTE(sampling_strategy=0.85)
X_train_svm_smote4,y_train_svm_smote4 = svm_smote4.fit_resample(X_train,y_train)

gbm_svm_smote4 = GradientBoostingClassifier(random_state=2021)
gbm_svm_smote4.fit(X_train_svm_smote4,y_train_svm_smote4)
pred = gbm_svm_smote4.predict(X_test)
pred_proba = gbm_svm_smote4.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

svm_smote5 =  SVMSMOTE(sampling_strategy=0.9)
X_train_svm_smote5,y_train_svm_smote5 = svm_smote5.fit_resample(X_train,y_train)

gbm_svm_smote5 = GradientBoostingClassifier(random_state=2021)
gbm_svm_smote5.fit(X_train_svm_smote5,y_train_svm_smote5)
pred = gbm_svm_smote5.predict(X_test)
pred_proba = gbm_svm_smote5.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

svm_smote6 =  SVMSMOTE(sampling_strategy=0.95)
X_train_svm_smote6,y_train_svm_smote6 = svm_smote6.fit_resample(X_train,y_train)

gbm_svm_smote6 = GradientBoostingClassifier(random_state=2021)
gbm_svm_smote6.fit(X_train_svm_smote6,y_train_svm_smote6)
pred = gbm_svm_smote6.predict(X_test)
pred_proba = gbm_svm_smote6.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

svm_smote7 = SVMSMOTE()
X_train_svm_smote7,y_train_svm_smote7 = svm_smote7.fit_resample(X_train,y_train)

gbm_svm_smote7 = GradientBoostingClassifier(random_state=2021)
gbm_svm_smote7.fit(X_train_svm_smote7,y_train_svm_smote7)
pred = gbm_svm_smote7.predict(X_test)
pred_proba = gbm_svm_smote7.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
```

```
[[1917   63]
 [  27   39]]
정확도: 0.956 정밀도: 0.382 재현율: 0.591 AUC: 0.926 F1: 0.464 F2: 0.533 Balanced_Accurac
y: 0.78 G-Mean: 0.756
```

```
matthews_corrcoef: 0.454


[[1896   84]
 [  27   39]]
정확도: 0.946 정밀도: 0.317 재현율: 0.591 AUC: 0.919 F1: 0.413 F2: 0.504 Balanced_Accurac
y: 0.774 G-Mean: 0.752
 matthews_corrcoef: 0.408


[[1917   63]
 [  27   39]]
정확도: 0.956 정밀도: 0.382 재현율: 0.591 AUC: 0.92 F1: 0.464 F2: 0.533 Balanced_Accurac
y: 0.78 G-Mean: 0.756
 matthews_corrcoef: 0.454


[[1903   77]
 [  26   40]]
정확도: 0.95 정밀도: 0.342 재현율: 0.606 AUC: 0.92 F1: 0.437 F2: 0.525 Balanced_Accuracy:
0.784 G-Mean: 0.763
 matthews_corrcoef: 0.432


[[1906   74]
 [  29   37]]
정확도: 0.95 정밀도: 0.333 재현율: 0.561 AUC: 0.921 F1: 0.418 F2: 0.493 Balanced_Accurac
y: 0.762 G-Mean: 0.735
 matthews_corrcoef: 0.408


[[1901   79]
 [  26   40]]
정확도: 0.949 정밀도: 0.336 재현율: 0.606 AUC: 0.924 F1: 0.432 F2: 0.522 Balanced_Accurac
y: 0.783 G-Mean: 0.763
 matthews_corrcoef: 0.427


[[1894   86]
 [  25   41]]
정확도: 0.946 정밀도: 0.323 재현율: 0.621 AUC: 0.923 F1: 0.425 F2: 0.524 Balanced_Accurac
y: 0.789 G-Mean: 0.771
 matthews_corrcoef: 0.423
```

SVM SMOTE LGBM

In [46]:

```
svm_smote1 = SVMSMOTE(sampling_strategy=0.7)
X_train_svm_smote1,y_train_svm_smote1 = svm_smote1.fit_resample(X_train,y_train)

lgbm_svm_smote1 = LGBMClassifier(random_state=2021)
lgbm_svm_smote1.fit(X_train_svm_smote1,y_train_svm_smote1)
pred = lgbm_svm_smote1.predict(X_test)
pred_proba = lgbm_svm_smote1.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

svm_smote2 = SVMSMOTE(sampling_strategy=0.75)
X_train_svm_smote2,y_train_svm_smote2 = svm_smote2.fit_resample(X_train,y_train)

lgbm_svm_smote2 = LGBMClassifier(random_state=2021)
lgbm_svm_smote2.fit(X_train_svm_smote2,y_train_svm_smote2)
pred = lgbm_svm_smote2.predict(X_test)
pred_proba = lgbm_svm_smote2.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')
```

```python
svm_smote3 = SVMSMOTE(sampling_strategy=0.8)
X_train_svm_smote3,y_train_svm_smote3 = svm_smote3.fit_resample(X_train,y_train)

lgbm_svm_smote3 = LGBMClassifier(random_state=2021)
lgbm_svm_smote3.fit(X_train_svm_smote3,y_train_svm_smote3)
pred = lgbm_svm_smote3.predict(X_test)
pred_proba = lgbm_svm_smote3.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

svm_smote4 =  SVMSMOTE(sampling_strategy=0.85)
X_train_svm_smote4,y_train_svm_smote4 = svm_smote4.fit_resample(X_train,y_train)

lgbm_svm_smote4 = LGBMClassifier(random_state=2021)
lgbm_svm_smote4.fit(X_train_svm_smote4,y_train_svm_smote4)
pred = lgbm_svm_smote4.predict(X_test)
pred_proba = lgbm_svm_smote4.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

svm_smote5 =  SVMSMOTE(sampling_strategy=0.9)
X_train_svm_smote5,y_train_svm_smote5 = svm_smote5.fit_resample(X_train,y_train)

lgbm_svm_smote5 = LGBMClassifier(random_state=2021)
lgbm_svm_smote5.fit(X_train_svm_smote5,y_train_svm_smote5)
pred = lgbm_svm_smote5.predict(X_test)
pred_proba = lgbm_svm_smote5.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

svm_smote6 =  SVMSMOTE(sampling_strategy=0.95)
X_train_svm_smote6,y_train_svm_smote6 = svm_smote6.fit_resample(X_train,y_train)

lgbm_svm_smote6 = LGBMClassifier(random_state=2021)
lgbm_svm_smote6.fit(X_train_svm_smote6,y_train_svm_smote6)
pred = lgbm_svm_smote6.predict(X_test)
pred_proba = lgbm_svm_smote6.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

svm_smote7 = SVMSMOTE()
X_train_svm_smote7,y_train_svm_smote7 = svm_smote7.fit_resample(X_train,y_train)

lgbm_svm_smote7 = LGBMClassifier(random_state=2021)
lgbm_svm_smote7.fit(X_train_svm_smote7,y_train_svm_smote7)
pred = lgbm_svm_smote7.predict(X_test)
pred_proba = lgbm_svm_smote7.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
```

```
[[1944   36]
 [  37   29]]
정확도: 0.964 정밀도: 0.446 재현율: 0.439 AUC: 0.928 F1: 0.443 F2: 0.441 Balanced_Accurac
y: 0.711 G-Mean: 0.657
 matthews_corrcoef: 0.424


[[1938   42]
 [  36   30]]
정확도: 0.962 정밀도: 0.417 재현율: 0.455 AUC: 0.928 F1: 0.435 F2: 0.446 Balanced_Accurac
y: 0.717 G-Mean: 0.667
 matthews_corrcoef: 0.416
```

```
[[1941   39]
 [  35   31]]
정확도: 0.964 정밀도: 0.443 재현율: 0.47 AUC: 0.934 F1: 0.456 F2: 0.464 Balanced_Accurac
y: 0.725 G-Mean: 0.679
 matthews_corrcoef: 0.437


[[1940   40]
 [  36   30]]
정확도: 0.963 정밀도: 0.429 재현율: 0.455 AUC: 0.932 F1: 0.441 F2: 0.449 Balanced_Accurac
y: 0.717 G-Mean: 0.667
 matthews_corrcoef: 0.422


[[1949   31]
 [  33   33]]
정확도: 0.969 정밀도: 0.516 재현율: 0.5 AUC: 0.929 F1: 0.508 F2: 0.503 Balanced_Accuracy:
0.742 G-Mean: 0.702
 matthews_corrcoef: 0.492


[[1948   32]
 [  36   30]]
정확도: 0.967 정밀도: 0.484 재현율: 0.455 AUC: 0.929 F1: 0.469 F2: 0.46 Balanced_Accurac
y: 0.719 G-Mean: 0.669
 matthews_corrcoef: 0.452


[[1937   43]
 [  32   34]]
정확도: 0.963 정밀도: 0.442 재현율: 0.515 AUC: 0.934 F1: 0.476 F2: 0.499 Balanced_Accurac
y: 0.747 G-Mean: 0.71
 matthews_corrcoef: 0.458
```

SVM SMOTE XGB

In [47]:

```
svm_smote1 = SVMSMOTE(sampling_strategy=0.7)
X_train_svm_smote1,y_train_svm_smote1 = svm_smote1.fit_resample(X_train,y_train)

xgb_svm_smote1 = XGBClassifier(random_state=2021)
xgb_svm_smote1.fit(X_train_svm_smote1,y_train_svm_smote1)
pred = xgb_svm_smote1.predict(X_test)
pred_proba = xgb_svm_smote1.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

svm_smote2 = SVMSMOTE(sampling_strategy=0.75)
X_train_svm_smote2,y_train_svm_smote2 = svm_smote2.fit_resample(X_train,y_train)

xgb_svm_smote2 = XGBClassifier(random_state=2021)
xgb_svm_smote2.fit(X_train_svm_smote2,y_train_svm_smote2)
pred = xgb_svm_smote2.predict(X_test)
pred_proba = xgb_svm_smote2.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

svm_smote3 = SVMSMOTE(sampling_strategy=0.8)
X_train_svm_smote3,y_train_svm_smote3 = svm_smote3.fit_resample(X_train,y_train)

xgb_svm_smote3 = XGBClassifier(random_state=2021)
xgb_svm_smote3.fit(X_train_svm_smote3,y_train_svm_smote3)
pred = xgb_svm_smote3.predict(X_test)
pred_proba = xgb_svm_smote3.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')
```

```python
svm_smote4 =  SVMSMOTE(sampling_strategy=0.85)
X_train_svm_smote4,y_train_svm_smote4 = svm_smote4.fit_resample(X_train,y_train)

xgb_svm_smote4 = XGBClassifier(random_state=2021)
xgb_svm_smote4.fit(X_train_svm_smote4,y_train_svm_smote4)
pred = xgb_svm_smote4.predict(X_test)
pred_proba = xgb_svm_smote4.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

svm_smote5 =  SVMSMOTE(sampling_strategy=0.9)
X_train_svm_smote5,y_train_svm_smote5 = svm_smote5.fit_resample(X_train,y_train)

xgb_svm_smote5 = XGBClassifier(random_state=2021)
xgb_svm_smote5.fit(X_train_svm_smote5,y_train_svm_smote5)
pred = xgb_svm_smote5.predict(X_test)
pred_proba = xgb_svm_smote5.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

svm_smote6 =  SVMSMOTE(sampling_strategy=0.95)
X_train_svm_smote6,y_train_svm_smote6 = svm_smote6.fit_resample(X_train,y_train)

xgb_svm_smote6 = XGBClassifier(random_state=2021)
xgb_svm_smote6.fit(X_train_svm_smote6,y_train_svm_smote6)
pred = xgb_svm_smote6.predict(X_test)
pred_proba = xgb_svm_smote6.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

svm_smote7 = SVMSMOTE()
X_train_svm_smote7,y_train_svm_smote7 = svm_smote7.fit_resample(X_train,y_train)

xgb_svm_smote7 = XGBClassifier(random_state=2021)
xgb_svm_smote7.fit(X_train_svm_smote7,y_train_svm_smote7)
pred = xgb_svm_smote7.predict(X_test)
pred_proba = xgb_svm_smote7.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
```

```
[17:23:59] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1944   36]
 [  37   29]]
정확도: 0.964 정밀도: 0.446 재현율: 0.439 AUC: 0.927 F1: 0.443 F2: 0.441 Balanced_Accurac
y: 0.711 G-Mean: 0.657
 matthews_corrcoef: 0.424


[17:24:02] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1941   39]
 [  35   31]]
정확도: 0.964 정밀도: 0.443 재현율: 0.47 AUC: 0.918 F1: 0.456 F2: 0.464 Balanced_Accurac
y: 0.725 G-Mean: 0.679
 matthews_corrcoef: 0.437


[17:24:04] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
```

```
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1938   42]
 [  31   35]]
정확도: 0.964 정밀도: 0.455 재현율: 0.53 AUC: 0.925 F1: 0.49 F2: 0.513 Balanced_Accuracy:
0.755 G-Mean: 0.72
 matthews_corrcoef: 0.473


[17:24:07] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1946   34]
 [  33   33]]
정확도: 0.967 정밀도: 0.493 재현율: 0.5 AUC: 0.931 F1: 0.496 F2: 0.498 Balanced_Accuracy:
0.741 G-Mean: 0.701
 matthews_corrcoef: 0.479


[17:24:10] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1939   41]
 [  36   30]]
정확도: 0.962 정밀도: 0.423 재현율: 0.455 AUC: 0.926 F1: 0.438 F2: 0.448 Balanced_Accurac
y: 0.717 G-Mean: 0.667
 matthews_corrcoef: 0.419


[17:24:12] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1943   37]
 [  34   32]]
정확도: 0.965 정밀도: 0.464 재현율: 0.485 AUC: 0.921 F1: 0.474 F2: 0.48 Balanced_Accurac
y: 0.733 G-Mean: 0.69
 matthews_corrcoef: 0.456


[17:24:15] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1932   48]
 [  30   36]]
정확도: 0.962 정밀도: 0.429 재현율: 0.545 AUC: 0.929 F1: 0.48 F2: 0.517 Balanced_Accurac
y: 0.761 G-Mean: 0.73
 matthews_corrcoef: 0.464
```

Resampling 6.) SMOTE ENN

In [48]:

```python
from imblearn.combine import SMOTEENN
```

SMOTE ENN GBM

In [49]:

```python
smote_enn1 = SMOTEENN(sampling_strategy=0.7)
X_train_smote_enn1,y_train_smote_enn1 = smote_enn1.fit_resample(X_train,y_train)

gbm_smote_enn1 = GradientBoostingClassifier(random_state=2021)
gbm_smote_enn1.fit(X_train_smote_enn1,y_train_smote_enn1)
pred = gbm_smote_enn1.predict(X_test)
```

```
pred_proba = gbm_smote_enn1.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote_enn2 = SMOTEENN(sampling_strategy=0.75)
X_train_smote_enn2,y_train_smote_enn2 = smote_enn2.fit_resample(X_train,y_train)

gbm_smote_enn2 = GradientBoostingClassifier(random_state=2021)
gbm_smote_enn2.fit(X_train_smote_enn2,y_train_smote_enn2)
pred = gbm_smote_enn2.predict(X_test)
pred_proba = gbm_smote_enn2.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote_enn3 = SMOTEENN(sampling_strategy=0.8)
X_train_smote_enn3,y_train_smote_enn3 = smote_enn3.fit_resample(X_train,y_train)

gbm_smote_enn3 = GradientBoostingClassifier(random_state=2021)
gbm_smote_enn3.fit(X_train_smote_enn3,y_train_smote_enn3)
pred = gbm_smote_enn3.predict(X_test)
pred_proba = gbm_smote_enn3.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote_enn4 =  SMOTEENN(sampling_strategy=0.85)
X_train_smote_enn4,y_train_smote_enn4 = smote_enn4.fit_resample(X_train,y_train)

gbm_smote_enn4 = GradientBoostingClassifier(random_state=2021)
gbm_smote_enn4.fit(X_train_smote_enn4,y_train_smote_enn4)
pred = gbm_smote_enn4.predict(X_test)
pred_proba = gbm_smote_enn4.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote_enn5 =  SMOTEENN(sampling_strategy=0.9)
X_train_smote_enn5,y_train_smote_enn5 = smote_enn5.fit_resample(X_train,y_train)

gbm_smote_enn5 = GradientBoostingClassifier(random_state=2021)
gbm_smote_enn5.fit(X_train_smote_enn5,y_train_smote_enn5)
pred = gbm_smote_enn5.predict(X_test)
pred_proba = gbm_smote_enn5.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote_enn6 =  SMOTEENN(sampling_strategy=0.95)
X_train_smote_enn6,y_train_smote_enn6 = smote_enn6.fit_resample(X_train,y_train)

gbm_smote_enn6 = GradientBoostingClassifier(random_state=2021)
gbm_smote_enn6.fit(X_train_smote_enn6,y_train_smote_enn6)
pred = gbm_smote_enn6.predict(X_test)
pred_proba = gbm_smote_enn6.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote_enn7 = SMOTEENN()
X_train_smote_enn7,y_train_smote_enn7 = smote_enn7.fit_resample(X_train,y_train)

gbm_smote_enn7 = GradientBoostingClassifier(random_state=2021)
gbm_smote_enn7.fit(X_train_smote_enn7,y_train_smote_enn7)
pred = gbm_smote_enn7.predict(X_test)
pred_proba = gbm_smote_enn7.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
```

```
[[1851  129]
 [  18   48]]
정확도: 0.928 정밀도: 0.271 재현율: 0.727 AUC: 0.922 F1: 0.395 F2: 0.544 Balanced_Accurac
y: 0.831 G-Mean: 0.825
 matthews_corrcoef: 0.416


[[1829  151]
 [  17   49]]
정확도: 0.918 정밀도: 0.245 재현율: 0.742 AUC: 0.926 F1: 0.368 F2: 0.528 Balanced_Accurac
y: 0.833 G-Mean: 0.828
 matthews_corrcoef: 0.396


[[1843  137]
 [  18   48]]
정확도: 0.924 정밀도: 0.259 재현율: 0.727 AUC: 0.926 F1: 0.382 F2: 0.535 Balanced_Accurac
y: 0.829 G-Mean: 0.823
 matthews_corrcoef: 0.405


[[1827  153]
 [  17   49]]
정확도: 0.917 정밀도: 0.243 재현율: 0.742 AUC: 0.927 F1: 0.366 F2: 0.526 Balanced_Accurac
y: 0.833 G-Mean: 0.828
 matthews_corrcoef: 0.394


[[1817  163]
 [  18   48]]
정확도: 0.912 정밀도: 0.227 재현율: 0.727 AUC: 0.923 F1: 0.347 F2: 0.505 Balanced_Accurac
y: 0.822 G-Mean: 0.817
 matthews_corrcoef: 0.375


[[1808  172]
 [  17   49]]
정확도: 0.908 정밀도: 0.222 재현율: 0.742 AUC: 0.922 F1: 0.341 F2: 0.505 Balanced_Accurac
y: 0.828 G-Mean: 0.823
 matthews_corrcoef: 0.373


[[1812  168]
 [  17   49]]
정확도: 0.91 정밀도: 0.226 재현율: 0.742 AUC: 0.924 F1: 0.346 F2: 0.509 Balanced_Accurac
y: 0.829 G-Mean: 0.824
 matthews_corrcoef: 0.377
```

SMOTE ENN Light GBM

In [50]:

```
smote_enn1 = SMOTEENN(sampling_strategy=0.7)
X_train_smote_enn1,y_train_smote_enn1 = smote_enn1.fit_resample(X_train,y_train)

lgbm_smote_enn1 = LGBMClassifier(random_state=2021)
lgbm_smote_enn1.fit(X_train_smote_enn1,y_train_smote_enn1)
pred = lgbm_smote_enn1.predict(X_test)
pred_proba = lgbm_smote_enn1.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote_enn2 = SMOTEENN(sampling_strategy=0.75)
X_train_smote_enn2,y_train_smote_enn2 = smote_enn2.fit_resample(X_train,y_train)

lgbm_smote_enn2 = LGBMClassifier(random_state=2021)
lgbm_smote_enn2.fit(X_train_smote_enn2,y_train_smote_enn2)
pred = lgbm_smote_enn2.predict(X_test)
```

```
pred_proba = lgbm_smote_enn2.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote_enn3 = SMOTEENN(sampling_strategy=0.8)
X_train_smote_enn3,y_train_smote_enn3 = smote_enn3.fit_resample(X_train,y_train)

lgbm_smote_enn3 = LGBMClassifier(random_state=2021)
lgbm_smote_enn3.fit(X_train_smote_enn3,y_train_smote_enn3)
pred = lgbm_smote_enn3.predict(X_test)
pred_proba = lgbm_smote_enn3.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote_enn4 =  SMOTEENN(sampling_strategy=0.85)
X_train_smote_enn4,y_train_smote_enn4 = smote_enn4.fit_resample(X_train,y_train)

lgbm_smote_enn4 = LGBMClassifier(random_state=2021)
lgbm_smote_enn4.fit(X_train_smote_enn4,y_train_smote_enn4)
pred = lgbm_smote_enn4.predict(X_test)
pred_proba = lgbm_smote_enn4.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote_enn5 =  SMOTEENN(sampling_strategy=0.9)
X_train_smote_enn5,y_train_smote_enn5 = smote_enn5.fit_resample(X_train,y_train)

lgbm_smote_enn5 = LGBMClassifier(random_state=2021)
lgbm_smote_enn5.fit(X_train_smote_enn5,y_train_smote_enn5)
pred = lgbm_smote_enn5.predict(X_test)
pred_proba = lgbm_smote_enn5.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote_enn6 =  SMOTEENN(sampling_strategy=0.95)
X_train_smote_enn6,y_train_smote_enn6 = smote_enn6.fit_resample(X_train,y_train)

lgbm_smote_enn6 = LGBMClassifier(random_state=2021)
lgbm_smote_enn6.fit(X_train_smote_enn6,y_train_smote_enn6)
pred = lgbm_smote_enn6.predict(X_test)
pred_proba = lgbm_smote_enn6.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote_enn7 = SMOTEENN()
X_train_smote_enn7,y_train_smote_enn7 = smote_enn7.fit_resample(X_train,y_train)

lgbm_smote_enn7 = LGBMClassifier(random_state=2021)
lgbm_smote_enn7.fit(X_train_smote_enn7,y_train_smote_enn7)
pred = lgbm_smote_enn7.predict(X_test)
pred_proba = lgbm_smote_enn7.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
```

```
[[1904   76]
 [  26   40]]
정확도: 0.95 정밀도: 0.345 재현율: 0.606 AUC: 0.926 F1: 0.44 F2: 0.526 Balanced_Accuracy:
0.784 G-Mean: 0.763
 matthews_corrcoef: 0.434


[[1904   76]
 [  26   40]]
정확도: 0.95 정밀도: 0.345 재현율: 0.606 AUC: 0.925 F1: 0.44 F2: 0.526 Balanced_Accuracy:
0.784 G-Mean: 0.763
```

```
 matthews_corrcoef: 0.434


[[1900   80]
 [  24   42]]
정확도: 0.949 정밀도: 0.344 재현율: 0.636 AUC: 0.926 F1: 0.447 F2: 0.544 Balanced_Accurac
y: 0.798 G-Mean: 0.781
 matthews_corrcoef: 0.445


[[1889   91]
 [  23   43]]
정확도: 0.944 정밀도: 0.321 재현율: 0.652 AUC: 0.927 F1: 0.43 F2: 0.54 Balanced_Accuracy:
0.803 G-Mean: 0.788
 matthews_corrcoef: 0.432


[[1894   86]
 [  22   44]]
정확도: 0.947 정밀도: 0.338 재현율: 0.667 AUC: 0.927 F1: 0.449 F2: 0.558 Balanced_Accurac
y: 0.812 G-Mean: 0.799
 matthews_corrcoef: 0.451


[[1892   88]
 [  23   43]]
정확도: 0.946 정밀도: 0.328 재현율: 0.652 AUC: 0.928 F1: 0.437 F2: 0.544 Balanced_Accurac
y: 0.804 G-Mean: 0.789
 matthews_corrcoef: 0.438


[[1893   87]
 [  25   41]]
정확도: 0.945 정밀도: 0.32 재현율: 0.621 AUC: 0.926 F1: 0.423 F2: 0.523 Balanced_Accurac
y: 0.789 G-Mean: 0.771
 matthews_corrcoef: 0.421
```

SMOTE ENN XGBoost

In [51]:

```
smote_enn1 = SMOTEENN(sampling_strategy=0.7)
X_train_smote_enn1,y_train_smote_enn1 = smote_enn1.fit_resample(X_train,y_train)

xgb_smote_enn1 = XGBClassifier(random_state=2021)
xgb_smote_enn1.fit(X_train_smote_enn1,y_train_smote_enn1)
pred = xgb_smote_enn1.predict(X_test)
pred_proba = xgb_smote_enn1.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote_enn2 = SMOTEENN(sampling_strategy=0.75)
X_train_smote_enn2,y_train_smote_enn2 = smote_enn2.fit_resample(X_train,y_train)

xgb_smote_enn2 = XGBClassifier(random_state=2021)
xgb_smote_enn2.fit(X_train_smote_enn2,y_train_smote_enn2)
pred = xgb_smote_enn2.predict(X_test)
pred_proba = xgb_smote_enn2.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote_enn3 = SMOTEENN(sampling_strategy=0.8)
X_train_smote_enn3,y_train_smote_enn3 = smote_enn3.fit_resample(X_train,y_train)

xgb_smote_enn3 = XGBClassifier(random_state=2021)
xgb_smote_enn3.fit(X_train_smote_enn3,y_train_smote_enn3)
pred = xgb_smote_enn3.predict(X_test)
```

```python
pred_proba = xgb_smote_enn3.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote_enn4 =  SMOTEENN(sampling_strategy=0.85)
X_train_smote_enn4,y_train_smote_enn4 = smote_enn4.fit_resample(X_train,y_train)

xgb_smote_enn4 = XGBClassifier(random_state=2021)
xgb_smote_enn4.fit(X_train_smote_enn4,y_train_smote_enn4)
pred = xgb_smote_enn4.predict(X_test)
pred_proba = xgb_smote_enn4.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote_enn5 =  SMOTEENN(sampling_strategy=0.9)
X_train_smote_enn5,y_train_smote_enn5 = smote_enn5.fit_resample(X_train,y_train)

xgb_smote_enn5 = XGBClassifier(random_state=2021)
xgb_smote_enn5.fit(X_train_smote_enn5,y_train_smote_enn5)
pred = xgb_smote_enn5.predict(X_test)
pred_proba = xgb_smote_enn5.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote_enn6 =  SMOTEENN(sampling_strategy=0.95)
X_train_smote_enn6,y_train_smote_enn6 = smote_enn6.fit_resample(X_train,y_train)

xgb_smote_enn6 = XGBClassifier(random_state=2021)
xgb_smote_enn6.fit(X_train_smote_enn6,y_train_smote_enn6)
pred = xgb_smote_enn6.predict(X_test)
pred_proba = xgb_smote_enn6.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smote_enn7 = SMOTEENN()
X_train_smote_enn7,y_train_smote_enn7 = smote_enn7.fit_resample(X_train,y_train)

xgb_smote_enn7 = XGBClassifier(random_state=2021)
xgb_smote_enn7.fit(X_train_smote_enn7,y_train_smote_enn7)
pred = xgb_smote_enn7.predict(X_test)
pred_proba = xgb_smote_enn7.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
```

```
[17:25:41] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1905   75]
 [  22   44]]
정확도: 0.953 정밀도: 0.37 재현율: 0.667 AUC: 0.926 F1: 0.476 F2: 0.574 Balanced_Accurac
y: 0.814 G-Mean: 0.801
 matthews_corrcoef: 0.475


[17:25:45] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1897   83]
 [  24   42]]
정확도: 0.948 정밀도: 0.336 재현율: 0.636 AUC: 0.925 F1: 0.44 F2: 0.54 Balanced_Accuracy:
0.797 G-Mean: 0.781
 matthews_corrcoef: 0.439
```

```
[17:25:48] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1897   83]
 [  25   41]]
정확도: 0.947 정밀도: 0.331 재현율: 0.621 AUC: 0.929 F1: 0.432 F2: 0.528 Balanced_Accurac
y: 0.79 G-Mean: 0.771
 matthews_corrcoef: 0.429


[17:25:52] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1890   90]
 [  21   45]]
정확도: 0.946 정밀도: 0.333 재현율: 0.682 AUC: 0.92 F1: 0.448 F2: 0.564 Balanced_Accurac
y: 0.818 G-Mean: 0.807
 matthews_corrcoef: 0.453


[17:25:55] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1889   91]
 [  24   42]]
정확도: 0.944 정밀도: 0.316 재현율: 0.636 AUC: 0.934 F1: 0.422 F2: 0.529 Balanced_Accurac
y: 0.795 G-Mean: 0.779
 matthews_corrcoef: 0.423


[17:25:59] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1889   91]
 [  21   45]]
정확도: 0.945 정밀도: 0.331 재현율: 0.682 AUC: 0.928 F1: 0.446 F2: 0.562 Balanced_Accurac
y: 0.818 G-Mean: 0.807
 matthews_corrcoef: 0.451


[17:26:03] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1886   94]
 [  23   43]]
정확도: 0.943 정밀도: 0.314 재현율: 0.652 AUC: 0.924 F1: 0.424 F2: 0.536 Balanced_Accurac
y: 0.802 G-Mean: 0.788
 matthews_corrcoef: 0.427
```

Resampling 7.) SMOTE Tomek

In [52]:

```python
from imblearn.combine import SMOTETomek
```

SMOTE_TOMEK GBM

In [53]:

```python
smotetomek1 = SMOTETomek(sampling_strategy=0.7)
X_train_smotetomek1,y_train_smotetomek1 = smotetomek1.fit_resample(X_train,y_train)
```

```python
gbm_smotetomek1 = GradientBoostingClassifier(random_state=2021)
gbm_smotetomek1.fit(X_train_smotetomek1,y_train_smotetomek1)
pred = gbm_smotetomek1.predict(X_test)
pred_proba = gbm_smotetomek1.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smotetomek2 = SMOTETomek(sampling_strategy=0.75)
X_train_smotetomek2,y_train_smotetomek2 = smotetomek2.fit_resample(X_train,y_train)

gbm_smotetomek2 = GradientBoostingClassifier(random_state=2021)
gbm_smotetomek2.fit(X_train_smotetomek2,y_train_smotetomek2)
pred = gbm_smotetomek2.predict(X_test)
pred_proba = gbm_smotetomek2.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smotetomek3 = SMOTETomek(sampling_strategy=0.8)
X_train_smotetomek3,y_train_smotetomek3 = smotetomek3.fit_resample(X_train,y_train)

gbm_smotetomek3 = GradientBoostingClassifier(random_state=2021)
gbm_smotetomek3.fit(X_train_smotetomek3,y_train_smotetomek3)
pred = gbm_smotetomek3.predict(X_test)
pred_proba = gbm_smotetomek3.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smotetomek4 =  SMOTETomek(sampling_strategy=0.85)
X_train_smotetomek4,y_train_smotetomek4 = smotetomek4.fit_resample(X_train,y_train)

gbm_smotetomek4 = GradientBoostingClassifier(random_state=2021)
gbm_smotetomek4.fit(X_train_smotetomek4,y_train_smotetomek4)
pred = gbm_smotetomek4.predict(X_test)
pred_proba = gbm_smotetomek4.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smotetomek5 =  SMOTETomek(sampling_strategy=0.9)
X_train_smotetomek5,y_train_smotetomek5 = smotetomek5.fit_resample(X_train,y_train)

gbm_smotetomek5 = GradientBoostingClassifier(random_state=2021)
gbm_smotetomek5.fit(X_train_smotetomek5,y_train_smotetomek5)
pred = gbm_smotetomek5.predict(X_test)
pred_proba = gbm_smotetomek5.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smotetomek6 =  SMOTETomek(sampling_strategy=0.95)
X_train_smotetomek6,y_train_smotetomek6 = smotetomek6.fit_resample(X_train,y_train)

gbm_smotetomek6 = GradientBoostingClassifier(random_state=2021)
gbm_smotetomek6.fit(X_train_smotetomek6,y_train_smotetomek6)
pred = gbm_smotetomek6.predict(X_test)
pred_proba = gbm_smotetomek6.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smotetomek7 = SMOTETomek()
X_train_smotetomek7,y_train_smotetomek7 = smotetomek7.fit_resample(X_train,y_train)

gbm_smotetomek7 = GradientBoostingClassifier(random_state=2021)
gbm_smotetomek7.fit(X_train_smotetomek7,y_train_smotetomek7)
pred = gbm_smotetomek7.predict(X_test)
```

```
pred_proba = gbm_smotetomek7.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
```

```
[[1863  117]
 [  19   47]]
정확도: 0.934 정밀도: 0.287 재현율: 0.712 AUC: 0.93 F1: 0.409 F2: 0.549 Balanced_Accurac
y: 0.827 G-Mean: 0.819
 matthews_corrcoef: 0.425


[[1854  126]
 [  22   44]]
정확도: 0.928 정밀도: 0.259 재현율: 0.667 AUC: 0.923 F1: 0.373 F2: 0.507 Balanced_Accurac
y: 0.802 G-Mean: 0.79
 matthews_corrcoef: 0.386


[[1859  121]
 [  19   47]]
정확도: 0.932 정밀도: 0.28 재현율: 0.712 AUC: 0.924 F1: 0.402 F2: 0.544 Balanced_Accurac
y: 0.826 G-Mean: 0.818
 matthews_corrcoef: 0.419


[[1854  126]
 [  20   46]]
정확도: 0.929 정밀도: 0.267 재현율: 0.697 AUC: 0.927 F1: 0.387 F2: 0.528 Balanced_Accurac
y: 0.817 G-Mean: 0.808
 matthews_corrcoef: 0.403


[[1855  125]
 [  19   47]]
정확도: 0.93 정밀도: 0.273 재현율: 0.712 AUC: 0.925 F1: 0.395 F2: 0.539 Balanced_Accurac
y: 0.824 G-Mean: 0.817
 matthews_corrcoef: 0.413


[[1845  135]
 [  16   50]]
정확도: 0.926 정밀도: 0.27 재현율: 0.758 AUC: 0.924 F1: 0.398 F2: 0.557 Balanced_Accurac
y: 0.845 G-Mean: 0.84
 matthews_corrcoef: 0.425


[[1840  140]
 [  18   48]]
정확도: 0.923 정밀도: 0.255 재현율: 0.727 AUC: 0.925 F1: 0.378 F2: 0.531 Balanced_Accurac
y: 0.828 G-Mean: 0.822
 matthews_corrcoef: 0.402
```

SMOTE_TOMEK Light GBM

In [54]:

```
smotetomek1 = SMOTETomek(sampling_strategy=0.7)
X_train_smotetomek1,y_train_smotetomek1 = smotetomek1.fit_resample(X_train,y_train)

lgbm_smotetomek1 = LGBMClassifier(random_state=2021)
lgbm_smotetomek1.fit(X_train_smotetomek1,y_train_smotetomek1)
pred = lgbm_smotetomek1.predict(X_test)
pred_proba = lgbm_smotetomek1.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smotetomek2 = SMOTETomek(sampling_strategy=0.75)
X_train_smotetomek2,y_train_smotetomek2 = smotetomek2.fit_resample(X_train,y_train)
```

```python
lgbm_smotetomek2 = LGBMClassifier(random_state=2021)
lgbm_smotetomek2.fit(X_train_smotetomek2,y_train_smotetomek2)
pred = lgbm_smotetomek2.predict(X_test)
pred_proba = lgbm_smotetomek2.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smotetomek3 = SMOTETomek(sampling_strategy=0.8)
X_train_smotetomek3,y_train_smotetomek3 = smotetomek3.fit_resample(X_train,y_train)

lgbm_smotetomek3 = LGBMClassifier(random_state=2021)
lgbm_smotetomek3.fit(X_train_smotetomek3,y_train_smotetomek3)
pred = lgbm_smotetomek3.predict(X_test)
pred_proba = lgbm_smotetomek3.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smotetomek4 =  SMOTETomek(sampling_strategy=0.85)
X_train_smotetomek4,y_train_smotetomek4 = smotetomek4.fit_resample(X_train,y_train)

lgbm_smotetomek4 = LGBMClassifier(random_state=2021)
lgbm_smotetomek4.fit(X_train_smotetomek4,y_train_smotetomek4)
pred = lgbm_smotetomek4.predict(X_test)
pred_proba = lgbm_smotetomek4.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smotetomek5 =  SMOTETomek(sampling_strategy=0.9)
X_train_smotetomek5,y_train_smotetomek5 = smotetomek5.fit_resample(X_train,y_train)

lgbm_smotetomek5 = LGBMClassifier(random_state=2021)
lgbm_smotetomek5.fit(X_train_smotetomek5,y_train_smotetomek5)
pred = lgbm_smotetomek5.predict(X_test)
pred_proba = lgbm_smotetomek5.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smotetomek6 =  SMOTETomek(sampling_strategy=0.95)
X_train_smotetomek6,y_train_smotetomek6 = smotetomek6.fit_resample(X_train,y_train)

lgbm_smotetomek6 = LGBMClassifier(random_state=2021)
lgbm_smotetomek6.fit(X_train_smotetomek6,y_train_smotetomek6)
pred = lgbm_smotetomek6.predict(X_test)
pred_proba = lgbm_smotetomek6.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smotetomek7 = SMOTETomek()
X_train_smotetomek7,y_train_smotetomek7 = smotetomek7.fit_resample(X_train,y_train)

lgbm_smotetomek7 = LGBMClassifier(random_state=2021)
lgbm_smotetomek7.fit(X_train_smotetomek7,y_train_smotetomek7)
pred = lgbm_smotetomek7.predict(X_test)
pred_proba = lgbm_smotetomek7.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
```

```
[[1919   61]
 [  26   40]]
정확도: 0.957 정밀도: 0.396 재현율: 0.606 AUC: 0.933 F1: 0.479 F2: 0.548 Balanced_Accurac
y: 0.788 G-Mean: 0.766
 matthews_corrcoef: 0.469
```

```
[[1920   60]
 [  26   40]]
정확도: 0.958 정밀도: 0.4 재현율: 0.606 AUC: 0.933 F1: 0.482 F2: 0.549 Balanced_Accuracy:
0.788 G-Mean: 0.767
 matthews_corrcoef: 0.472


[[1919   61]
 [  25   41]]
정확도: 0.958 정밀도: 0.402 재현율: 0.621 AUC: 0.929 F1: 0.488 F2: 0.56 Balanced_Accurac
y: 0.795 G-Mean: 0.776
 matthews_corrcoef: 0.479


[[1919   61]
 [  27   39]]
정확도: 0.957 정밀도: 0.39 재현율: 0.591 AUC: 0.936 F1: 0.47 F2: 0.536 Balanced_Accuracy:
0.78 G-Mean: 0.757
 matthews_corrcoef: 0.459


[[1918   62]
 [  26   40]]
정확도: 0.957 정밀도: 0.392 재현율: 0.606 AUC: 0.931 F1: 0.476 F2: 0.546 Balanced_Accurac
y: 0.787 G-Mean: 0.766
 matthews_corrcoef: 0.467


[[1914   66]
 [  27   39]]
정확도: 0.955 정밀도: 0.371 재현율: 0.591 AUC: 0.929 F1: 0.456 F2: 0.528 Balanced_Accurac
y: 0.779 G-Mean: 0.756
 matthews_corrcoef: 0.446


[[1911   69]
 [  24   42]]
정확도: 0.955 정밀도: 0.378 재현율: 0.636 AUC: 0.928 F1: 0.475 F2: 0.56 Balanced_Accurac
y: 0.801 G-Mean: 0.784
 matthews_corrcoef: 0.469
```

SMOTE_TOMEK XGBoost

In [55]:

```
smotetomek1 = SMOTETomek(sampling_strategy=0.7)
X_train_smotetomek1,y_train_smotetomek1 = smotetomek1.fit_resample(X_train,y_train)

xgb_smotetomek1 = XGBClassifier(random_state=2021)
xgb_smotetomek1.fit(X_train_smotetomek1,y_train_smotetomek1)
pred = xgb_smotetomek1.predict(X_test)
pred_proba = xgb_smotetomek1.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smotetomek2 = SMOTETomek(sampling_strategy=0.75)
X_train_smotetomek2,y_train_smotetomek2 = smotetomek2.fit_resample(X_train,y_train)

xgb_smotetomek2 = XGBClassifier(random_state=2021)
xgb_smotetomek2.fit(X_train_smotetomek2,y_train_smotetomek2)
pred = xgb_smotetomek2.predict(X_test)
pred_proba = xgb_smotetomek2.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smotetomek3 = SMOTETomek(sampling_strategy=0.8)
```

```
X_train_smotetomek3,y_train_smotetomek3 = smotetomek3.fit_resample(X_train,y_train)

xgb_smotetomek3 = XGBClassifier(random_state=2021)
xgb_smotetomek3.fit(X_train_smotetomek3,y_train_smotetomek3)
pred = xgb_smotetomek3.predict(X_test)
pred_proba = xgb_smotetomek3.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smotetomek4 =  SMOTETomek(sampling_strategy=0.85)
X_train_smotetomek4,y_train_smotetomek4 = smotetomek4.fit_resample(X_train,y_train)

xgb_smotetomek4 = XGBClassifier(random_state=2021)
xgb_smotetomek4.fit(X_train_smotetomek4,y_train_smotetomek4)
pred = xgb_smotetomek4.predict(X_test)
pred_proba = xgb_smotetomek4.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smotetomek5 =  SMOTETomek(sampling_strategy=0.9)
X_train_smotetomek5,y_train_smotetomek5 = smotetomek5.fit_resample(X_train,y_train)

xgb_smotetomek5 = XGBClassifier(random_state=2021)
xgb_smotetomek5.fit(X_train_smotetomek5,y_train_smotetomek5)
pred = xgb_smotetomek5.predict(X_test)
pred_proba = xgb_smotetomek5.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smotetomek6 =  SMOTETomek(sampling_strategy=0.95)
X_train_smotetomek6,y_train_smotetomek6 = smotetomek6.fit_resample(X_train,y_train)

xgb_smotetomek6 = XGBClassifier(random_state=2021)
xgb_smotetomek6.fit(X_train_smotetomek6,y_train_smotetomek6)
pred = xgb_smotetomek6.predict(X_test)
pred_proba = xgb_smotetomek6.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
print('\n')

smotetomek7 = SMOTETomek()
X_train_smotetomek7,y_train_smotetomek7 = smotetomek7.fit_resample(X_train,y_train)

xgb_smotetomek7 = XGBClassifier(random_state=2021)
xgb_smotetomek7.fit(X_train_smotetomek7,y_train_smotetomek7)
pred = xgb_smotetomek7.predict(X_test)
pred_proba = xgb_smotetomek7.predict_proba(X_test)[:,1]
get_clf_eval(y_test,pred,pred_proba)
```

```
[17:27:38] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1921   59]
 [  25   41]]
정확도: 0.959 정밀도: 0.41 재현율: 0.621 AUC: 0.929 F1: 0.494 F2: 0.563 Balanced_Accurac
y: 0.796 G-Mean: 0.776
 matthews_corrcoef: 0.485


[17:27:42] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1918   62]
```

```
 [  29   37]]
정확도: 0.956 정밀도: 0.374 재현율: 0.561 AUC: 0.924 F1: 0.448 F2: 0.51 Balanced_Accurac
y: 0.765 G-Mean: 0.737
 matthews_corrcoef: 0.436


[17:27:46] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1924   56]
 [  27   39]]
정확도: 0.959 정밀도: 0.411 재현율: 0.591 AUC: 0.925 F1: 0.484 F2: 0.543 Balanced_Accurac
y: 0.781 G-Mean: 0.758
 matthews_corrcoef: 0.472


[17:27:50] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1921   59]
 [  25   41]]
정확도: 0.959 정밀도: 0.41 재현율: 0.621 AUC: 0.929 F1: 0.494 F2: 0.563 Balanced_Accurac
y: 0.796 G-Mean: 0.776
 matthews_corrcoef: 0.485


[17:27:54] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1919   61]
 [  23   43]]
정확도: 0.959 정밀도: 0.413 재현율: 0.652 AUC: 0.927 F1: 0.506 F2: 0.584 Balanced_Accurac
y: 0.81 G-Mean: 0.795
 matthews_corrcoef: 0.499


[17:27:58] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1916   64]
 [  27   39]]
정확도: 0.956 정밀도: 0.379 재현율: 0.591 AUC: 0.934 F1: 0.462 F2: 0.531 Balanced_Accurac
y: 0.779 G-Mean: 0.756
 matthews_corrcoef: 0.451


[17:28:03] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/lea
rner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
[[1922   58]
 [  27   39]]
정확도: 0.958 정밀도: 0.402 재현율: 0.591 AUC: 0.929 F1: 0.479 F2: 0.54 Balanced_Accurac
y: 0.781 G-Mean: 0.757
 matthews_corrcoef: 0.467
```

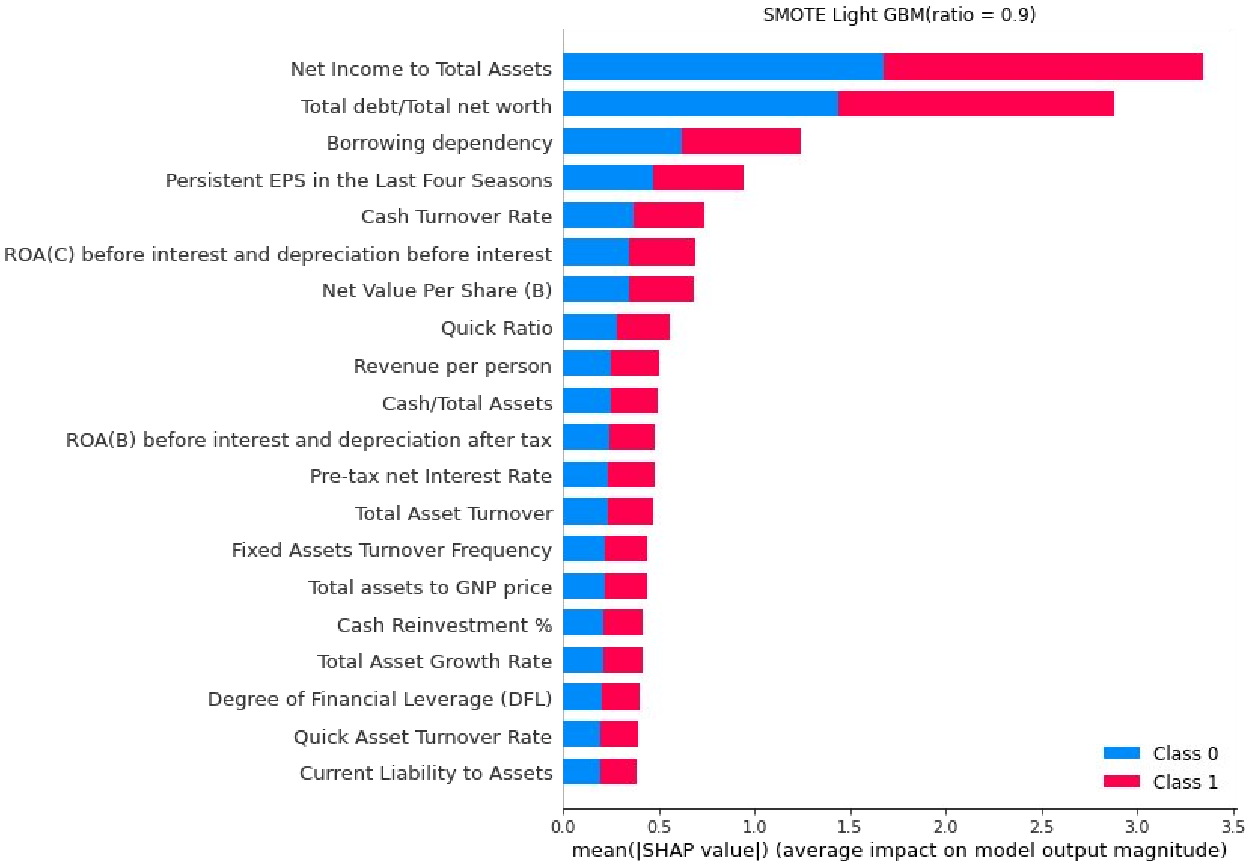
종합 최고 성능 모형 Top10개의 변수 중요도

In [77]:

```
import shap
```

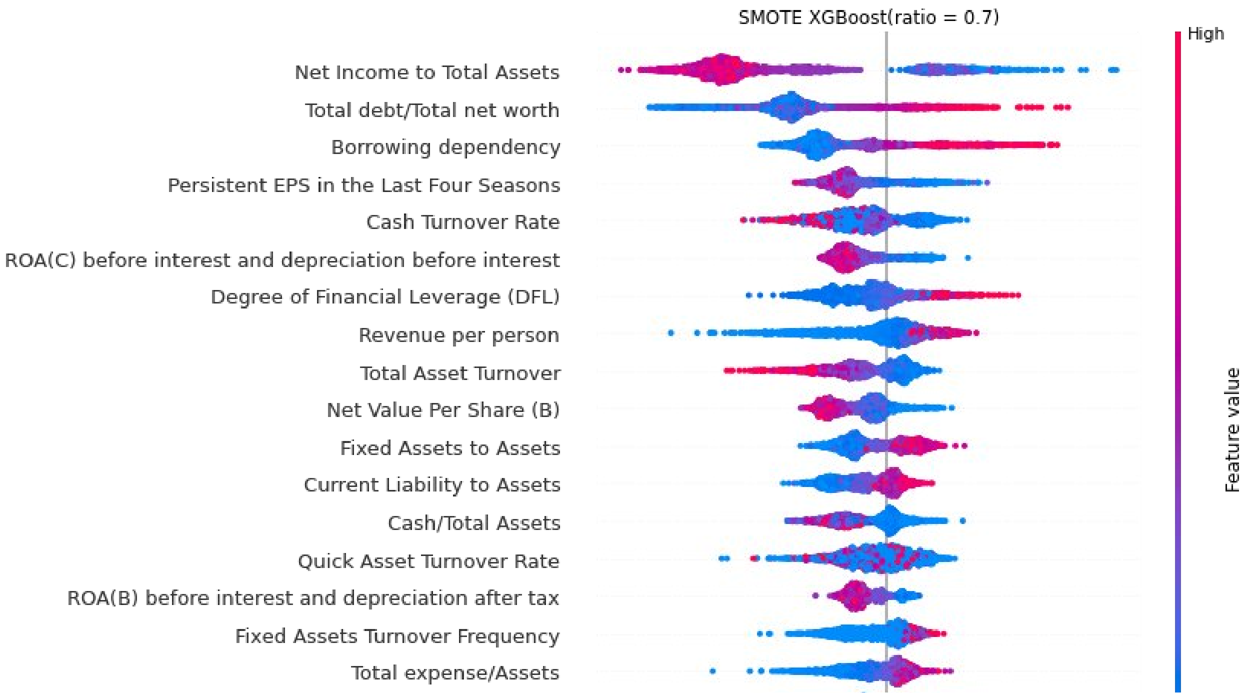
In [91]:

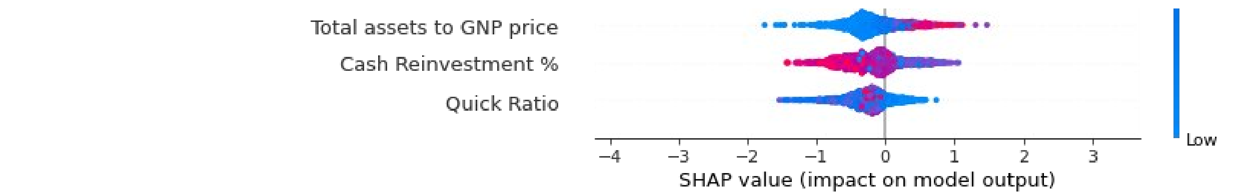
```
plt.title("SMOTE Light GBM(ratio = 0.9)")
explainer = shap.TreeExplainer(lgbm_smote_5)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
plt.show()
```



In [79]:

```
plt.title("SMOTE XGBoost(ratio = 0.7)")
explainer = shap.TreeExplainer(xgb_smote_1)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
plt.show()
```



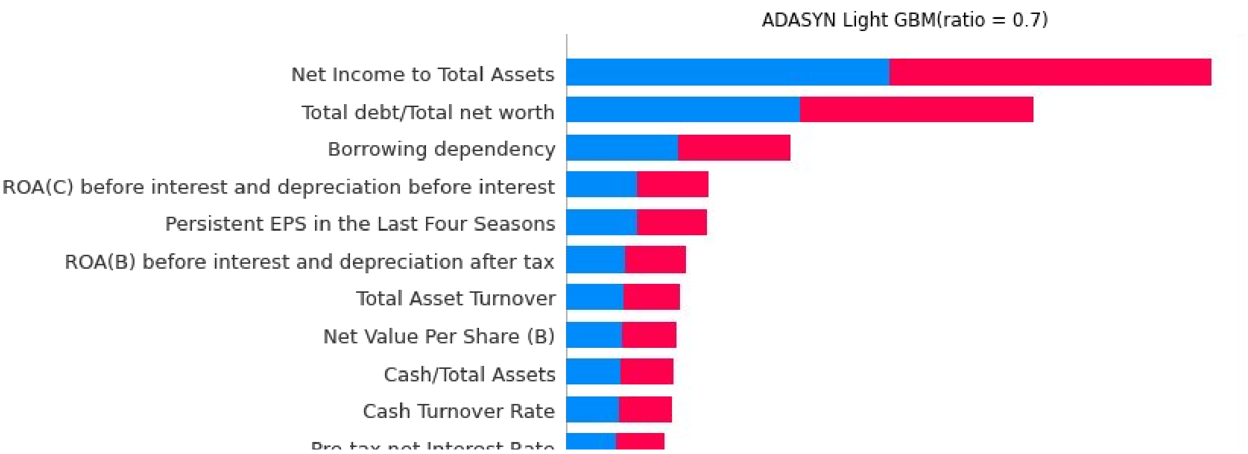


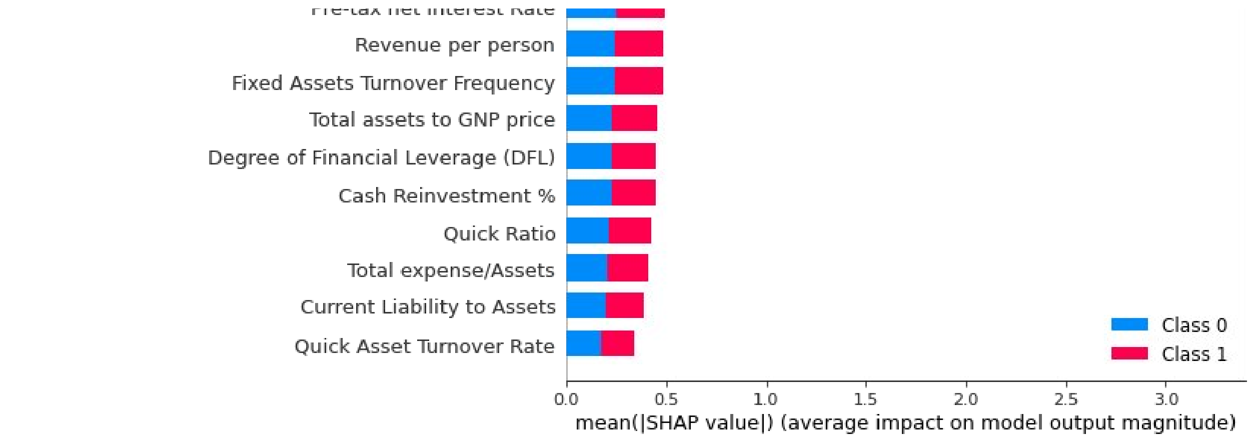
In [80]:

```
plt.title("SMOTE XGBoost(ratio = 1.0)")
explainer = shap.TreeExplainer(xgb_smote_7)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
plt.show()
```

In [81]:

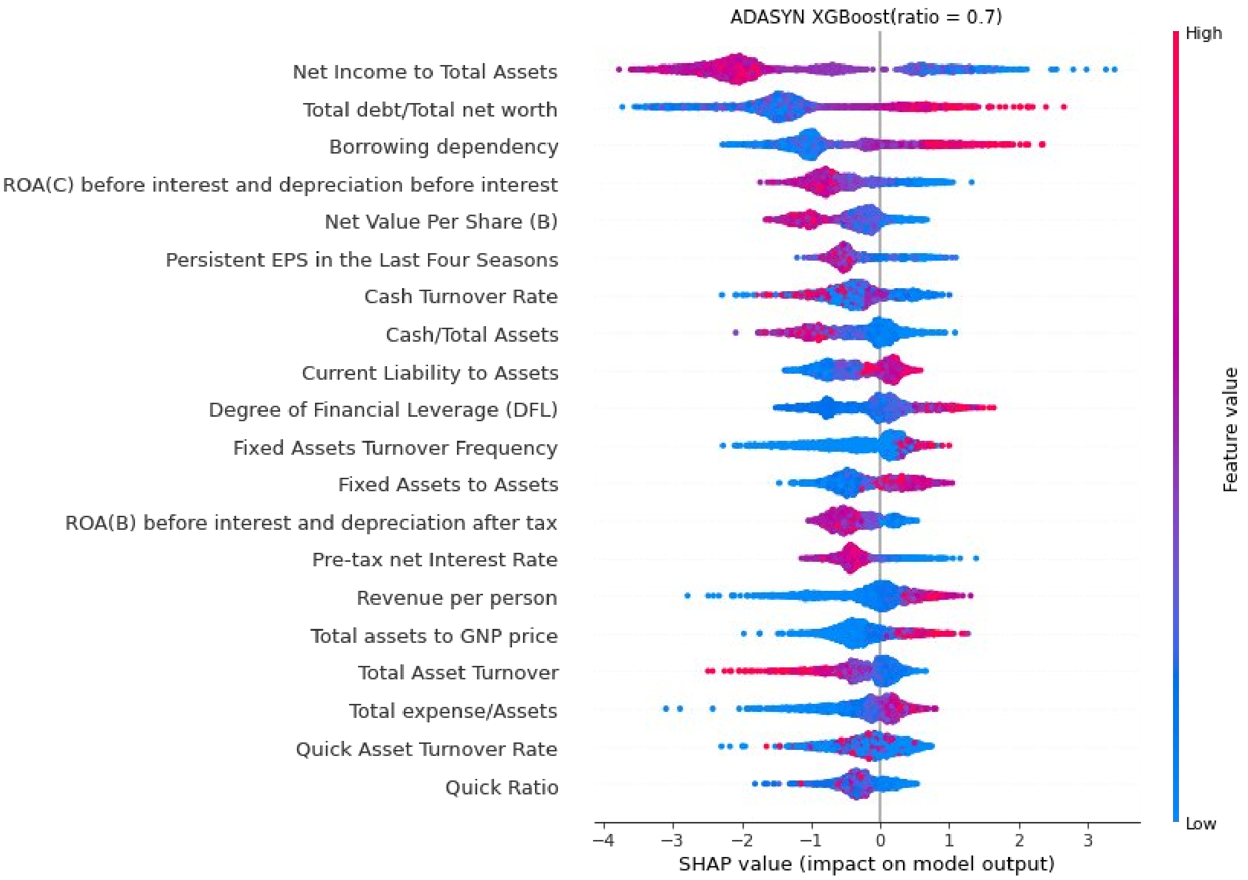
```
plt.title("ADASYN Light GBM(ratio = 0.7)")
explainer = shap.TreeExplainer(lgbm_ads1)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
plt.show()
```





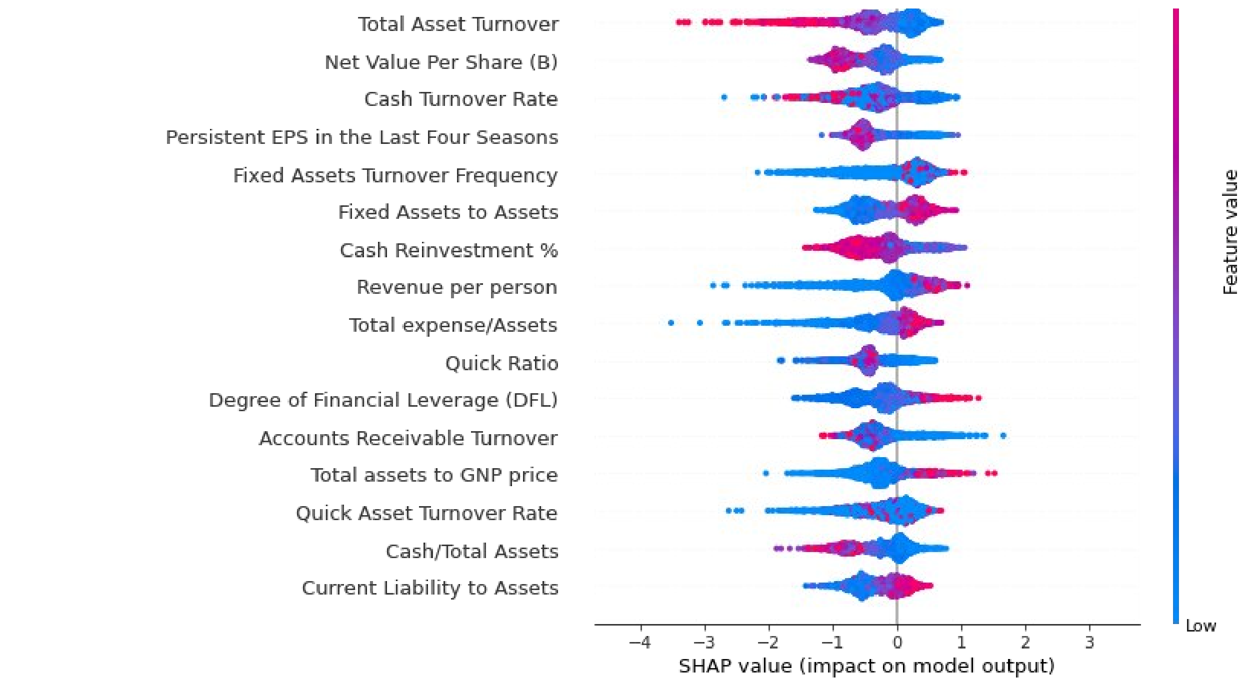
In [82]:

```
plt.title("ADASYN XGBoost(ratio = 0.7)")
explainer = shap.TreeExplainer(xgb_ads1)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
plt.show()
```



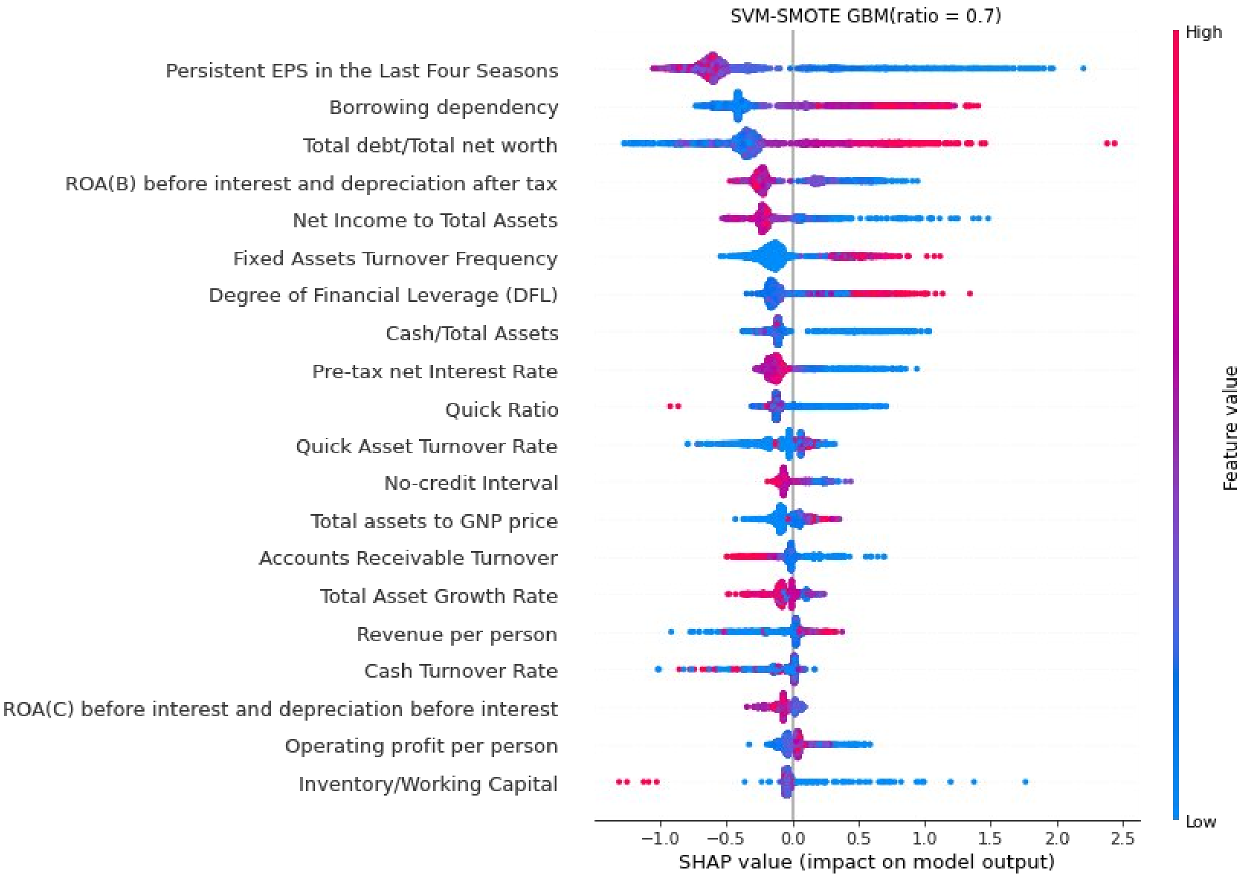
In [83]:

```
plt.title("ADASYN XGBoost(ratio = 1.0)")
explainer = shap.TreeExplainer(xgb_ads7)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
plt.show()
```



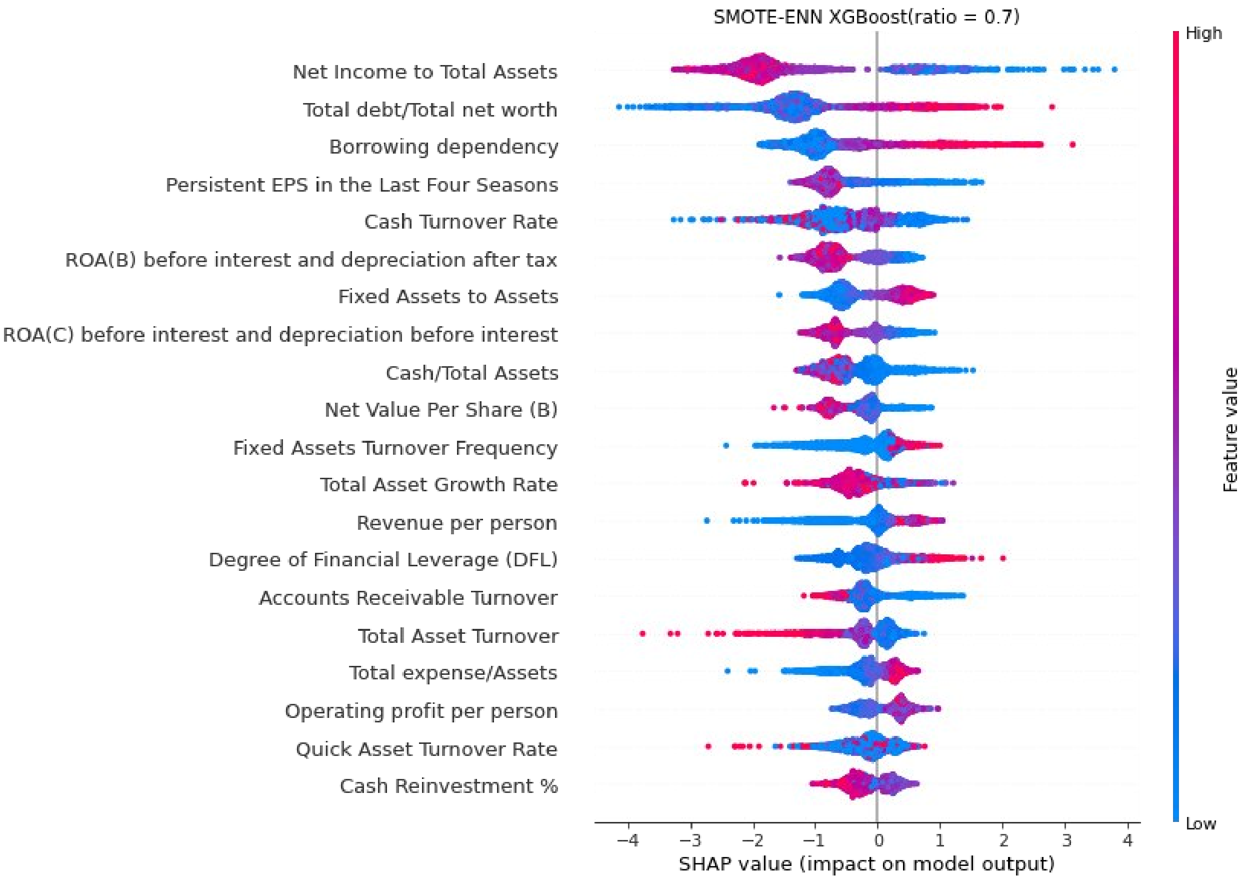
In [84]:

```
plt.title("SVM-SMOTE GBM(ratio = 0.7)")
explainer = shap.TreeExplainer(gbm_svm_smote1)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
plt.show()
```



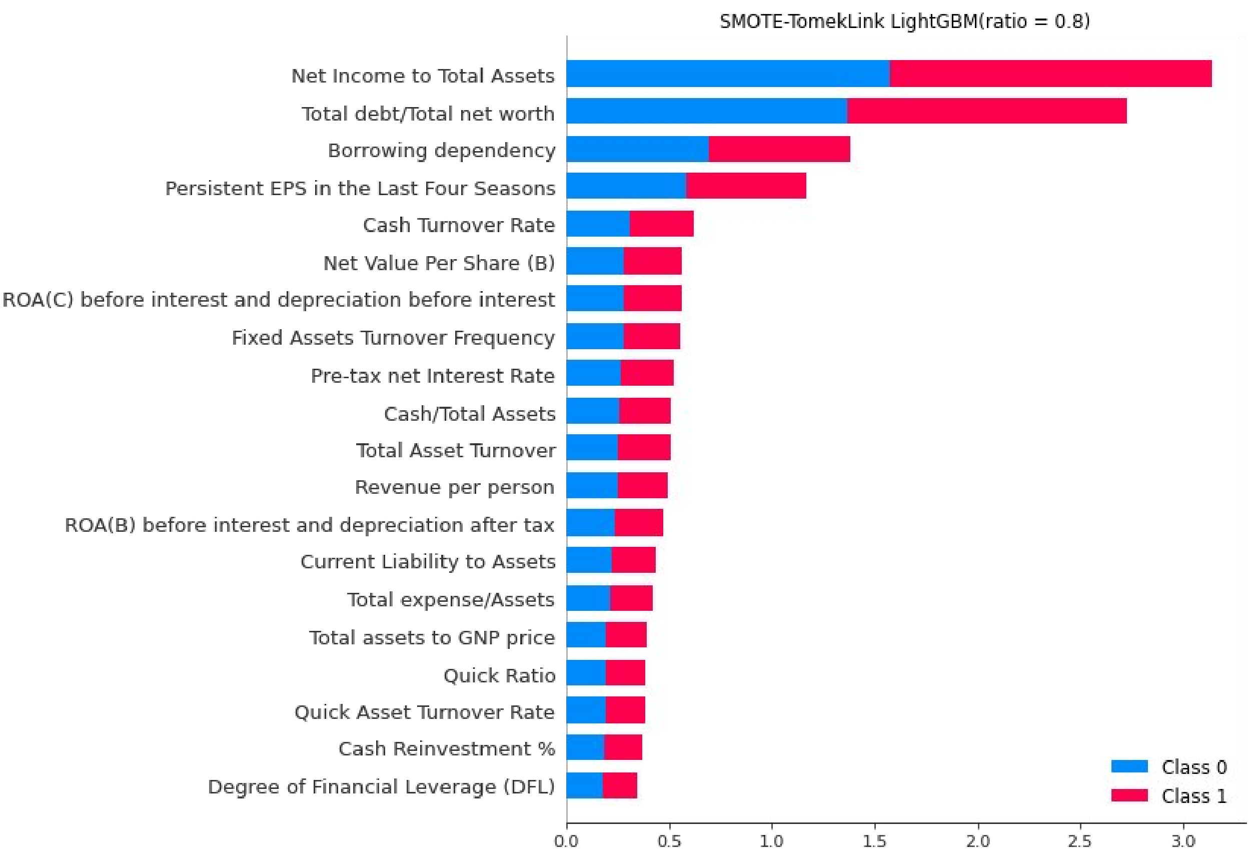
In [85]:

```
plt.title("SMOTE-ENN XGBoost(ratio = 0.7)")
explainer = shap.TreeExplainer(xgb_smote_enn1)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
plt.show()
```



In [92]:

```
plt.title("SMOTE-TomekLink LightGBM(ratio = 0.8)")
explainer = shap.TreeExplainer(lgbm_smotetomek3)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
plt.show()
```



mean(|SHAP value|) (average impact on model output magnitude)

In [87]:

```
plt.title("SMOTE-TomekLink  XGBoost(ratio = 0.9)")
explainer = shap.TreeExplainer(xgb_smotetomek5)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
plt.show()
```