In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dt
import time
import os

import shap
import lime
from lime import lime_tabular

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, RandomizedSearchCV, GridSearchCV
from sklearn.preprocessing import MinMaxScaler
import statsmodels.api as sm
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE
from statsmodels.stats.outliers_influence import variance_inflation_factor

from sklearn import metrics
from sklearn.metrics import confusion_matrix

from sklearn.metrics import precision_score, recall_score
from sklearn.metrics import precision_recall_curve

from sklearn.cluster import KMeans

import missingno as msno

from fancyimpute import IterativeImputer as MICE
from sklearn.impute import IterativeImputer
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC

import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.optimizers import Adam


from sklearn.cluster import DBSCAN
from imblearn.over_sampling import SMOTE
from sklearn.neighbors import NearestNeighbors
from collections import Counter

from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import numpy as np

from imblearn.over_sampling import KMeansSMOTE
from sklearn.mixture import GaussianMixture


from xgboost import XGBClassifier
from rgf.sklearn import RGFClassifier  # Regularized Greedy Forest
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense


from sklearn.metrics import accuracy_score, confusion_matrix, f1_score, roc_auc_score, roc_curve
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

from joblib import dump, load
import logging
```

In [ ]:
```
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
```

```
def split_dataset(dataset, target_column, test_size=0.2):
    """
    Split dataset into training and testing sets.
    """
    X = dataset.drop(columns=[target_column])
    y = dataset[target_column]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=42, stratify=y)

    logging.info("Dataset has been split and returned")
    return X_train, X_test, y_train, y_test

def train_ann(X_train, y_train):
    """
    Train an Artificial Neural Network (ANN) on the training data.
    """
    start_time = time.time()
    model = Sequential([
        Input(shape=(X_train.shape[1],)),
        Dense(12, activation='relu'),
        Dense(8, activation='relu'),
        Dense(1, activation='sigmoid')
    ])
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    model.fit(X_train, y_train, epochs=150, batch_size=10, verbose=0)
    end_time = time.time()

    logging.info(f"ANN has been trained in {end_time - start_time:.2f} seconds")
    return model

def train_models(X_train, y_train):
    """
    Train multiple models on the training data.
    """
    models = {}
    param_grids = {
        'RandomForest': {
            'n_estimators': [100, 200, 300],
            'max_depth': [None, 10, 20],
            'min_samples_split': [2, 5]
        },
        'XGBoost': {
            'n_estimators': [100, 200, 300],
            'max_depth': [3, 6],
            'learning_rate': [0.01, 0.1]
        },
        'SVM': {
            'C': [0.1, 1, 10],
            'kernel': ['linear', 'rbf']
        },
        'LogisticRegression': {
            'C': [0.1, 1, 10],
            'penalty': ['l2']
        },
        'GradientBoosting': {
            'n_estimators': [100, 200, 300],
            'learning_rate': [0.01, 0.1],
            'max_depth': [3, 5, 7]
        },
        'KNN': {
            'n_neighbors': [3, 5, 7],
            'weights': ['uniform', 'distance']
        }
    }

    models['ANN'] = train_ann(X_train, y_train)

    for model_name, param_grid in param_grids.items():
        start_time = time.time()
        try:
            if model_name == 'RandomForest':
                model = GridSearchCV(RandomForestClassifier(), param_grid, cv=5)
            elif model_name == 'XGBoost':
                model = GridSearchCV(XGBClassifier(), param_grid, cv=5)
            elif model_name == 'SVM':
                model = GridSearchCV(SVC(probability=True), param_grid, cv=5)
```

```
            elif model_name == 'LogisticRegression':
                model = GridSearchCV(LogisticRegression(), param_grid, cv=5)
            elif model_name == 'GradientBoosting':
                model = GridSearchCV(GradientBoostingClassifier(), param_grid, cv=5)
            elif model_name == 'KNN':
                model = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5)

            model.fit(X_train, y_train)
            models[model_name] = model.best_estimator_
            end_time = time.time()
            logging.info(f"{model_name} has been trained in {end_time - start_time:.2f} seconds")
        except Exception as e:
            logging.error(f"Error training {model_name}: {e}")

    try:
        start_time = time.time()
        nb = GaussianNB()
        nb.fit(X_train, y_train)
        models['NaiveBayes'] = nb
        end_time = time.time()
        logging.info(f"Naive Bayes has been trained in {end_time - start_time:.2f} seconds")
    except Exception as e:
        logging.error(f"Error training Naive Bayes: {e}")

    return models

def test_models(models, X_test):
    """
    Test trained models on the test data.
    """
    start_time = time.time()
    predictions = {}
    for name, model in models.items():
        try:
            if name == 'ANN':
                predictions[name] = (model.predict(X_test) > 0.5).astype("int32")
            else:
                predictions[name] = model.predict(X_test)
        except Exception as e:
            logging.error(f"Error testing {name}: {e}")
    end_time = time.time()

    logging.info(f"Models have been tested in {end_time - start_time:.2f} seconds")
    return predictions

def evaluate_models(models, predictions, y_test, X_test):
    """
    Evaluate the performance of models.
    """
    start_time = time.time()
    metrics = {}
    for name, y_pred in predictions.items():
        try:
            accuracy = accuracy_score(y_test, y_pred)
            cm = confusion_matrix(y_test, y_pred)
            f1 = f1_score(y_test, y_pred)
            auc = roc_auc_score(y_test, models[name].predict_proba(X_test)[:, 1]) if name != 'ANN' else roc_auc_score
            metrics[name] = {
                'accuracy': accuracy,
                'confusion_matrix': cm,
                'f1_score': f1,
                'auc_roc': auc
            }
        except Exception as e:
            logging.error(f"Error evaluating {name}: {e}")
    end_time = time.time()

    logging.info(f"Models have been evaluated in {end_time - start_time:.2f} seconds")
    return metrics

def explainability_shap(models, X_test, feature_names):
    """
    Generate SHAP explanations for models.
    """
    shap.initjs()
```

```
    for name, model in models.items():
        if name == 'ANN':
            continue
        try:
            explainer = shap.TreeExplainer(model)
            shap_values = explainer.shap_values(X_test)
            shap.summary_plot(shap_values, X_test, feature_names=feature_names)
            logging.info(f"SHAP summary plot for {name} created")
        except Exception as e:
            logging.error(f"Error generating SHAP explanations for {name}: {e}")

def explainability_lime(models, X_train, X_test, feature_names):
    """
    Generate LIME explanations for models.
    """
    explainer = lime.lime_tabular.LimeTabularExplainer(X_train, feature_names=feature_names, class_names=['class1',
    for name, model in models.items():
        if name == 'ANN':
            continue
        try:
            i = np.random.randint(0, X_test.shape[0])
            exp = explainer.explain_instance(X_test[i], model.predict_proba)
            exp.show_in_notebook(show_table=True)
            logging.info(f"LIME explanation for a sample of {name} created")
        except Exception as e:
            logging.error(f"Error generating LIME explanations for {name}: {e}")

def save_models(models, directory='models'):
    """
    Save trained models to disk.
    """
    if not os.path.exists(directory):
        os.makedirs(directory)
    for name, model in models.items():
        try:
            if name == 'ANN':
                model.save(os.path.join(directory, f'{name}_model.h5'))
            else:
                dump(model, os.path.join(directory, f'{name}_model.joblib'))
            logging.info(f"{name} model saved")
        except Exception as e:
            logging.error(f"Error saving {name} model: {e}")

def load_models(directory='models'):
    """
    Load trained models from disk.
    """
    models = {}
    for filename in os.listdir(directory):
        model_name, ext = os.path.splitext(filename)
        try:
            if ext == '.h5':
                models[model_name] = load_model(os.path.join(directory, filename))
            elif ext == '.joblib':
                models[model_name] = load(os.path.join(directory, filename))
            logging.info(f"{model_name} model loaded")
        except Exception as e:
            logging.error(f"Error loading {model_name} model: {e}")
    return models

def main(dataset, target_column):
    """
    Main function to train, test, evaluate, and explain models.
    """
    X_train, X_test, y_train, y_test = split_dataset(dataset, target_column)

    # Standardization
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    logging.info("Data has been standardized")

    models = train_models(X_train, y_train)
    predictions = test_models(models, X_test)
```

```
    metrics = evaluate_models(models, predictions, y_test, X_test)

    explainability_shap(models, X_test, feature_names=dataset.drop(columns=[target_column]).columns)
    explainability_lime(models, X_train, X_test, feature_names=dataset.drop(columns=[target_column]).columns)

    save_models(models)
    logging.info("Models have been saved")

    return metrics

def modelling_gs(df):
    """
    Function to run the main pipeline with the given dataset.
    """
    target_column = 'LABEL'  # Replace with your target column
    results = main(df, target_column)
    logging.info(results)
    return results

# To run the modelling function with a dataset 'df':
# results = modelling_gs(df)
```

In [ ]:

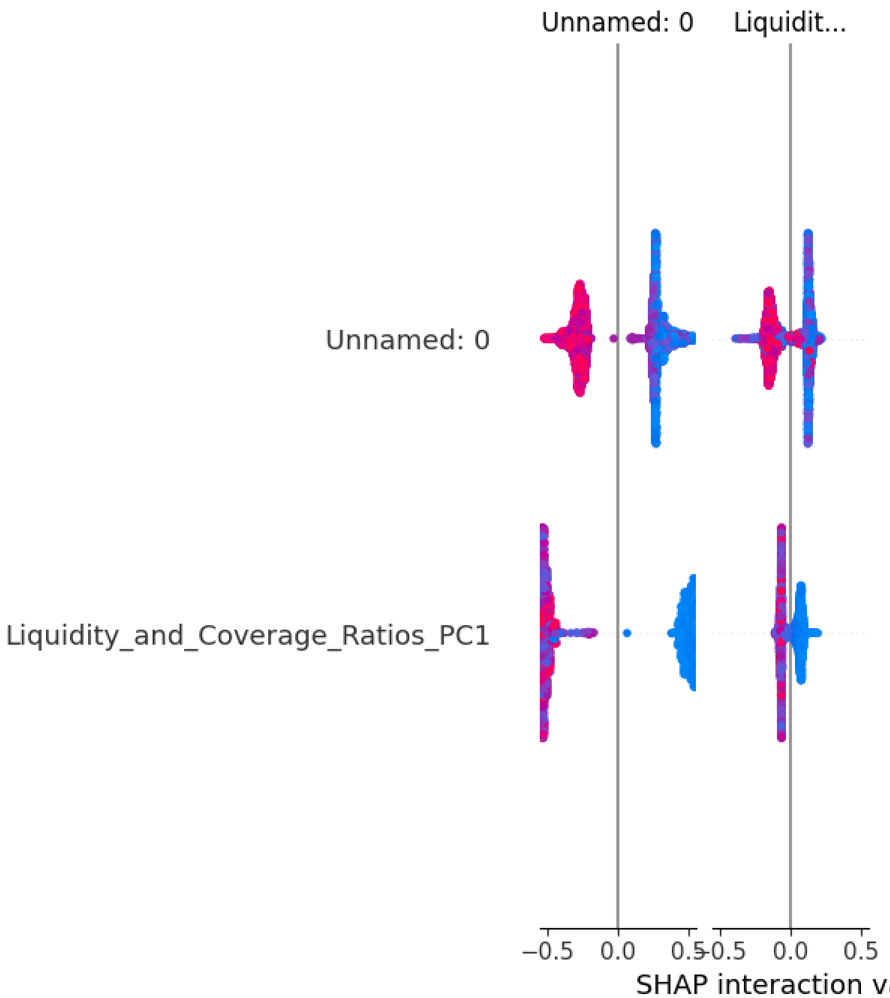
```
df_mice = pd.read_excel("C:\\Users\\dev\\Desktop\\Msc thesis Prior RS\\ML training\\df_mice_labeled_after_PCA.xlsx")
df_AE = pd.read_excel("C:\\Users\\dev\\Desktop\\Msc thesis Prior RS\\ML training\\df_autoencoder_labeled_after_PCA.x
```

In [ ]:

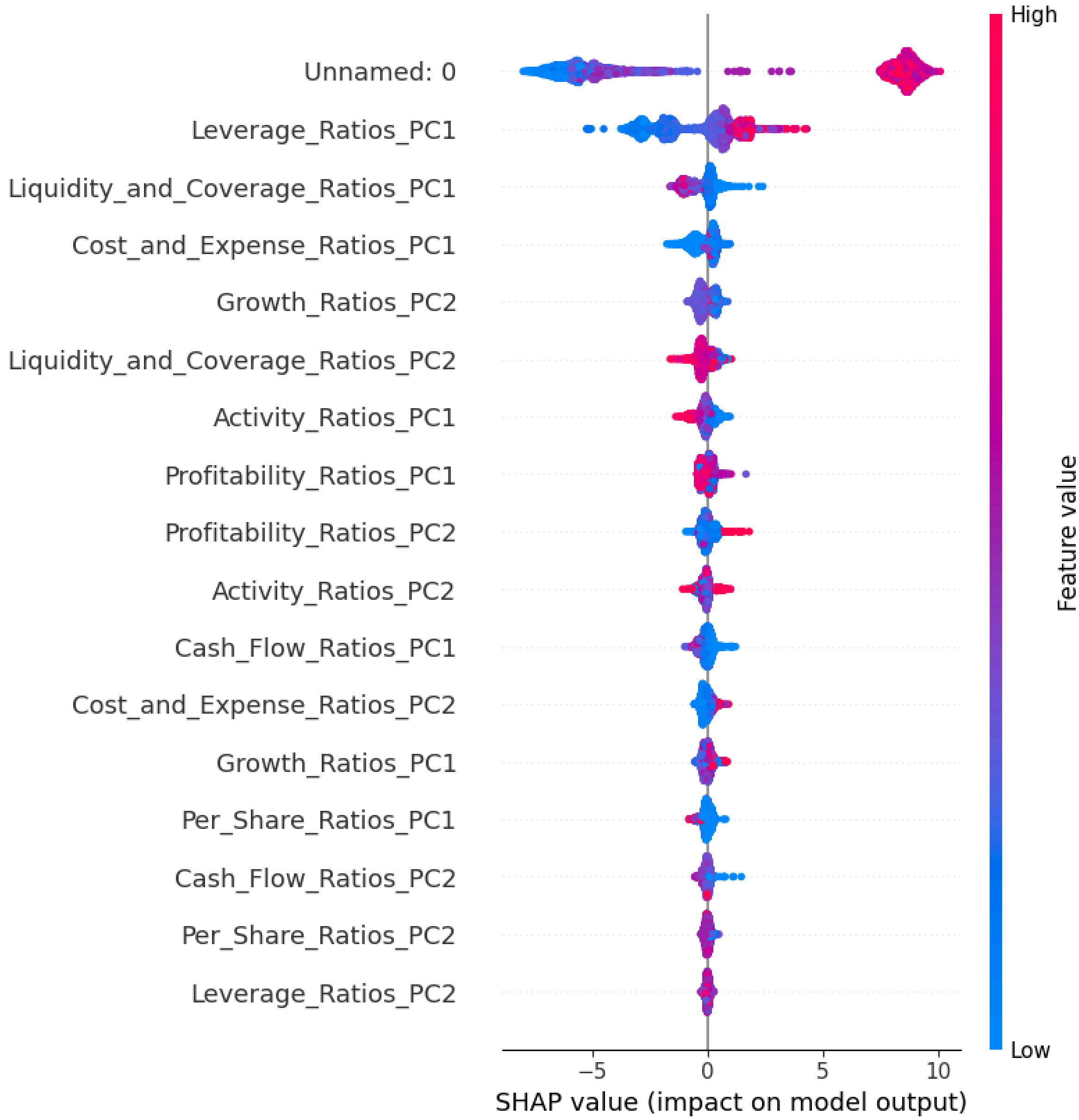
```
results_mice = modelling_gs(df_mice)
results_ae = modelling_gs(df_AE)

print("Results for df_mice")
print(f"{results_mice}")
print(" ")
print("___________________________________________________________________")
print(" ")
print("Results for df_AE")
print(f"{results_ae}")
```

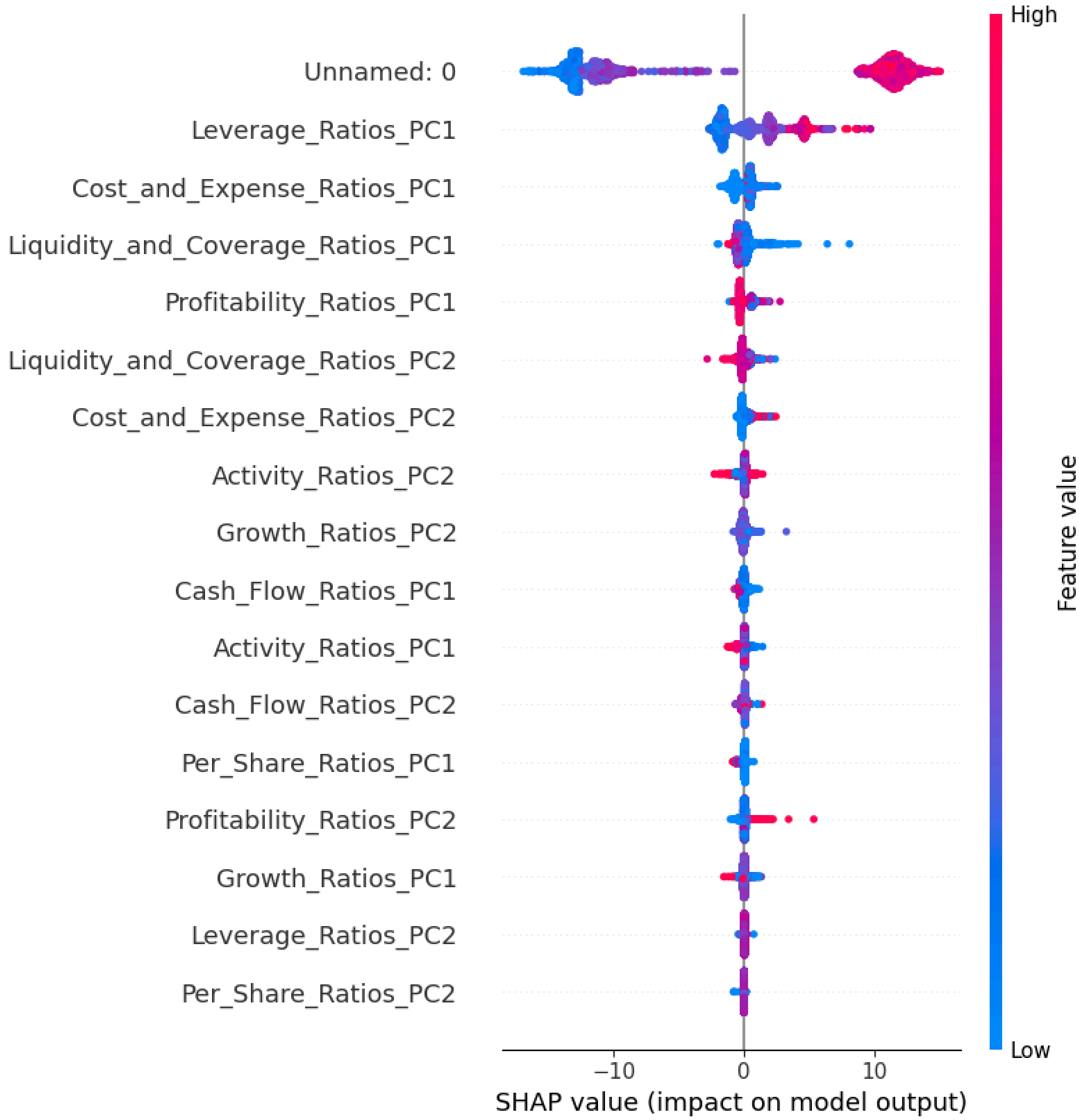
```
2024-06-28 17:33:48,449 - INFO - Dataset has been split and returned
2024-06-28 17:33:48,457 - INFO - Data has been standardized
2024-06-28 17:35:47,950 - INFO - ANN has been trained in 119.49 seconds
2024-06-28 17:43:06,218 - INFO - RandomForest has been trained in 438.27 seconds
2024-06-28 17:43:16,772 - INFO - XGBoost has been trained in 10.55 seconds
2024-06-28 17:45:22,266 - INFO - SVM has been trained in 125.49 seconds
2024-06-28 17:45:22,611 - INFO - LogisticRegression has been trained in 0.35 seconds
2024-06-28 18:17:28,288 - INFO - GradientBoosting has been trained in 1925.68 seconds
2024-06-28 18:17:30,087 - INFO - KNN has been trained in 1.80 seconds
2024-06-28 18:17:30,093 - INFO - Naive Bayes has been trained in 0.00 seconds
126/126 ━━━━━━━━━━━━━━━━━━━━ 0s 685us/step
2024-06-28 18:17:30,770 - INFO - Models have been tested in 0.68 seconds
126/126 ━━━━━━━━━━━━━━━━━━━━ 0s 527us/step
2024-06-28 18:17:31,453 - INFO - Models have been evaluated in 0.68 seconds
```



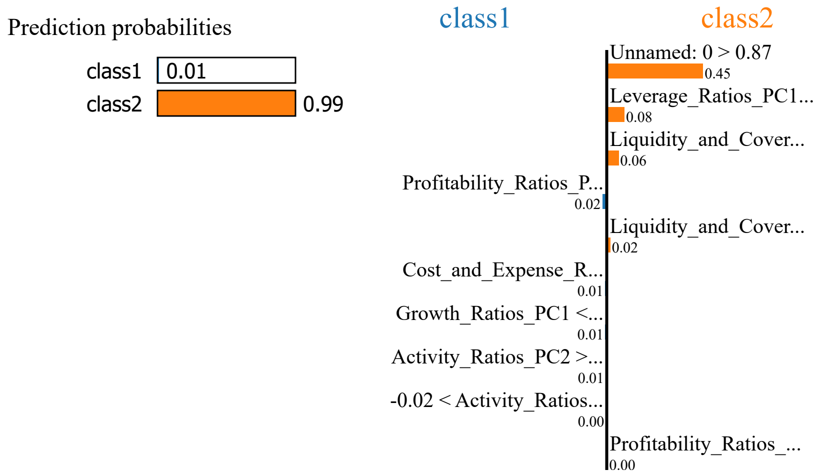

```
2024-06-28 18:18:02,647 - INFO - SHAP summary plot for RandomForest created
```




```
2024-06-28 18:18:03,822 - INFO - SHAP summary plot for XGBoost created
2024-06-28 18:18:03,822 - ERROR - Error generating SHAP explanations for SVM: Model type not yet supported by TreeExp
lainer: <class 'sklearn.svm._classes.SVC'>
2024-06-28 18:18:03,823 - ERROR - Error generating SHAP explanations for LogisticRegression: Model type not yet suppo
rted by TreeExplainer: <class 'sklearn.linear_model._logistic.LogisticRegression'>
```

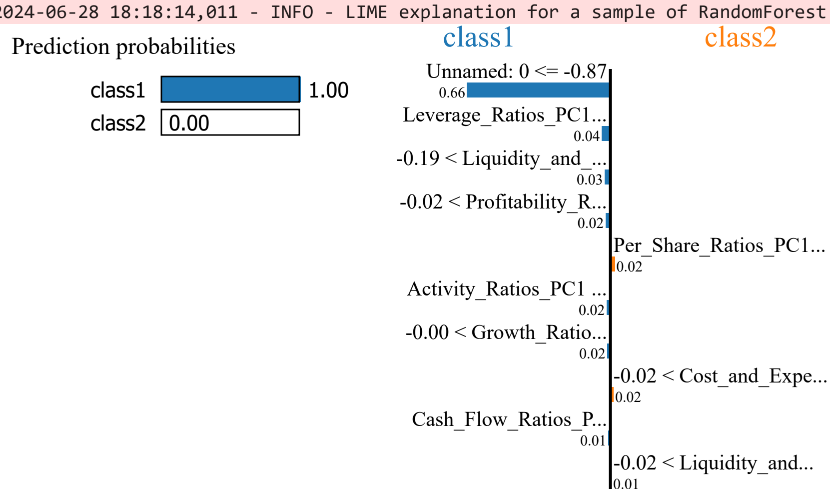



```
2024-06-28 18:18:13,847 - INFO - SHAP summary plot for GradientBoosting created
2024-06-28 18:18:13,847 - ERROR - Error generating SHAP explanations for KNN: Model type not yet supported by TreeExp
lainer: <class 'sklearn.neighbors._classification.KNeighborsClassifier'>
2024-06-28 18:18:13,848 - ERROR - Error generating SHAP explanations for NaiveBayes: Model type not yet supported by
TreeExplainer: <class 'sklearn.naive_bayes.GaussianNB'>
```



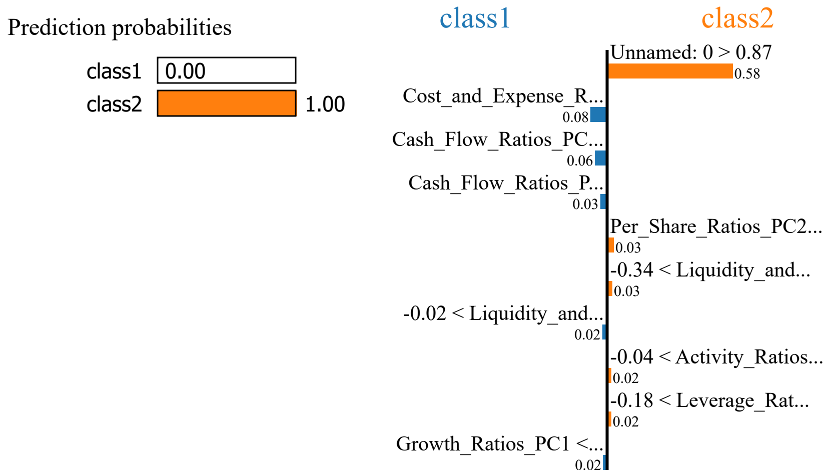
Feature	Value
Unnamed: 0	1.04
Leverage_Ratios_PC1	2.79
Liquidity_and_Coverage_Ratios_PC1	-0.56
Profitability_Ratios_PC1	1.07
Liquidity_and_Coverage_Ratios_PC2	-0.04
Cost_and_Expense_Ratios_PC1	1.64
Growth_Ratios_PC1	-0.21
Activity_Ratios_PC2	0.94

```
2024-06-28 18:18:14,011 - INFO - LIME explanation for a sample of RandomForest created
```



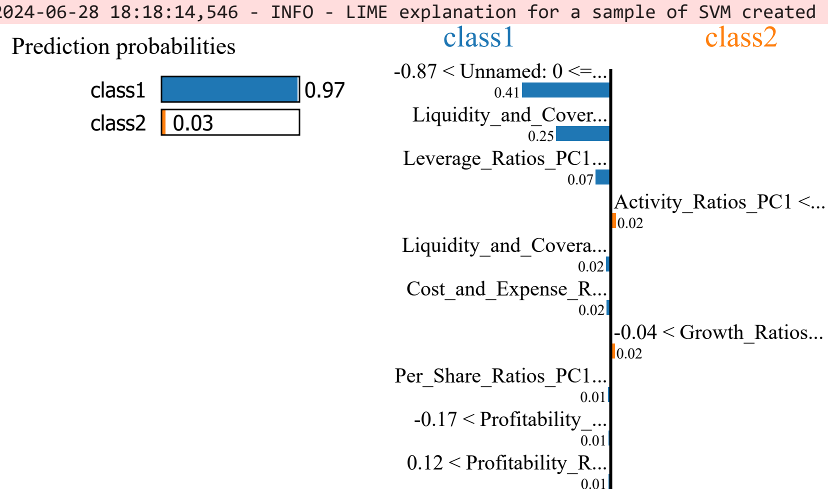
Feature	Value
Unnamed: 0	-1.07
Leverage_Ratios_PC1	-0.44
Liquidity_and_Coverage_Ratios_PC1	-0.02
Profitability_Ratios_PC1	0.09
Per_Share_Ratios_PC1	-0.03
Activity_Ratios_PC1	0.05
Growth_Ratios_PC2	0.00
Cost_and_Expense_Ratios_PC1	-0.02

```
2024-06-28 18:18:14,082 - INFO - LIME explanation for a sample of XGBoost created
```



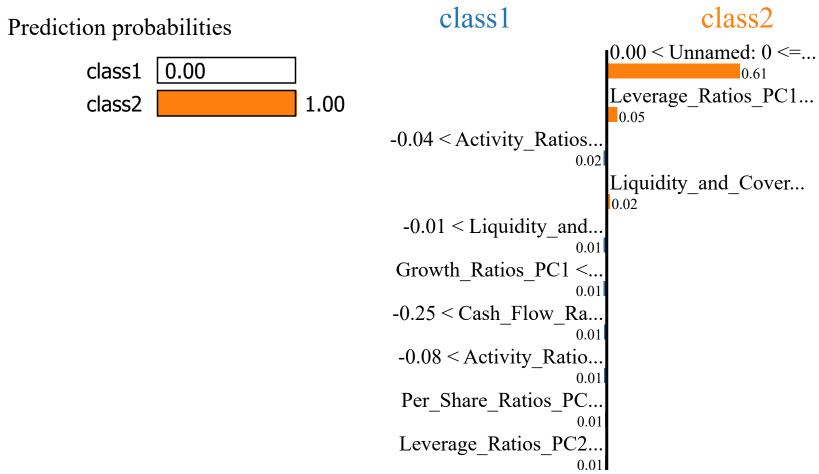
Feature	Value
Unnamed: 0	1.59
Cost_and_Expense_Ratios_PC1	-0.02
Cash_Flow_Ratios_PC1	0.37
Cash_Flow_Ratios_PC2	-0.05
Per_Share_Ratios_PC2	0.02
Liquidity_and_Coverage_Ratios_PC1	-0.33
Liquidity_and_Coverage_Ratios_PC2	-0.01
Activity_Ratios_PC1	-0.03

```
2024-06-28 18:18:14,546 - INFO - LIME explanation for a sample of SVM created
```



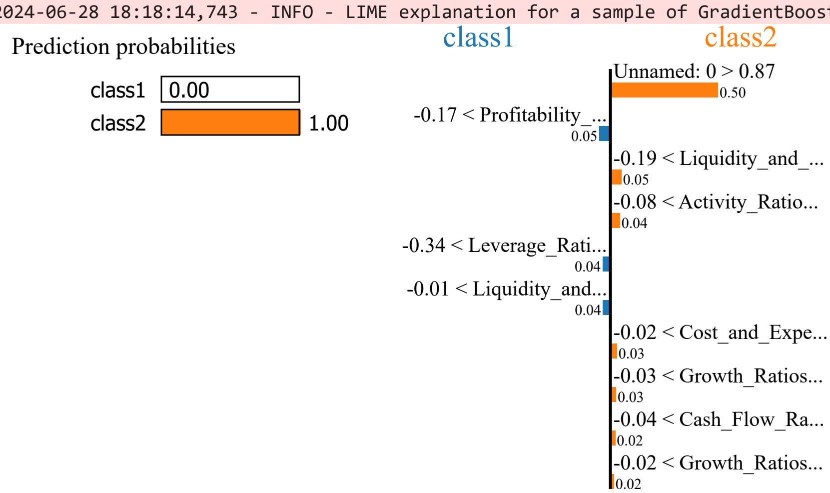
Feature	Value
Unnamed: 0	-0.30
Liquidity_and_Coverage_Ratios_PC1	0.35
Leverage_Ratios_PC1	-0.76
Activity_Ratios_PC1	-0.06
Liquidity_and_Coverage_Ratios_PC2	-0.01
Cost_and_Expense_Ratios_PC1	-0.01
Growth_Ratios_PC1	-0.04
Per_Share_Ratios_PC1	-0.03

```
2024-06-28 18:18:14,628 - INFO - LIME explanation for a sample of LogisticRegression created
```



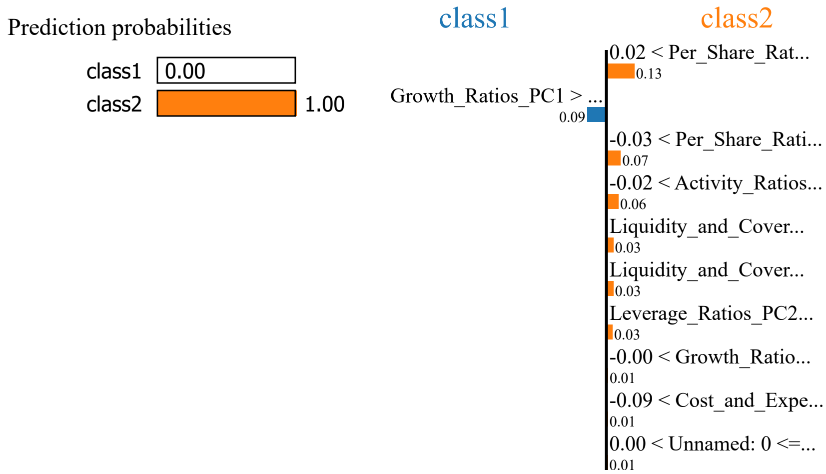
Feature	Value
Unnamed: 0	0.36
Leverage_Ratios_PC1	1.13
Activity_Ratios_PC1	-0.03
Liquidity_and_Coverage_Ratios_PC1	-0.37
Liquidity_and_Coverage_Ratios_PC2	-0.01
Growth_Ratios_PC1	-0.07
Cash_Flow_Ratios_PC1	-0.25
Activity_Ratios_PC2	0.05

```
2024-06-28 18:18:14,743 - INFO - LIME explanation for a sample of GradientBoosting created
```



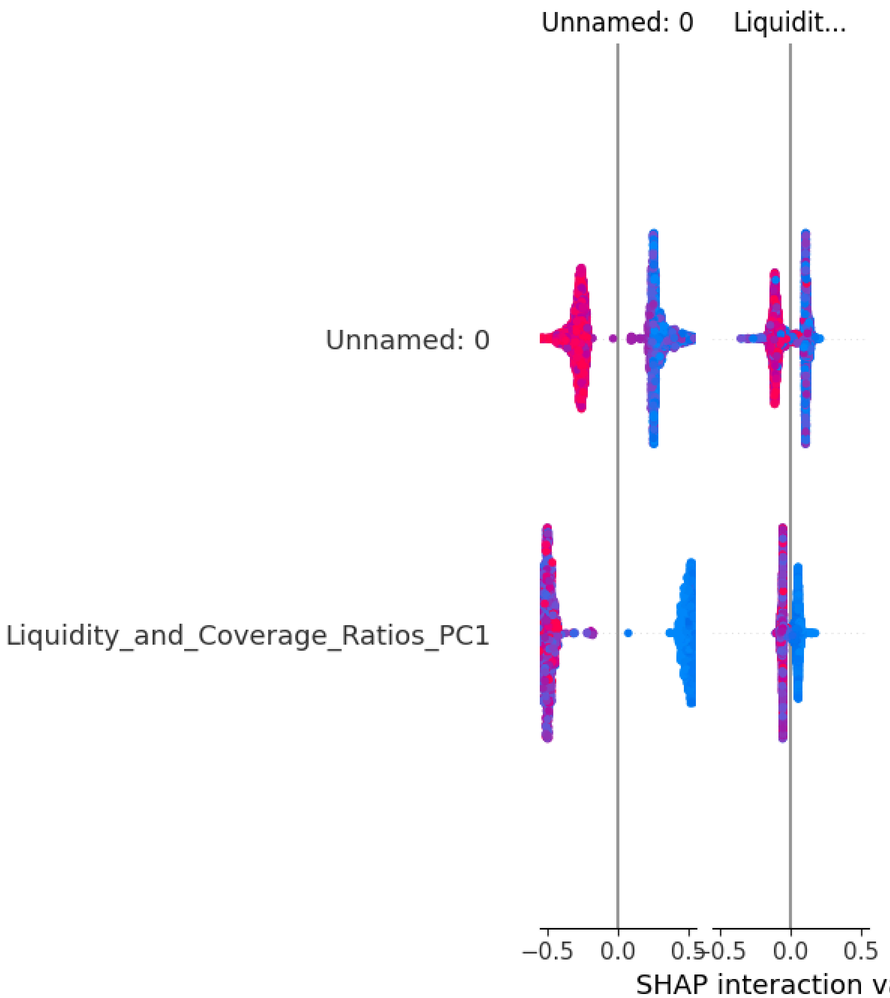
Feature	Value
Unnamed: 0	1.72
Profitability_Ratios_PC2	-0.17
Liquidity_and_Coverage_Ratios_PC1	-0.17
Activity_Ratios_PC2	0.08
Leverage_Ratios_PC1	-0.18
Liquidity_and_Coverage_Ratios_PC2	-0.01
Cost_and_Expense_Ratios_PC1	-0.02
Growth_Ratios_PC1	-0.03

```
2024-06-28 18:18:14,977 - INFO - LIME explanation for a sample of KNN created
```

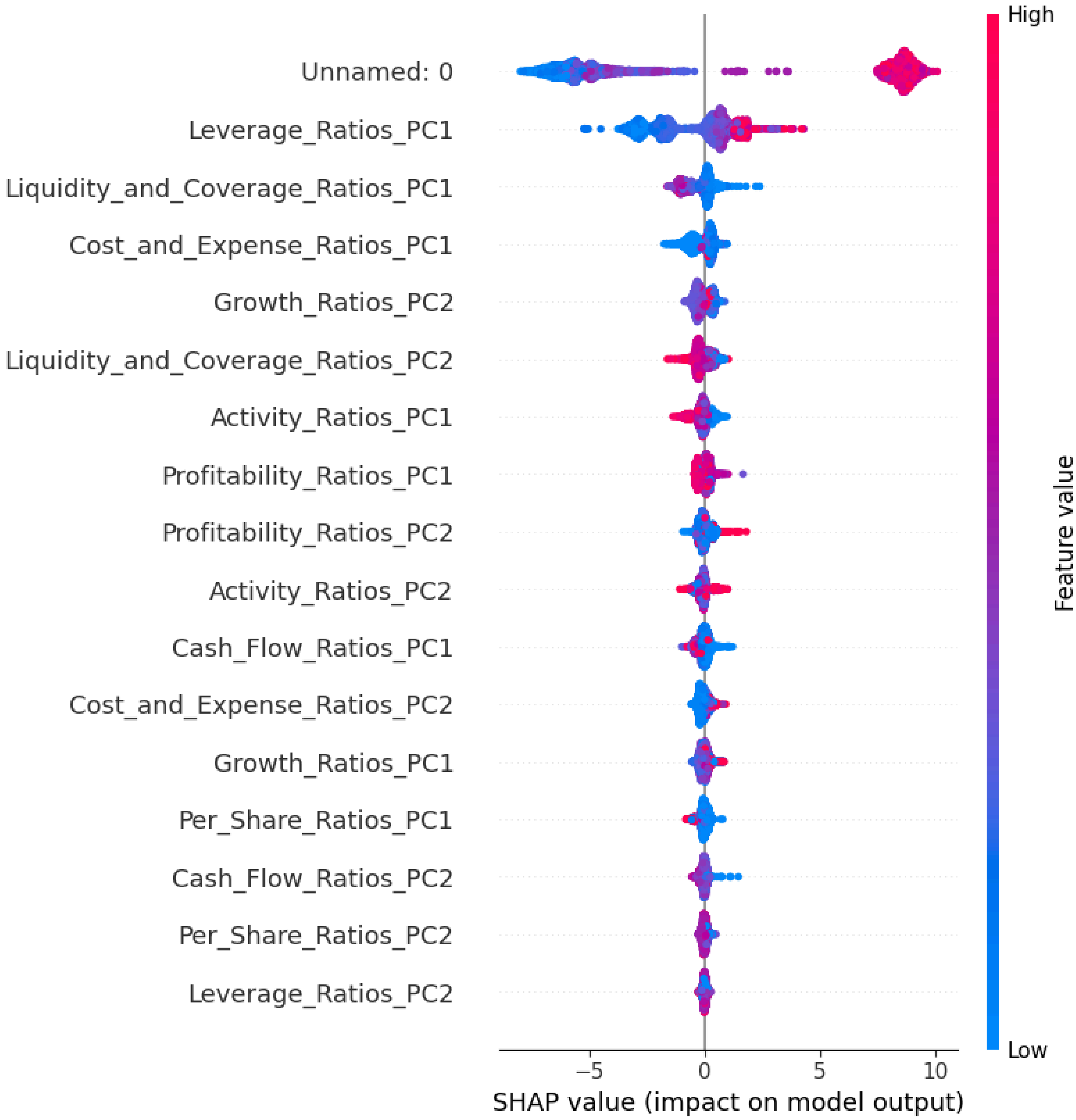


Feature	Value
Per_Share_Ratios_PC2	0.02
Growth_Ratios_PC1	0.09
Per_Share_Ratios_PC1	-0.03
Activity_Ratios_PC1	-0.02
Liquidity_and_Coverage_Ratios_PC1	-0.44
Liquidity_and_Coverage_Ratios_PC2	-0.02
Leverage_Ratios_PC2	-0.03
Growth_Ratios_PC2	0.01

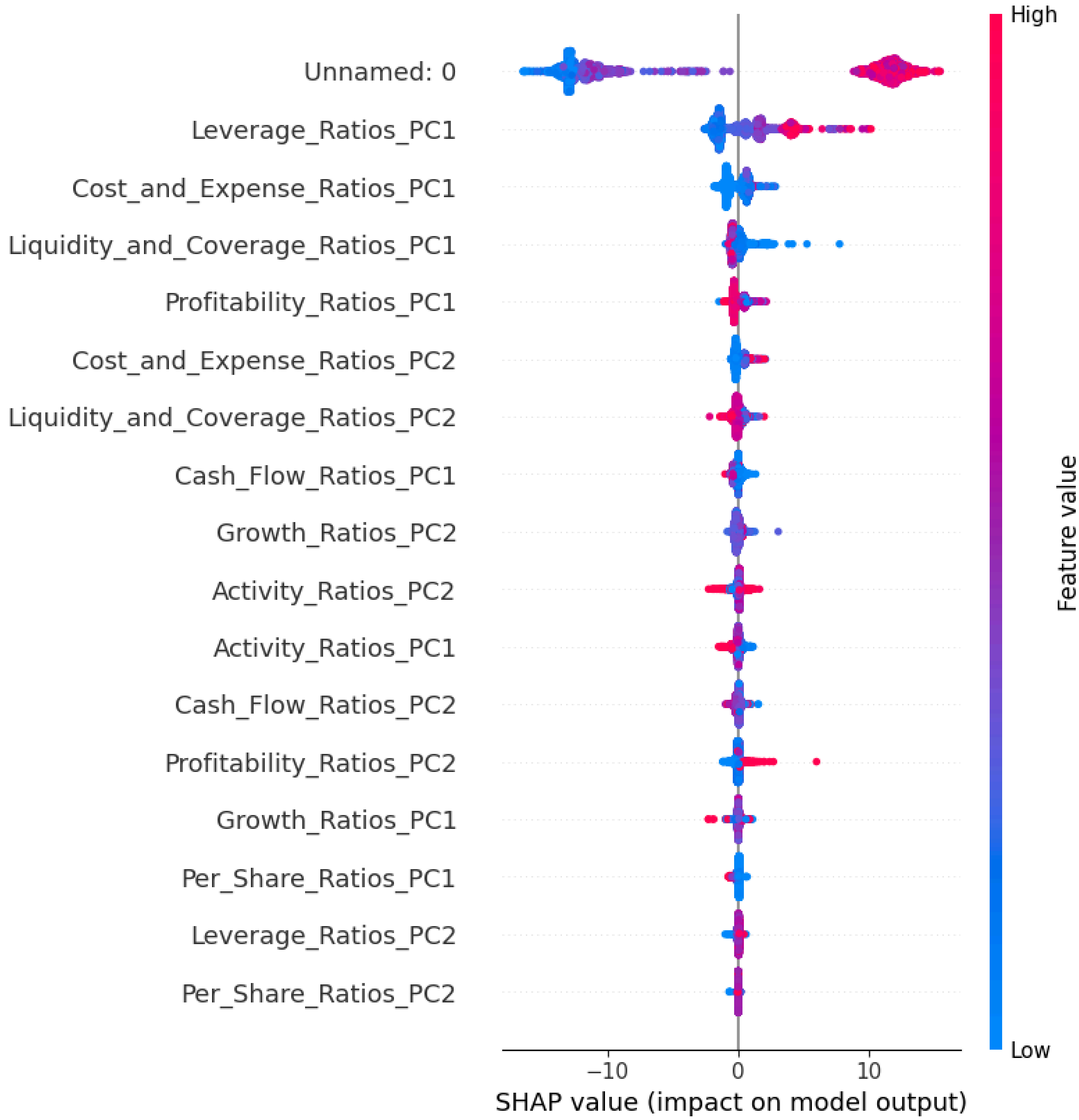
```
2024-06-28 18:18:15,039 - INFO - LIME explanation for a sample of NaiveBayes created
2024-06-28 18:18:15,039 - WARNING - You are saving your model as an HDF5 file via `model.save()` or `keras.saving.sav
e_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `mod
el.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
2024-06-28 18:18:15,059 - INFO - ANN model saved
2024-06-28 18:18:15,148 - INFO - RandomForest model saved
2024-06-28 18:18:15,156 - INFO - XGBoost model saved
2024-06-28 18:18:15,159 - INFO - SVM model saved
2024-06-28 18:18:15,161 - INFO - LogisticRegression model saved
2024-06-28 18:18:15,188 - INFO - GradientBoosting model saved
2024-06-28 18:18:15,192 - INFO - KNN model saved
2024-06-28 18:18:15,194 - INFO - NaiveBayes model saved
2024-06-28 18:18:15,194 - INFO - Models have been saved
2024-06-28 18:18:15,197 - INFO - {'ANN': {'accuracy': 0.9878260869565217, 'confusion_matrix': array([[1995,   17],
       [  32, 1981]], dtype=int64), 'f1_score': 0.987783595113438, 'auc_roc': 0.9993836039895747}, 'RandomForest':
{'accuracy': 0.9937888198757764, 'confusion_matrix': array([[2008,    4],
       [  21, 1992]], dtype=int64), 'f1_score': 0.9937640309304066, 'auc_roc': 0.9998035878124201}, 'XGBoost': {'accu
racy': 0.995527950310559, 'confusion_matrix': array([[2005,    7],
       [  11, 2002]], dtype=int64), 'f1_score': 0.9955246146195923, 'auc_roc': 0.9998471663807518}, 'SVM': {'accurac
y': 0.9836024844720497, 'confusion_matrix': array([[1986,   26],
       [  40, 1973]], dtype=int64), 'f1_score': 0.9835493519441675, 'auc_roc': 0.997308004926 23}, 'LogisticRegressio
n': {'accuracy': 0.977888198757764, 'confusion_matrix': array([[1971,   41],
       [  48, 1965]], dtype=int64), 'f1_score': 0.977855187857676, 'auc_roc': 0.9942133093145054}, 'GradientBoostin
g': {'accuracy': 0.9975155279503105, 'confusion_matrix': array([[2012,    0],
       [  10, 2003]], dtype=int64), 'f1_score': 0.9975099601593626, 'auc_roc': 0.9999367925581139}, 'KNN': {'accurac
y': 0.9853416149068323, 'confusion_matrix': array([[1978,   34],
       [  25, 1988]], dtype=int64), 'f1_score': 0.9853779429987608, 'auc_roc': 0.996077681946078}, 'NaiveBayes': {'ac
curacy': 0.564223602484472, 'confusion_matrix': array([[ 317, 1695],
       [  59, 1954]], dtype=int64), 'f1_score': 0.6902154715648181, 'auc_roc': 0.944120794359526}}
2024-06-28 18:18:15,211 - INFO - Dataset has been split and returned
2024-06-28 18:18:15,223 - INFO - Data has been standardized
2024-06-28 18:20:14,168 - INFO - ANN has been trained in 118.95 seconds
2024-06-28 18:27:42,607 - INFO - RandomForest has been trained in 448.44 seconds
2024-06-28 18:27:53,366 - INFO - XGBoost has been trained in 10.76 seconds
2024-06-28 18:29:58,901 - INFO - SVM has been trained in 125.53 seconds
2024-06-28 18:29:59,247 - INFO - LogisticRegression has been trained in 0.35 seconds
2024-06-28 20:41:58,820 - INFO - GradientBoosting has been trained in 7919.57 seconds
2024-06-28 20:42:07,660 - INFO - KNN has been trained in 8.84 seconds
2024-06-28 20:42:07,688 - INFO - Naive Bayes has been trained in 0.03 seconds
126/126 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step
2024-06-28 20:42:11,334 - INFO - Models have been tested in 3.64 seconds
126/126 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
2024-06-28 20:42:15,153 - INFO - Models have been evaluated in 3.82 seconds
```



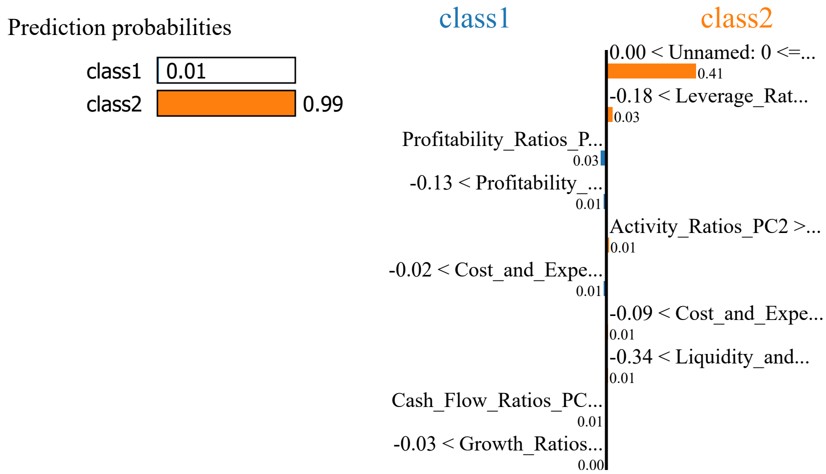

```
2024-06-28 20:43:41,903 - INFO - SHAP summary plot for RandomForest created
```




```
2024-06-28 20:43:47,904 - INFO - SHAP summary plot for XGBoost created
2024-06-28 20:43:47,907 - ERROR - Error generating SHAP explanations for SVM: Model type not yet supported by TreeExp
lainer: <class 'sklearn.svm._classes.SVC'>
2024-06-28 20:43:47,909 - ERROR - Error generating SHAP explanations for LogisticRegression: Model type not yet suppo
rted by TreeExplainer: <class 'sklearn.linear_model._logistic.LogisticRegression'>
```

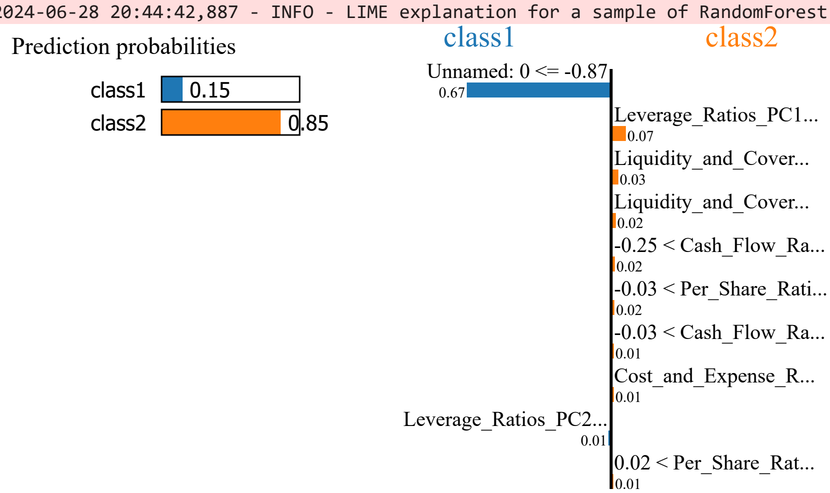



```
2024-06-28 20:44:41,998 - INFO - SHAP summary plot for GradientBoosting created
2024-06-28 20:44:42,001 - ERROR - Error generating SHAP explanations for KNN: Model type not yet supported by TreeExplainer: <class 'sklearn.neighbors._classification.KNeighborsClassifier'>
2024-06-28 20:44:42,004 - ERROR - Error generating SHAP explanations for NaiveBayes: Model type not yet supported by TreeExplainer: <class 'sklearn.naive_bayes.GaussianNB'>
```



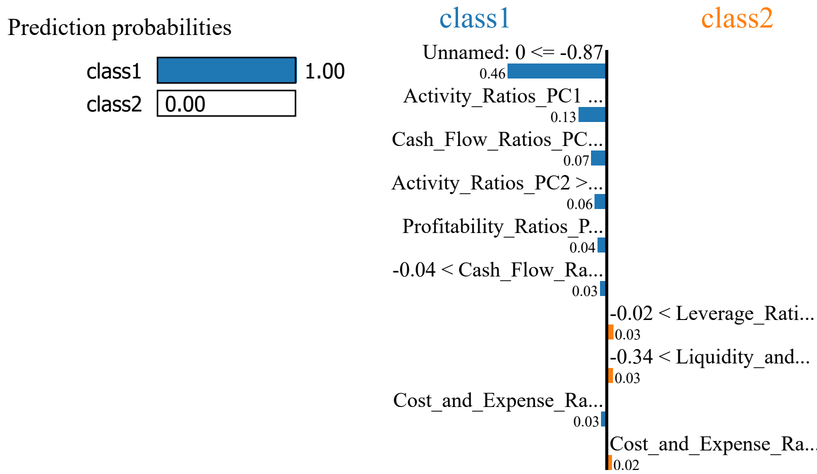
Feature	Value
Unnamed: 0	0.38
Leverage_Ratios_PC1	-0.15
Profitability_Ratios_PC1	0.26
Profitability_Ratios_PC2	-0.10
Activity_Ratios_PC2	0.92
Cost_and_Expense_Ratios_PC1	-0.02
Cost_and_Expense_Ratios_PC2	-0.08
Liquidity_and_Coverage_Ratios_PC1	-0.25

```
2024-06-28 20:44:42,887 - INFO - LIME explanation for a sample of RandomForest created
```



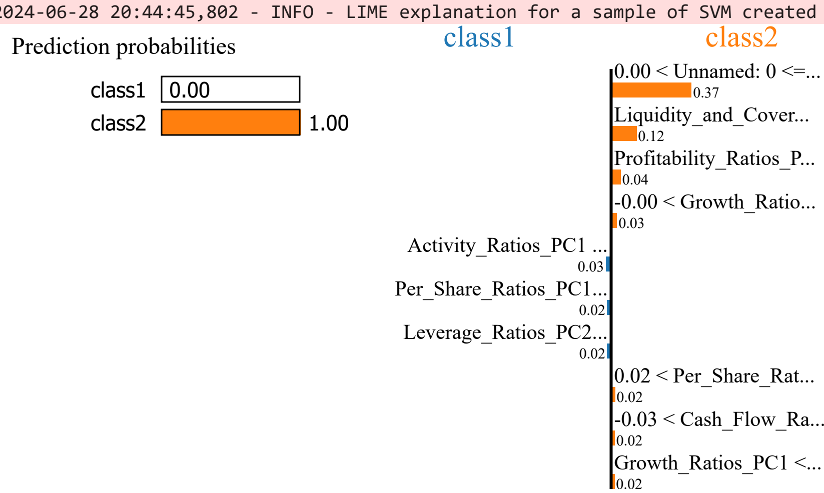
Feature	Value
Unnamed: 0	-1.15
Leverage_Ratios_PC1	1.02
Liquidity_and_Coverage_Ratios_PC1	-0.47
Liquidity_and_Coverage_Ratios_PC2	-0.02
Cash_Flow_Ratios_PC1	-0.24
Per_Share_Ratios_PC1	-0.03
Cash_Flow_Ratios_PC2	-0.03
Cost_and_Expense_Ratios_PC2	0.23

```
2024-06-28 20:44:43,253 - INFO - LIME explanation for a sample of XGBoost created
```



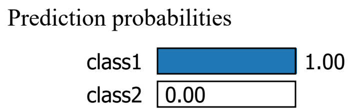
Feature	Value
Unnamed: 0	-1.61
Activity_Ratios_PC1	0.09
Cash_Flow_Ratios_PC1	0.19
Activity_Ratios_PC2	1.40
Profitability_Ratios_PC1	0.29
Cash_Flow_Ratios_PC2	-0.03
Leverage_Ratios_PC2	-0.02
Liquidity_and_Coverage_Ratios_PC1	-0.24

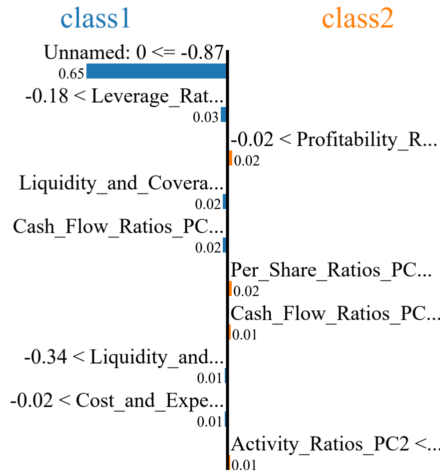
```
2024-06-28 20:44:45,802 - INFO - LIME explanation for a sample of SVM created
```



Feature	Value
Unnamed: 0	0.86
Liquidity_and_Coverage_Ratios_PC1	-0.36
Profitability_Ratios_PC1	-0.36
Growth_Ratios_PC2	0.00
Activity_Ratios_PC1	0.00
Per_Share_Ratios_PC1	-0.03
Leverage_Ratios_PC2	0.01
Per_Share_Ratios_PC2	0.02

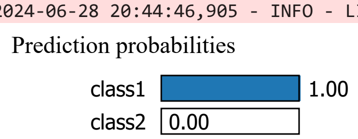
```
2024-06-28 20:44:46,109 - INFO - LIME explanation for a sample of LogisticRegression created
```

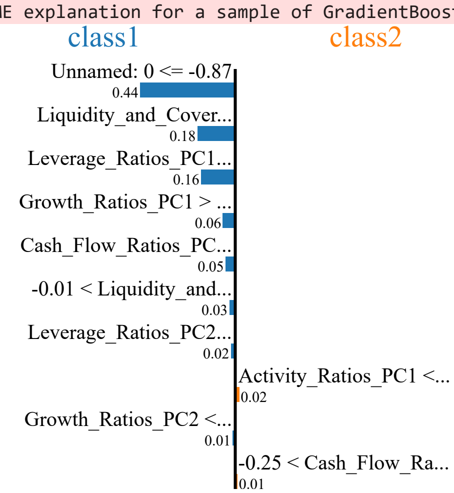




Feature	Value
Unnamed: 0	-0.99
Leverage_Ratios_PC1	-0.13
Profitability_Ratios_PC1	0.03
Liquidity_and_Coverage_Ratios_PC2	-0.00
Cash_Flow_Ratios_PC1	0.48
Per_Share_Ratios_PC2	0.02
Cash_Flow_Ratios_PC2	0.15
Liquidity_and_Coverage_Ratios_PC1	-0.20

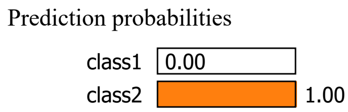
```
2024-06-28 20:44:46,905 - INFO - LIME explanation for a sample of GradientBoosting created
```

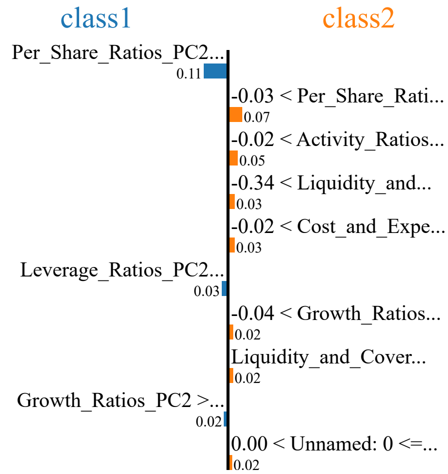




Feature	Value
Unnamed: 0	-0.97
Liquidity_and_Coverage_Ratios_PC1	0.46
Leverage_Ratios_PC1	-0.50
Growth_Ratios_PC1	0.01
Cash_Flow_Ratios_PC2	0.00
Liquidity_and_Coverage_Ratios_PC2	-0.01
Leverage_Ratios_PC2	-0.02
Activity_Ratios_PC1	-0.06

```
2024-06-28 20:44:48,180 - INFO - LIME explanation for a sample of KNN created
```





Feature	Value
Per_Share_Ratios_PC2	0.02
Per_Share_Ratios_PC1	-0.03
Activity_Ratios_PC1	-0.01
Liquidity_and_Coverage_Ratios_PC1	-0.25
Cost_and_Expense_Ratios_PC1	-0.02
Leverage_Ratios_PC2	0.04
Growth_Ratios_PC1	-0.04
Liquidity_and_Coverage_Ratios_PC2	-0.03

```
2024-06-28 20:44:48,481 - INFO - LIME explanation for a sample of NaiveBayes created
2024-06-28 20:44:48,485 - WARNING - You are saving your model as an HDF5 file via `model.save()` or `keras.saving.sav
e_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `mod
el.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
2024-06-28 20:44:48,613 - INFO - ANN model saved
2024-06-28 20:44:48,861 - INFO - RandomForest model saved
2024-06-28 20:44:48,901 - INFO - XGBoost model saved
2024-06-28 20:44:48,913 - INFO - SVM model saved
2024-06-28 20:44:48,922 - INFO - LogisticRegression model saved
2024-06-28 20:44:49,080 - INFO - GradientBoosting model saved
2024-06-28 20:44:49,096 - INFO - KNN model saved
2024-06-28 20:44:49,106 - INFO - NaiveBayes model saved
2024-06-28 20:44:49,108 - INFO - Models have been saved
2024-06-28 20:44:49,117 - INFO - {'ANN': {'accuracy': 0.986832298136646, 'confusion_matrix': array([[1982,   30],
       [  23, 1990]], dtype=int64), 'f1_score': 0.9868584180510787, 'auc_roc': 0.9982137725065405}, 'RandomForest':
{'accuracy': 0.9935403726708074, 'confusion_matrix': array([[2007,    5],
       [  21, 1992]], dtype=int64), 'f1_score': 0.9935162094763093, 'auc_roc': 0.9997444542876868}, 'XGBoost': {'accu
racy': 0.995527950310559, 'confusion_matrix': array([[2005,    7],
       [  11, 2002]], dtype=int64), 'f1_score': 0.9955246146195923, 'auc_roc': 0.9998471663807518}, 'SVM': {'accurac
y': 0.9836024844720497, 'confusion_matrix': array([[1986,   26],
       [  40, 1973]], dtype=int64), 'f1_score': 0.9835493519441675, 'auc_roc': 0.99730800492623}, 'LogisticRegressio
n': {'accuracy': 0.977888198757764, 'confusion_matrix': array([[1971,   41],
       [  48, 1965]], dtype=int64), 'f1_score': 0.977855187857676, 'auc_roc': 0.9942133093145054}, 'GradientBoostin
g': {'accuracy': 0.9970186335403727, 'confusion_matrix': array([[2010,    2],
       [  10, 2003]], dtype=int64), 'f1_score': 0.9970134395221503, 'auc_roc': 0.9999017321801926}, 'KNN': {'accurac
y': 0.9853416149068323, 'confusion_matrix': array([[1978,   34],
       [  25, 1988]], dtype=int64), 'f1_score': 0.9853779429987608, 'auc_roc': 0.996077681946078}, 'NaiveBayes': {'ac
curacy': 0.564223602484472, 'confusion_matrix': array([[ 317, 1695],
       [  59, 1954]], dtype=int64), 'f1_score': 0.6902154715648181, 'auc_roc': 0.944120794359526}}
```

```
Results for df_mice
{'ANN': {'accuracy': 0.9878260869565217, 'confusion_matrix': array([[1995,   17],
       [  32, 1981]], dtype=int64), 'f1_score': 0.987783595113438, 'auc_roc': 0.9993836039895747}, 'RandomForest':
{'accuracy': 0.9937888198757764, 'confusion_matrix': array([[2008,    4],
       [  21, 1992]], dtype=int64), 'f1_score': 0.9937640309304066, 'auc_roc': 0.9998035878124201}, 'XGBoost': {'accu
racy': 0.995527950310559, 'confusion_matrix': array([[2005,    7],
       [  11, 2002]], dtype=int64), 'f1_score': 0.9955246146195923, 'auc_roc': 0.9998471663807518}, 'SVM': {'accurac
y': 0.9836024844720497, 'confusion_matrix': array([[1986,   26],
       [  40, 1973]], dtype=int64), 'f1_score': 0.9835493519441675, 'auc_roc': 0.99730800492623}, 'LogisticRegressio
n': {'accuracy': 0.977888198757764, 'confusion_matrix': array([[1971,   41],
       [  48, 1965]], dtype=int64), 'f1_score': 0.977855187857676, 'auc_roc': 0.9942133093145054}, 'GradientBoostin
g': {'accuracy': 0.9975155279503105, 'confusion_matrix': array([[2012,    0],
       [  10, 2003]], dtype=int64), 'f1_score': 0.9975099601593626, 'auc_roc': 0.9999367925581139}, 'KNN': {'accurac
y': 0.9853416149068323, 'confusion_matrix': array([[1978,   34],
       [  25, 1988]], dtype=int64), 'f1_score': 0.9853779429987608, 'auc_roc': 0.996077681946078}, 'NaiveBayes': {'ac
curacy': 0.564223602484472, 'confusion_matrix': array([[ 317, 1695],
       [  59, 1954]], dtype=int64), 'f1_score': 0.6902154715648181, 'auc_roc': 0.944120794359526}}
```

```
Results for df_AE
{'ANN': {'accuracy': 0.986832298136646, 'confusion_matrix': array([[1982,   30],
       [  23, 1990]], dtype=int64), 'f1_score': 0.9868584180510787, 'auc_roc': 0.9982137725065405}, 'RandomForest':
{'accuracy': 0.9935403726708074, 'confusion_matrix': array([[2007,    5],
       [  21, 1992]], dtype=int64), 'f1_score': 0.9935162094763093, 'auc_roc': 0.9997444542876868}, 'XGBoost': {'accu
racy': 0.995527950310559, 'confusion_matrix': array([[2005,    7],
       [  11, 2002]], dtype=int64), 'f1_score': 0.9955246146195923, 'auc_roc': 0.9998471663807518}, 'SVM': {'accurac
y': 0.9836024844720497, 'confusion_matrix': array([[1986,   26],
       [  40, 1973]], dtype=int64), 'f1_score': 0.9835493519441675, 'auc_roc': 0.99730800492623}, 'LogisticRegressio
n': {'accuracy': 0.977888198757764, 'confusion_matrix': array([[1971,   41],
       [  48, 1965]], dtype=int64), 'f1_score': 0.977855187857676, 'auc_roc': 0.9942133093145054}, 'GradientBoostin
g': {'accuracy': 0.9970186335403727, 'confusion_matrix': array([[2010,    2],
       [  10, 2003]], dtype=int64), 'f1_score': 0.9970134395221503, 'auc_roc': 0.9999017321801926}, 'KNN': {'accurac
y': 0.9853416149068323, 'confusion_matrix': array([[1978,   34],
       [  25, 1988]], dtype=int64), 'f1_score': 0.9853779429987608, 'auc_roc': 0.996077681946078}, 'NaiveBayes': {'ac
curacy': 0.564223602484472, 'confusion_matrix': array([[ 317, 1695],
       [  59, 1954]], dtype=int64), 'f1_score': 0.6902154715648181, 'auc_roc': 0.944120794359526}}
```