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

from datetime import datetime

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
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, precision_score, re
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)
            precision = precision_score(y_test, y_pred)
            recall = recall_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,
                'precision': precision,
                'recall': recall,
                '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, df_name, X_test, feature_names):

    """

    """
    # Ensure X_test is a DataFrame with named columns
    X_test = pd.DataFrame(X_test, columns=feature_names).reset_index(drop=True)

    for name, model in models.items():
        if name == 'ANN':
            continue
        try:
            if name in ['RandomForest', 'XGBoost', 'GradientBoosting']:
                explainer = shap.TreeExplainer(model)

            # No existing methods to analyse other models using SHAP, so only these three models.

            shap_values = explainer.shap_values(X_test)

            plt.figure(figsize=(10, 6))
            shap.summary_plot(shap_values[1] if isinstance(shap_values, list) else shap_values,
                              X_test, plot_type="bar", show=False, max_display=10)
            plt.title(f"Top 10 Most Important Features - {name}")
            plt.tight_layout()
            plt.savefig(f"C:\\Users\\dev\\Desktop\\MSC thesis\\Code\\final_codes\\Lime and shap graphs\\{df_name}_sha
            plt.close()
            logging.info(f"SHAP explanations for {name} created and saved")
        except Exception as e:
            logging.error(f"Error generating SHAP explanations for {name}: {e}")



def explainability_lime(models, df_name, X_train, X_test, feature_names):

    """

    """
    # Ensure X_train and X_test are DataFrames with named columns
    X_train = pd.DataFrame(X_train, columns=feature_names).reset_index(drop=True)
    X_test = pd.DataFrame(X_test, columns=feature_names).reset_index(drop=True)

    explainer = lime.lime_tabular.LimeTabularExplainer(
        X_train.values,  # Use .values to get numpy array
        feature_names=feature_names,
        class_names=['Negative', 'Positive'],
        mode='classification'
    )
    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.iloc[i].values,  # Use .iloc[i].values to get numpy array
                model.predict_proba,
                num_features=6
            )
            feature_importance = pd.DataFrame(exp.as_list(), columns=['Feature', 'Importance'])
            feature_importance['Absolute Importance'] = abs(feature_importance['Importance'])
            feature_importance = feature_importance.sort_values('Absolute Importance', ascending=True)
            plt.figure(figsize=(10, 6))
            colors = ['red' if imp < 0 else 'green' for imp in feature_importance['Importance']]
            plt.barh(feature_importance['Feature'], feature_importance['Importance'], color=colors)
            plt.title(f"LIME Explanation for {name}\nTop 6 Features' Impact on Prediction")
            plt.xlabel('Impact on Prediction (Red = Negative, Green = Positive)')
            plt.tight_layout()
            plt.savefig(f"C:\\Users\\dev\\Desktop\\MSC thesis\\Code\\final_codes\\Lime and shap graphs\\{df_name}_lir
            plt.close()
            logging.info(f"LIME explanation for {name} created and saved")
        except Exception as e:
            logging.error(f"Error generating LIME explanations for {name}: {e}")
```

```
def interpret_results(models, X_test, feature_names):
    summary = "Model Interpretation Summary:\n\n"
    for name, model in models.items():
        if name == 'ANN':
            continue
        summary += f"{name} Model:\n"
        summary += f"Feature Importance from {name} Model:\n"
        try:
            if name in ['RandomForest', 'XGBoost', 'GradientBoosting']:
                importances = model.feature_importances_
                importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': importances})
                importance_df = importance_df.sort_values('Importance', ascending=False).head(10)
            else:
                importances = model.coef_[0] if hasattr(model, 'coef_') else None
                importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': importances})
                importance_df = importance_df.sort_values('Importance', ascending=False).head(10)
            summary += importance_df.to_string(index=False)
            summary += "\n\n"
        except Exception as e:
            logging.error(f"Error interpreting results for {name}: {e}")
    logging.info("Model interpretation summary created")
    return summary


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}")


# Use only if needed to run back with best models
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, name):
    """
    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, name, X_test, feature_names=dataset.drop(columns=[target_column]).columns)
    explainability_lime(models, name, X_train, X_test, feature_names=dataset.drop(columns=[target_column]).columns)

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

    # Interpret results
    summary = interpret_results(models, X_test, feature_names=dataset.drop(columns=[target_column]).columns)
    print(summary)

    return metrics


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

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

In [ ]:

```
file_paths = [
    "C:\\Users\\dev\\Desktop\\MSC thesis\\Code\\final_codes\\Processed Datasets\\ADASYN_AE_3_PCA.xlsx",
    "C:\\Users\\dev\\Desktop\\MSC thesis\\Code\\final_codes\\Processed Datasets\\ADASYN_MICE_3_PCA.xlsx",
    "C:\\Users\\dev\\Desktop\\MSC thesis\\Code\\final_codes\\Processed Datasets\\KMSMOTE_AE_3_PCA.xlsx",
    "C:\\Users\\dev\\Desktop\\MSC thesis\\Code\\final_codes\\Processed Datasets\\KMSMOTE_MICE_3_PCA.xlsx",
    "C:\\Users\\dev\\Desktop\\MSC thesis\\Code\\final_codes\\Processed Datasets\\SVMSMOTE_AE_3_PCA.xlsx",
    "C:\\Users\\dev\\Desktop\\MSC thesis\\Code\\final_codes\\Processed Datasets\\SVMSMOTE_MICE_3_PCA.xlsx"
]

# Read the Excel files into dataframes
dfs = [pd.read_excel(file_path) for file_path in file_paths]

print("Datasets are read into dataframes")

tot_start_time = time.time()
start_time = time.time()
# Store results in variables
results_ADASYN_AE_3_PCA = modelling_gs(dfs[0], "ADASYN_AE_3_PCA" )
end_time = time.time()  # End timing
elapsed_time = (end_time - start_time) / 60
print("_____")
print(f" Total time taken by ADASYN_AE_3_PCA: {elapsed_time:.2f} mins")

start_time = time.time()
results_ADASYN_MICE_3_PCA = modelling_gs(dfs[1], "ADASYN_MICE_3_PCA")

end_time = time.time()  # End timing
elapsed_time = (end_time - start_time) / 60
print("_____")
print(f" Total time taken by ADASYN_MICE_3_PCA: {elapsed_time:.2f} mins")

start_time = time.time()
results_KMSMOTE_AE_3_PCA = modelling_gs(dfs[2], "KMSMOTE_AE_3_PCA")

end_time = time.time()  # End timing
elapsed_time = (end_time - start_time) / 60
print("_____")
print(f" Total time taken by KMSMOTE_AE_3_PCA: {elapsed_time:.2f} mins")

start_time = time.time()
results_KMSMOTE_MICE_3_PCA = modelling_gs(dfs[3], "KMSMOTE_MICE_3_PCA")

end_time = time.time()  # End timing
elapsed_time = (end_time - start_time) / 60
print("_____")
```

```
print(f" Total time taken by KMSMOTE_MICE_3_PCA: {elapsed_time:.2f} mins")

start_time = time.time()
results_SVMSMOTE_AE_3_PCA = modelling_gs(dfs[4], "SVMSMOTE_AE_3_PCA")

end_time = time.time()  # End timing
elapsed_time = (end_time - start_time) / 60
print("_____")
print(f" Total time taken by SVMSMOTE_AE_3_PCA: {elapsed_time:.2f} mins")

start_time = time.time()
results_SVMSMOTE_MICE_3_PCA = modelling_gs(dfs[5], "SVMSMOTE_MICE_3_PCA")

end_time = time.time()  # End timing
elapsed_time = (end_time - start_time) / 60
print("_____")
print(f" Total time taken by SVMSMOTE_MICE_3_PCA: {elapsed_time:.2f} mins")


print(" ")
print("_____")
tot_end_time = time.time()  # End timing
tot_elapsed_time = (tot_end_time - tot_start_time) / 60
print(f" Total time taken by all the models : {tot_elapsed_time:.2f} mins")

# Print the results with variable names
print("Results for ADASYN_AE_3_PCA:", results_ADASYN_AE_3_PCA)
print("Results for ADASYN_MICE_3_PCA:", results_ADASYN_MICE_3_PCA)
print("Results for KMSMOTE_AE_3_PCA:", results_KMSMOTE_AE_3_PCA)
print("Results for KMSMOTE_MICE_3_PCA:", results_KMSMOTE_MICE_3_PCA)
print("Results for SVMSMOTE_AE_3_PCA:", results_SVMSMOTE_AE_3_PCA)
print("Results for SVMSMOTE_MICE_3_PCA:", results_SVMSMOTE_MICE_3_PCA)
```

```
2024-07-11 08:31:11,199 - INFO - Dataset has been split and returned
2024-07-11 08:31:11,205 - INFO - Data has been standardized
Datasets are read into dataframes
2024-07-11 08:34:06,839 - INFO - ANN has been trained in 175.63 seconds
2024-07-11 08:53:11,627 - INFO - RandomForest has been trained in 1144.79 seconds
2024-07-11 08:53:25,576 - INFO - XGBoost has been trained in 13.95 seconds
2024-07-11 09:12:31,895 - INFO - SVM has been trained in 1146.32 seconds
2024-07-11 09:12:32,570 - INFO - LogisticRegression has been trained in 0.68 seconds
2024-07-11 10:04:09,153 - INFO - GradientBoosting has been trained in 3096.58 seconds
2024-07-11 10:04:12,375 - INFO - KNN has been trained in 3.22 seconds
2024-07-11 10:04:12,382 - INFO - Naive Bayes has been trained in 0.01 seconds
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 624us/step
2024-07-11 10:04:15,766 - INFO - Models have been tested in 3.38 seconds
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 496us/step
```

```
2024-07-11 10:04:19,158 - INFO - Models have been evaluated in 3.39 seconds
2024-07-11 10:10:01,674 - INFO - SHAP explanations for RandomForest created and saved
2024-07-11 10:10:03,183 - INFO - SHAP explanations for XGBoost created and saved
2024-07-11 10:10:04,567 - INFO - SHAP explanations for SVM created and saved
2024-07-11 10:10:05,978 - INFO - SHAP explanations for LogisticRegression created and saved
2024-07-11 10:10:28,596 - INFO - SHAP explanations for GradientBoosting created and saved
2024-07-11 10:10:51,199 - INFO - SHAP explanations for KNN created and saved
2024-07-11 10:11:14,685 - INFO - SHAP explanations for NaiveBayes created and saved
2024-07-11 10:11:15,338 - INFO - LIME explanation for RandomForest created and saved
2024-07-11 10:11:15,769 - INFO - LIME explanation for XGBoost created and saved
2024-07-11 10:11:19,169 - INFO - LIME explanation for SVM created and saved
2024-07-11 10:11:19,511 - INFO - LIME explanation for LogisticRegression created and saved
2024-07-11 10:11:19,904 - INFO - LIME explanation for GradientBoosting created and saved
2024-07-11 10:11:20,419 - INFO - LIME explanation for KNN created and saved
2024-07-11 10:11:20,755 - INFO - LIME explanation for NaiveBayes created and saved
2024-07-11 10:11:20,756 - 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')`.
2024-07-11 10:11:20,822 - INFO - ANN model saved
2024-07-11 10:11:20,926 - INFO - RandomForest model saved
2024-07-11 10:11:20,938 - INFO - XGBoost model saved
2024-07-11 10:11:20,942 - INFO - SVM model saved
2024-07-11 10:11:20,945 - INFO - LogisticRegression model saved
2024-07-11 10:11:20,985 - INFO - GradientBoosting model saved
2024-07-11 10:11:20,990 - INFO - KNN model saved
2024-07-11 10:11:20,994 - INFO - NaiveBayes model saved
2024-07-11 10:11:20,995 - INFO - Models have been saved
2024-07-11 10:11:21,046 - INFO - Model interpretation summary created
2024-07-11 10:11:21,058 - INFO - Results have been documented.
2024-07-11 10:11:21,072 - INFO - Dataset has been split and returned
2024-07-11 10:11:21,083 - INFO - Data has been standardized
```

```
Model Interpretation Summary:

RandomForest Model:
Feature Importance from RandomForest Model:
                          Feature  Importance
             Leverage_Ratios_PC1    0.234203
      Cost_and_Expense_Ratios_PC1    0.170083
Liquidity_and_Coverage_Ratios_PC1    0.156748
      Cost_and_Expense_Ratios_PC2    0.100231
Liquidity_and_Coverage_Ratios_PC2    0.051738
             Leverage_Ratios_PC2    0.041782
         Profitability_Ratios_PC1    0.033791
            Cash_Flow_Ratios_PC1    0.029567
         Profitability_Ratios_PC2    0.026615
              Activity_Ratios_PC1    0.026388

XGBoost Model:
Feature Importance from XGBoost Model:
                          Feature  Importance
             Leverage_Ratios_PC1    0.429280
      Cost_and_Expense_Ratios_PC1    0.142012
Liquidity_and_Coverage_Ratios_PC1    0.055873
Liquidity_and_Coverage_Ratios_PC2    0.055459
              Activity_Ratios_PC2    0.033337
              Activity_Ratios_PC1    0.033284
            Cash_Flow_Ratios_PC1    0.032824
         Profitability_Ratios_PC2    0.030094
            Per_Share_Ratios_PC2    0.028540
         Profitability_Ratios_PC1    0.026827

SVM Model:
Feature Importance from SVM Model:
                          Feature Importance
Liquidity_and_Coverage_Ratios_PC1       None
Liquidity_and_Coverage_Ratios_PC2       None
             Leverage_Ratios_PC1       None
             Leverage_Ratios_PC2       None
              Activity_Ratios_PC1       None
              Activity_Ratios_PC2       None
         Profitability_Ratios_PC1       None
         Profitability_Ratios_PC2       None
      Cost_and_Expense_Ratios_PC1       None
      Cost_and_Expense_Ratios_PC2       None

LogisticRegression Model:
Feature Importance from LogisticRegression Model:
                          Feature  Importance
      Cost_and_Expense_Ratios_PC1    7.757142
         Profitability_Ratios_PC2    4.283041
            Per_Share_Ratios_PC1    3.294781
             Leverage_Ratios_PC1    1.825988
         Profitability_Ratios_PC1    1.739144
            Per_Share_Ratios_PC2    0.632684
Liquidity_and_Coverage_Ratios_PC2    0.175129
               Growth_Ratios_PC2   -0.047485
             Leverage_Ratios_PC2   -0.198037
            Cash_Flow_Ratios_PC2   -0.260373

GradientBoosting Model:
Feature Importance from GradientBoosting Model:
                          Feature  Importance
             Leverage_Ratios_PC1    0.511656
      Cost_and_Expense_Ratios_PC1    0.156245
Liquidity_and_Coverage_Ratios_PC2    0.063828
Liquidity_and_Coverage_Ratios_PC1    0.057982
         Profitability_Ratios_PC2    0.029997
              Activity_Ratios_PC1    0.027750
            Cash_Flow_Ratios_PC1    0.022171
         Profitability_Ratios_PC1    0.021451
               Growth_Ratios_PC1    0.019907
            Per_Share_Ratios_PC2    0.017677

KNN Model:
Feature Importance from KNN Model:
                          Feature Importance
```

```
Liquidity_and_Coverage_Ratios_PC1       None
Liquidity_and_Coverage_Ratios_PC2       None
              Leverage_Ratios_PC1       None
              Leverage_Ratios_PC2       None
              Activity_Ratios_PC1       None
              Activity_Ratios_PC2       None
         Profitability_Ratios_PC1       None
         Profitability_Ratios_PC2       None
      Cost_and_Expense_Ratios_PC1       None
      Cost_and_Expense_Ratios_PC2       None

NaiveBayes Model:
Feature Importance from NaiveBayes Model:
                          Feature Importance
Liquidity_and_Coverage_Ratios_PC1       None
Liquidity_and_Coverage_Ratios_PC2       None
              Leverage_Ratios_PC1       None
              Leverage_Ratios_PC2       None
              Activity_Ratios_PC1       None
              Activity_Ratios_PC2       None
         Profitability_Ratios_PC1       None
         Profitability_Ratios_PC2       None
      Cost_and_Expense_Ratios_PC1       None
      Cost_and_Expense_Ratios_PC2       None


_____________________________________________________________________________
 Total time taken by ADASYN_AE_3_PCA: 100.16 mins
2024-07-11 10:14:19,285 - INFO - ANN has been trained in 178.20 seconds
2024-07-11 10:34:25,161 - INFO - RandomForest has been trained in 1205.88 seconds
2024-07-11 10:34:41,048 - INFO - XGBoost has been trained in 15.89 seconds
2024-07-11 10:58:19,925 - INFO - SVM has been trained in 1418.88 seconds
2024-07-11 10:58:20,618 - INFO - LogisticRegression has been trained in 0.69 seconds
2024-07-11 11:49:17,087 - INFO - GradientBoosting has been trained in 3056.47 seconds
2024-07-11 11:49:20,226 - INFO - KNN has been trained in 3.14 seconds
2024-07-11 11:49:20,235 - INFO - Naive Bayes has been trained in 0.01 seconds
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 682us/step
2024-07-11 11:49:24,164 - INFO - Models have been tested in 3.93 seconds
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 528us/step
2024-07-11 11:49:27,624 - INFO - Models have been evaluated in 3.46 seconds
2024-07-11 11:55:20,267 - INFO - SHAP explanations for RandomForest created and saved
2024-07-11 11:55:21,661 - INFO - SHAP explanations for XGBoost created and saved
2024-07-11 11:55:22,993 - INFO - SHAP explanations for SVM created and saved
2024-07-11 11:55:24,332 - INFO - SHAP explanations for LogisticRegression created and saved
2024-07-11 11:55:45,181 - INFO - SHAP explanations for GradientBoosting created and saved
2024-07-11 11:56:06,101 - INFO - SHAP explanations for KNN created and saved
2024-07-11 11:56:26,945 - INFO - SHAP explanations for NaiveBayes created and saved
2024-07-11 11:56:27,387 - INFO - LIME explanation for RandomForest created and saved
2024-07-11 11:56:27,704 - INFO - LIME explanation for XGBoost created and saved
2024-07-11 11:56:30,411 - INFO - LIME explanation for SVM created and saved
2024-07-11 11:56:30,673 - INFO - LIME explanation for LogisticRegression created and saved
2024-07-11 11:56:30,985 - INFO - LIME explanation for GradientBoosting created and saved
2024-07-11 11:56:31,404 - INFO - LIME explanation for KNN created and saved
2024-07-11 11:56:31,673 - INFO - LIME explanation for NaiveBayes created and saved
2024-07-11 11:56:31,674 - 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')`.
2024-07-11 11:56:31,691 - INFO - ANN model saved
2024-07-11 11:56:31,780 - INFO - RandomForest model saved
2024-07-11 11:56:31,790 - INFO - XGBoost model saved
2024-07-11 11:56:31,793 - INFO - SVM model saved
2024-07-11 11:56:31,794 - INFO - LogisticRegression model saved
2024-07-11 11:56:31,823 - INFO - GradientBoosting model saved
2024-07-11 11:56:31,826 - INFO - KNN model saved
2024-07-11 11:56:31,827 - INFO - NaiveBayes model saved
2024-07-11 11:56:31,828 - INFO - Models have been saved
2024-07-11 11:56:31,860 - INFO - Model interpretation summary created
2024-07-11 11:56:31,869 - INFO - Results have been documented.
2024-07-11 11:56:31,885 - INFO - Dataset has been split and returned
2024-07-11 11:56:31,896 - INFO - Data has been standardized
```

```
Model Interpretation Summary:

RandomForest Model:
Feature Importance from RandomForest Model:
                          Feature  Importance
             Leverage_Ratios_PC1    0.236691
Liquidity_and_Coverage_Ratios_PC1    0.182210
      Cost_and_Expense_Ratios_PC1    0.171348
      Cost_and_Expense_Ratios_PC2    0.070511
Liquidity_and_Coverage_Ratios_PC2    0.053309
            Per_Share_Ratios_PC2    0.037249
         Profitability_Ratios_PC1    0.035276
              Activity_Ratios_PC1    0.031092
                Growth_Ratios_PC1    0.026876
            Cash_Flow_Ratios_PC2    0.025863

XGBoost Model:
Feature Importance from XGBoost Model:
                          Feature  Importance
             Leverage_Ratios_PC1    0.401111
      Cost_and_Expense_Ratios_PC1    0.139944
Liquidity_and_Coverage_Ratios_PC1    0.059314
Liquidity_and_Coverage_Ratios_PC2    0.046645
                Growth_Ratios_PC1    0.043735
            Per_Share_Ratios_PC1    0.042803
              Activity_Ratios_PC1    0.039137
            Per_Share_Ratios_PC2    0.033031
            Cash_Flow_Ratios_PC2    0.029865
         Profitability_Ratios_PC2    0.029408

SVM Model:
Feature Importance from SVM Model:
                          Feature Importance
Liquidity_and_Coverage_Ratios_PC1       None
Liquidity_and_Coverage_Ratios_PC2       None
             Leverage_Ratios_PC1        None
             Leverage_Ratios_PC2        None
             Activity_Ratios_PC1        None
             Activity_Ratios_PC2        None
         Profitability_Ratios_PC1       None
         Profitability_Ratios_PC2       None
      Cost_and_Expense_Ratios_PC1       None
      Cost_and_Expense_Ratios_PC2       None

LogisticRegression Model:
Feature Importance from LogisticRegression Model:
                          Feature  Importance
            Per_Share_Ratios_PC1    3.245274
         Profitability_Ratios_PC2    3.064337
             Leverage_Ratios_PC1    1.457914
         Profitability_Ratios_PC1    1.093241
Liquidity_and_Coverage_Ratios_PC2    0.165945
      Cost_and_Expense_Ratios_PC2    0.067829
             Leverage_Ratios_PC2   -0.006824
             Activity_Ratios_PC1   -0.067760
               Growth_Ratios_PC2   -0.142598
            Per_Share_Ratios_PC2   -0.181289

GradientBoosting Model:
Feature Importance from GradientBoosting Model:
                          Feature  Importance
             Leverage_Ratios_PC1    0.524207
      Cost_and_Expense_Ratios_PC1    0.138977
Liquidity_and_Coverage_Ratios_PC2    0.054066
Liquidity_and_Coverage_Ratios_PC1    0.052351
         Profitability_Ratios_PC2    0.031918
                Growth_Ratios_PC1    0.029270
              Activity_Ratios_PC1    0.027034
            Cash_Flow_Ratios_PC2    0.026836
         Profitability_Ratios_PC1    0.025801
            Per_Share_Ratios_PC2    0.020553

KNN Model:
Feature Importance from KNN Model:
                          Feature Importance
```

```
Liquidity_and_Coverage_Ratios_PC1       None
Liquidity_and_Coverage_Ratios_PC2       None
              Leverage_Ratios_PC1       None
              Leverage_Ratios_PC2       None
              Activity_Ratios_PC1       None
              Activity_Ratios_PC2       None
         Profitability_Ratios_PC1       None
         Profitability_Ratios_PC2       None
      Cost_and_Expense_Ratios_PC1       None
      Cost_and_Expense_Ratios_PC2       None

NaiveBayes Model:
Feature Importance from NaiveBayes Model:
                          Feature Importance
Liquidity_and_Coverage_Ratios_PC1       None
Liquidity_and_Coverage_Ratios_PC2       None
              Leverage_Ratios_PC1       None
              Leverage_Ratios_PC2       None
              Activity_Ratios_PC1       None
              Activity_Ratios_PC2       None
         Profitability_Ratios_PC1       None
         Profitability_Ratios_PC2       None
      Cost_and_Expense_Ratios_PC1       None
      Cost_and_Expense_Ratios_PC2       None


_____________________________________________________________________________
 Total time taken by ADASYN_MICE_3_PCA: 105.18 mins
2024-07-11 11:59:23,819 - INFO - ANN has been trained in 171.92 seconds
2024-07-11 12:17:58,999 - INFO - RandomForest has been trained in 1115.18 seconds
2024-07-11 12:18:12,724 - INFO - XGBoost has been trained in 13.72 seconds
2024-07-11 12:33:01,294 - INFO - SVM has been trained in 888.57 seconds
2024-07-11 12:33:01,994 - INFO - LogisticRegression has been trained in 0.70 seconds
2024-07-11 13:23:38,074 - INFO - GradientBoosting has been trained in 3036.08 seconds
2024-07-11 13:23:41,140 - INFO - KNN has been trained in 3.07 seconds
2024-07-11 13:23:41,147 - INFO - Naive Bayes has been trained in 0.01 seconds
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 881us/step
2024-07-11 13:23:43,739 - INFO - Models have been tested in 2.59 seconds
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 531us/step
2024-07-11 13:23:46,391 - INFO - Models have been evaluated in 2.65 seconds
2024-07-11 13:26:56,668 - INFO - SHAP explanations for RandomForest created and saved
2024-07-11 13:26:58,125 - INFO - SHAP explanations for XGBoost created and saved
2024-07-11 13:26:59,509 - INFO - SHAP explanations for SVM created and saved
2024-07-11 13:27:00,886 - INFO - SHAP explanations for LogisticRegression created and saved
2024-07-11 13:27:21,886 - INFO - SHAP explanations for GradientBoosting created and saved
2024-07-11 13:27:42,893 - INFO - SHAP explanations for KNN created and saved
2024-07-11 13:28:03,824 - INFO - SHAP explanations for NaiveBayes created and saved
2024-07-11 13:28:04,239 - INFO - LIME explanation for RandomForest created and saved
2024-07-11 13:28:04,538 - INFO - LIME explanation for XGBoost created and saved
2024-07-11 13:28:06,698 - INFO - LIME explanation for SVM created and saved
2024-07-11 13:28:07,156 - INFO - LIME explanation for LogisticRegression created and saved
2024-07-11 13:28:07,485 - INFO - LIME explanation for GradientBoosting created and saved
2024-07-11 13:28:07,902 - INFO - LIME explanation for KNN created and saved
2024-07-11 13:28:08,177 - INFO - LIME explanation for NaiveBayes created and saved
2024-07-11 13:28:08,177 - 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')`.
2024-07-11 13:28:08,194 - INFO - ANN model saved
2024-07-11 13:28:08,256 - INFO - RandomForest model saved
2024-07-11 13:28:08,265 - INFO - XGBoost model saved
2024-07-11 13:28:08,268 - INFO - SVM model saved
2024-07-11 13:28:08,270 - INFO - LogisticRegression model saved
2024-07-11 13:28:08,299 - INFO - GradientBoosting model saved
2024-07-11 13:28:08,303 - INFO - KNN model saved
2024-07-11 13:28:08,305 - INFO - NaiveBayes model saved
2024-07-11 13:28:08,306 - INFO - Models have been saved
2024-07-11 13:28:08,332 - INFO - Model interpretation summary created
2024-07-11 13:28:08,335 - INFO - Results have been documented.
2024-07-11 13:28:08,353 - INFO - Dataset has been split and returned
2024-07-11 13:28:08,364 - INFO - Data has been standardized
```

```
Model Interpretation Summary:

RandomForest Model:
Feature Importance from RandomForest Model:
                          Feature  Importance
             Leverage_Ratios_PC1    0.307824
Liquidity_and_Coverage_Ratios_PC1    0.183045
      Cost_and_Expense_Ratios_PC1    0.132522
      Cost_and_Expense_Ratios_PC2    0.075325
Liquidity_and_Coverage_Ratios_PC2    0.061870
         Profitability_Ratios_PC1    0.035834
             Cash_Flow_Ratios_PC2    0.033610
             Cash_Flow_Ratios_PC1    0.027903
              Activity_Ratios_PC2    0.026189
             Per_Share_Ratios_PC1    0.023271

XGBoost Model:
Feature Importance from XGBoost Model:
                          Feature  Importance
             Leverage_Ratios_PC1    0.606076
      Cost_and_Expense_Ratios_PC1    0.075337
Liquidity_and_Coverage_Ratios_PC1    0.039747
             Cash_Flow_Ratios_PC2    0.032893
              Activity_Ratios_PC2    0.032756
Liquidity_and_Coverage_Ratios_PC2    0.032008
              Activity_Ratios_PC1    0.025009
             Per_Share_Ratios_PC2    0.023531
         Profitability_Ratios_PC2    0.022202
             Per_Share_Ratios_PC1    0.021892

SVM Model:
Feature Importance from SVM Model:
                          Feature Importance
Liquidity_and_Coverage_Ratios_PC1       None
Liquidity_and_Coverage_Ratios_PC2       None
             Leverage_Ratios_PC1        None
             Leverage_Ratios_PC2        None
              Activity_Ratios_PC1       None
              Activity_Ratios_PC2       None
         Profitability_Ratios_PC1       None
         Profitability_Ratios_PC2       None
      Cost_and_Expense_Ratios_PC1       None
      Cost_and_Expense_Ratios_PC2       None

LogisticRegression Model:
Feature Importance from LogisticRegression Model:
                          Feature  Importance
      Cost_and_Expense_Ratios_PC1    5.997802
         Profitability_Ratios_PC2    5.744439
              Leverage_Ratios_PC1    2.677239
             Per_Share_Ratios_PC1    1.701508
         Profitability_Ratios_PC1    1.637186
             Per_Share_Ratios_PC2    0.320759
             Cash_Flow_Ratios_PC2    0.271707
              Leverage_Ratios_PC2    0.045888
                Growth_Ratios_PC2    0.024765
Liquidity_and_Coverage_Ratios_PC2   -0.204850

GradientBoosting Model:
Feature Importance from GradientBoosting Model:
                          Feature  Importance
             Leverage_Ratios_PC1    0.678253
      Cost_and_Expense_Ratios_PC1    0.072998
             Cash_Flow_Ratios_PC2    0.047987
Liquidity_and_Coverage_Ratios_PC1    0.047915
Liquidity_and_Coverage_Ratios_PC2    0.024636
              Activity_Ratios_PC2    0.016701
         Profitability_Ratios_PC2    0.014970
         Profitability_Ratios_PC1    0.014931
             Cash_Flow_Ratios_PC1    0.014924
             Per_Share_Ratios_PC2    0.013983

KNN Model:
Feature Importance from KNN Model:
                          Feature Importance
```

```
Liquidity_and_Coverage_Ratios_PC1       None
Liquidity_and_Coverage_Ratios_PC2       None
              Leverage_Ratios_PC1       None
              Leverage_Ratios_PC2       None
              Activity_Ratios_PC1       None
              Activity_Ratios_PC2       None
         Profitability_Ratios_PC1       None
         Profitability_Ratios_PC2       None
      Cost_and_Expense_Ratios_PC1       None
      Cost_and_Expense_Ratios_PC2       None

NaiveBayes Model:
Feature Importance from NaiveBayes Model:
                          Feature Importance
Liquidity_and_Coverage_Ratios_PC1       None
Liquidity_and_Coverage_Ratios_PC2       None
              Leverage_Ratios_PC1       None
              Leverage_Ratios_PC2       None
              Activity_Ratios_PC1       None
              Activity_Ratios_PC2       None
         Profitability_Ratios_PC1       None
         Profitability_Ratios_PC2       None
      Cost_and_Expense_Ratios_PC1       None
      Cost_and_Expense_Ratios_PC2       None


_____________________________________________________________________________
 Total time taken by KMSMOTE_AE_3_PCA: 91.61 mins
2024-07-11 13:31:05,462 - INFO - ANN has been trained in 177.10 seconds
2024-07-11 13:50:36,387 - INFO - RandomForest has been trained in 1170.92 seconds
2024-07-11 13:50:50,082 - INFO - XGBoost has been trained in 13.69 seconds
2024-07-11 14:06:06,416 - INFO - SVM has been trained in 916.33 seconds
2024-07-11 14:06:07,081 - INFO - LogisticRegression has been trained in 0.66 seconds
2024-07-11 14:57:28,303 - INFO - GradientBoosting has been trained in 3081.22 seconds
2024-07-11 14:57:31,468 - INFO - KNN has been trained in 3.16 seconds
2024-07-11 14:57:31,479 - INFO - Naive Bayes has been trained in 0.01 seconds
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 723us/step
2024-07-11 14:57:34,237 - INFO - Models have been tested in 2.76 seconds
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 513us/step
2024-07-11 14:57:36,949 - INFO - Models have been evaluated in 2.71 seconds
2024-07-11 15:02:28,396 - INFO - SHAP explanations for RandomForest created and saved
2024-07-11 15:02:29,777 - INFO - SHAP explanations for XGBoost created and saved
2024-07-11 15:02:31,095 - INFO - SHAP explanations for SVM created and saved
2024-07-11 15:02:32,418 - INFO - SHAP explanations for LogisticRegression created and saved
2024-07-11 15:02:54,163 - INFO - SHAP explanations for GradientBoosting created and saved
2024-07-11 15:03:15,874 - INFO - SHAP explanations for KNN created and saved
2024-07-11 15:03:37,514 - INFO - SHAP explanations for NaiveBayes created and saved
2024-07-11 15:03:37,961 - INFO - LIME explanation for RandomForest created and saved
2024-07-11 15:03:38,266 - INFO - LIME explanation for XGBoost created and saved
2024-07-11 15:03:40,400 - INFO - LIME explanation for SVM created and saved
2024-07-11 15:03:40,671 - INFO - LIME explanation for LogisticRegression created and saved
2024-07-11 15:03:40,991 - INFO - LIME explanation for GradientBoosting created and saved
2024-07-11 15:03:41,412 - INFO - LIME explanation for KNN created and saved
2024-07-11 15:03:41,687 - INFO - LIME explanation for NaiveBayes created and saved
2024-07-11 15:03:41,688 - 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')`.
2024-07-11 15:03:41,713 - INFO - ANN model saved
2024-07-11 15:03:41,804 - INFO - RandomForest model saved
2024-07-11 15:03:41,814 - INFO - XGBoost model saved
2024-07-11 15:03:41,817 - INFO - SVM model saved
2024-07-11 15:03:41,819 - INFO - LogisticRegression model saved
2024-07-11 15:03:41,848 - INFO - GradientBoosting model saved
2024-07-11 15:03:41,852 - INFO - KNN model saved
2024-07-11 15:03:41,853 - INFO - NaiveBayes model saved
2024-07-11 15:03:41,854 - INFO - Models have been saved
2024-07-11 15:03:41,890 - INFO - Model interpretation summary created
2024-07-11 15:03:41,915 - INFO - Results have been documented.
2024-07-11 15:03:41,931 - INFO - Dataset has been split and returned
2024-07-11 15:03:41,942 - INFO - Data has been standardized
```

```
Model Interpretation Summary:

RandomForest Model:
Feature Importance from RandomForest Model:
                          Feature  Importance
             Leverage_Ratios_PC1    0.287167
Liquidity_and_Coverage_Ratios_PC1    0.205343
      Cost_and_Expense_Ratios_PC1    0.139605
Liquidity_and_Coverage_Ratios_PC2    0.061355
      Cost_and_Expense_Ratios_PC2    0.060887
            Per_Share_Ratios_PC2    0.035925
         Profitability_Ratios_PC1    0.035466
             Cash_Flow_Ratios_PC2    0.033167
              Activity_Ratios_PC1    0.025172
             Cash_Flow_Ratios_PC1    0.021792

XGBoost Model:
Feature Importance from XGBoost Model:
                          Feature  Importance
             Leverage_Ratios_PC1    0.604689
      Cost_and_Expense_Ratios_PC1    0.093743
Liquidity_and_Coverage_Ratios_PC1    0.035761
            Per_Share_Ratios_PC1    0.031384
             Cash_Flow_Ratios_PC2    0.030782
Liquidity_and_Coverage_Ratios_PC2    0.028183
            Per_Share_Ratios_PC2    0.027109
              Activity_Ratios_PC1    0.025653
              Activity_Ratios_PC2    0.021720
                Growth_Ratios_PC1    0.019134

SVM Model:
Feature Importance from SVM Model:
                          Feature Importance
Liquidity_and_Coverage_Ratios_PC1       None
Liquidity_and_Coverage_Ratios_PC2       None
             Leverage_Ratios_PC1        None
             Leverage_Ratios_PC2        None
              Activity_Ratios_PC1       None
              Activity_Ratios_PC2       None
         Profitability_Ratios_PC1       None
         Profitability_Ratios_PC2       None
      Cost_and_Expense_Ratios_PC1       None
      Cost_and_Expense_Ratios_PC2       None

LogisticRegression Model:
Feature Importance from LogisticRegression Model:
                          Feature  Importance
         Profitability_Ratios_PC2    4.362338
             Per_Share_Ratios_PC1    3.693510
              Leverage_Ratios_PC1    2.355179
         Profitability_Ratios_PC1    0.696369
             Cash_Flow_Ratios_PC2    0.156147
              Leverage_Ratios_PC2    0.047282
                Growth_Ratios_PC2   -0.116730
Liquidity_and_Coverage_Ratios_PC2   -0.207306
              Activity_Ratios_PC1   -0.215940
             Per_Share_Ratios_PC2   -0.217398

GradientBoosting Model:
Feature Importance from GradientBoosting Model:
                          Feature  Importance
             Leverage_Ratios_PC1    0.674061
      Cost_and_Expense_Ratios_PC1    0.089625
             Cash_Flow_Ratios_PC2    0.046111
Liquidity_and_Coverage_Ratios_PC1    0.029470
              Activity_Ratios_PC2    0.023156
Liquidity_and_Coverage_Ratios_PC2    0.021903
             Per_Share_Ratios_PC2    0.019182
         Profitability_Ratios_PC1    0.017872
              Activity_Ratios_PC1    0.017478
         Profitability_Ratios_PC2    0.013321

KNN Model:
Feature Importance from KNN Model:
                          Feature Importance
```

```
Liquidity_and_Coverage_Ratios_PC1       None
Liquidity_and_Coverage_Ratios_PC2       None
              Leverage_Ratios_PC1       None
              Leverage_Ratios_PC2       None
              Activity_Ratios_PC1       None
              Activity_Ratios_PC2       None
         Profitability_Ratios_PC1       None
         Profitability_Ratios_PC2       None
      Cost_and_Expense_Ratios_PC1       None
      Cost_and_Expense_Ratios_PC2       None

NaiveBayes Model:
Feature Importance from NaiveBayes Model:
                          Feature Importance
Liquidity_and_Coverage_Ratios_PC1       None
Liquidity_and_Coverage_Ratios_PC2       None
              Leverage_Ratios_PC1       None
              Leverage_Ratios_PC2       None
              Activity_Ratios_PC1       None
              Activity_Ratios_PC2       None
         Profitability_Ratios_PC1       None
         Profitability_Ratios_PC2       None
      Cost_and_Expense_Ratios_PC1       None
      Cost_and_Expense_Ratios_PC2       None


_____________________________________________________________________________
 Total time taken by KMSMOTE_MICE_3_PCA: 95.56 mins
2024-07-11 15:06:34,493 - INFO - ANN has been trained in 172.55 seconds
2024-07-11 15:23:55,027 - INFO - RandomForest has been trained in 1040.53 seconds
2024-07-11 15:24:08,573 - INFO - XGBoost has been trained in 13.54 seconds
2024-07-11 15:37:29,695 - INFO - SVM has been trained in 801.12 seconds
2024-07-11 15:37:30,340 - INFO - LogisticRegression has been trained in 0.64 seconds
2024-07-11 16:28:03,706 - INFO - GradientBoosting has been trained in 3033.36 seconds
2024-07-11 16:28:06,836 - INFO - KNN has been trained in 3.13 seconds
2024-07-11 16:28:06,844 - INFO - Naive Bayes has been trained in 0.01 seconds
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 618us/step
2024-07-11 16:28:09,236 - INFO - Models have been tested in 2.39 seconds
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 531us/step
2024-07-11 16:28:11,819 - INFO - Models have been evaluated in 2.58 seconds
2024-07-11 16:29:31,045 - INFO - SHAP explanations for RandomForest created and saved
2024-07-11 16:29:32,408 - INFO - SHAP explanations for XGBoost created and saved
2024-07-11 16:29:33,720 - INFO - SHAP explanations for SVM created and saved
2024-07-11 16:29:35,037 - INFO - SHAP explanations for LogisticRegression created and saved
2024-07-11 16:29:55,617 - INFO - SHAP explanations for GradientBoosting created and saved
2024-07-11 16:30:16,003 - INFO - SHAP explanations for KNN created and saved
2024-07-11 16:30:36,376 - INFO - SHAP explanations for NaiveBayes created and saved
2024-07-11 16:30:36,741 - INFO - LIME explanation for RandomForest created and saved
2024-07-11 16:30:37,043 - INFO - LIME explanation for XGBoost created and saved
2024-07-11 16:30:39,509 - INFO - LIME explanation for SVM created and saved
2024-07-11 16:30:39,770 - INFO - LIME explanation for LogisticRegression created and saved
2024-07-11 16:30:40,087 - INFO - LIME explanation for GradientBoosting created and saved
2024-07-11 16:30:40,495 - INFO - LIME explanation for KNN created and saved
2024-07-11 16:30:40,763 - INFO - LIME explanation for NaiveBayes created and saved
2024-07-11 16:30:40,764 - 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')`.
2024-07-11 16:30:40,782 - INFO - ANN model saved
2024-07-11 16:30:40,815 - INFO - RandomForest model saved
2024-07-11 16:30:40,824 - INFO - XGBoost model saved
2024-07-11 16:30:40,827 - INFO - SVM model saved
2024-07-11 16:30:40,829 - INFO - LogisticRegression model saved
2024-07-11 16:30:40,858 - INFO - GradientBoosting model saved
2024-07-11 16:30:40,862 - INFO - KNN model saved
2024-07-11 16:30:40,863 - INFO - NaiveBayes model saved
2024-07-11 16:30:40,864 - INFO - Models have been saved
2024-07-11 16:30:40,884 - INFO - Model interpretation summary created
2024-07-11 16:30:40,886 - INFO - Results have been documented.
2024-07-11 16:30:40,902 - INFO - Dataset has been split and returned
2024-07-11 16:30:40,912 - INFO - Data has been standardized
```

```
Model Interpretation Summary:

RandomForest Model:
Feature Importance from RandomForest Model:
                          Feature  Importance
             Leverage_Ratios_PC1    0.256740
Liquidity_and_Coverage_Ratios_PC1    0.239777
      Cost_and_Expense_Ratios_PC1    0.111281
Liquidity_and_Coverage_Ratios_PC2    0.084551
      Cost_and_Expense_Ratios_PC2    0.071137
            Cash_Flow_Ratios_PC2    0.037640
            Cash_Flow_Ratios_PC1    0.036886
         Profitability_Ratios_PC1    0.036248
              Activity_Ratios_PC1    0.020348
              Activity_Ratios_PC2    0.020314

XGBoost Model:
Feature Importance from XGBoost Model:
                          Feature  Importance
             Leverage_Ratios_PC1    0.523971
Liquidity_and_Coverage_Ratios_PC1    0.079233
      Cost_and_Expense_Ratios_PC1    0.074354
Liquidity_and_Coverage_Ratios_PC2    0.045701
            Cash_Flow_Ratios_PC1    0.039600
            Cash_Flow_Ratios_PC2    0.033878
      Cost_and_Expense_Ratios_PC2    0.028859
         Profitability_Ratios_PC1    0.026411
              Activity_Ratios_PC1    0.026028
            Per_Share_Ratios_PC1    0.022965

SVM Model:
Feature Importance from SVM Model:
                          Feature Importance
Liquidity_and_Coverage_Ratios_PC1       None
Liquidity_and_Coverage_Ratios_PC2       None
             Leverage_Ratios_PC1       None
             Leverage_Ratios_PC2       None
             Activity_Ratios_PC1       None
             Activity_Ratios_PC2       None
         Profitability_Ratios_PC1       None
         Profitability_Ratios_PC2       None
      Cost_and_Expense_Ratios_PC1       None
      Cost_and_Expense_Ratios_PC2       None

LogisticRegression Model:
Feature Importance from LogisticRegression Model:
                          Feature  Importance
             Activity_Ratios_PC2   15.374160
            Per_Share_Ratios_PC1    5.067911
         Profitability_Ratios_PC2    4.082284
              Leverage_Ratios_PC1    2.656709
         Profitability_Ratios_PC1    1.513397
            Per_Share_Ratios_PC2    1.172755
            Cash_Flow_Ratios_PC2    1.055348
               Growth_Ratios_PC2    0.117733
             Leverage_Ratios_PC2   -0.011707
Liquidity_and_Coverage_Ratios_PC2   -0.405275

GradientBoosting Model:
Feature Importance from GradientBoosting Model:
                          Feature  Importance
             Leverage_Ratios_PC1    0.645841
Liquidity_and_Coverage_Ratios_PC1    0.076339
      Cost_and_Expense_Ratios_PC1    0.056755
Liquidity_and_Coverage_Ratios_PC2    0.051936
            Cash_Flow_Ratios_PC1    0.030002
            Cash_Flow_Ratios_PC2    0.029184
         Profitability_Ratios_PC1    0.021687
         Profitability_Ratios_PC2    0.013848
              Activity_Ratios_PC2    0.012675
              Activity_Ratios_PC1    0.012428

KNN Model:
Feature Importance from KNN Model:
                          Feature Importance
```

```
Liquidity_and_Coverage_Ratios_PC1       None
Liquidity_and_Coverage_Ratios_PC2       None
              Leverage_Ratios_PC1       None
              Leverage_Ratios_PC2       None
              Activity_Ratios_PC1       None
              Activity_Ratios_PC2       None
         Profitability_Ratios_PC1       None
         Profitability_Ratios_PC2       None
      Cost_and_Expense_Ratios_PC1       None
      Cost_and_Expense_Ratios_PC2       None

NaiveBayes Model:
Feature Importance from NaiveBayes Model:
                          Feature Importance
Liquidity_and_Coverage_Ratios_PC1       None
Liquidity_and_Coverage_Ratios_PC2       None
              Leverage_Ratios_PC1       None
              Leverage_Ratios_PC2       None
              Activity_Ratios_PC1       None
              Activity_Ratios_PC2       None
         Profitability_Ratios_PC1       None
         Profitability_Ratios_PC2       None
      Cost_and_Expense_Ratios_PC1       None
      Cost_and_Expense_Ratios_PC2       None


_____________________________________________________________________________
 Total time taken by SVMSMOTE_AE_3_PCA: 86.98 mins
2024-07-11 16:33:33,160 - INFO - ANN has been trained in 172.25 seconds
2024-07-11 16:50:54,337 - INFO - RandomForest has been trained in 1041.18 seconds
2024-07-11 16:51:07,548 - INFO - XGBoost has been trained in 13.21 seconds
2024-07-11 17:04:32,283 - INFO - SVM has been trained in 804.73 seconds
2024-07-11 17:04:33,091 - INFO - LogisticRegression has been trained in 0.81 seconds
2024-07-11 17:54:30,627 - INFO - GradientBoosting has been trained in 2997.53 seconds
2024-07-11 17:54:33,744 - INFO - KNN has been trained in 3.12 seconds
2024-07-11 17:54:33,753 - INFO - Naive Bayes has been trained in 0.01 seconds
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 659us/step
2024-07-11 17:54:36,206 - INFO - Models have been tested in 2.45 seconds
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 502us/step
2024-07-11 17:54:38,736 - INFO - Models have been evaluated in 2.53 seconds
2024-07-11 17:56:01,491 - INFO - SHAP explanations for RandomForest created and saved
2024-07-11 17:56:02,823 - INFO - SHAP explanations for XGBoost created and saved
2024-07-11 17:56:04,094 - INFO - SHAP explanations for SVM created and saved
2024-07-11 17:56:05,346 - INFO - SHAP explanations for LogisticRegression created and saved
2024-07-11 17:56:25,363 - INFO - SHAP explanations for GradientBoosting created and saved
2024-07-11 17:56:45,483 - INFO - SHAP explanations for KNN created and saved
2024-07-11 17:57:05,563 - INFO - SHAP explanations for NaiveBayes created and saved
2024-07-11 17:57:05,942 - INFO - LIME explanation for RandomForest created and saved
2024-07-11 17:57:06,242 - INFO - LIME explanation for XGBoost created and saved
2024-07-11 17:57:08,228 - INFO - LIME explanation for SVM created and saved
2024-07-11 17:57:08,492 - INFO - LIME explanation for LogisticRegression created and saved
2024-07-11 17:57:08,809 - INFO - LIME explanation for GradientBoosting created and saved
2024-07-11 17:57:09,223 - INFO - LIME explanation for KNN created and saved
2024-07-11 17:57:09,488 - INFO - LIME explanation for NaiveBayes created and saved
2024-07-11 17:57:09,489 - 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')`.
2024-07-11 17:57:09,505 - INFO - ANN model saved
2024-07-11 17:57:09,537 - INFO - RandomForest model saved
2024-07-11 17:57:09,546 - INFO - XGBoost model saved
2024-07-11 17:57:09,549 - INFO - SVM model saved
2024-07-11 17:57:09,551 - INFO - LogisticRegression model saved
2024-07-11 17:57:09,581 - INFO - GradientBoosting model saved
2024-07-11 17:57:09,585 - INFO - KNN model saved
2024-07-11 17:57:09,587 - INFO - NaiveBayes model saved
2024-07-11 17:57:09,587 - INFO - Models have been saved
2024-07-11 17:57:09,608 - INFO - Model interpretation summary created
2024-07-11 17:57:09,610 - INFO - Results have been documented.
```

```
Model Interpretation Summary:

RandomForest Model:
Feature Importance from RandomForest Model:
                          Feature  Importance
             Leverage_Ratios_PC1    0.275064
Liquidity_and_Coverage_Ratios_PC1    0.242968
      Cost_and_Expense_Ratios_PC1    0.125846
Liquidity_and_Coverage_Ratios_PC2    0.078226
      Cost_and_Expense_Ratios_PC2    0.047307
         Profitability_Ratios_PC1    0.037334
             Cash_Flow_Ratios_PC1    0.033399
             Cash_Flow_Ratios_PC2    0.030279
              Activity_Ratios_PC2    0.023490
             Per_Share_Ratios_PC2    0.020633

XGBoost Model:
Feature Importance from XGBoost Model:
                          Feature  Importance
             Leverage_Ratios_PC1    0.515313
Liquidity_and_Coverage_Ratios_PC1    0.094884
      Cost_and_Expense_Ratios_PC1    0.071955
Liquidity_and_Coverage_Ratios_PC2    0.053566
             Cash_Flow_Ratios_PC1    0.041987
             Per_Share_Ratios_PC2    0.033123
             Cash_Flow_Ratios_PC2    0.025626
             Per_Share_Ratios_PC1    0.024225
              Activity_Ratios_PC2    0.021327
              Activity_Ratios_PC1    0.020635

SVM Model:
Feature Importance from SVM Model:
                          Feature Importance
Liquidity_and_Coverage_Ratios_PC1       None
Liquidity_and_Coverage_Ratios_PC2       None
             Leverage_Ratios_PC1        None
             Leverage_Ratios_PC2        None
             Activity_Ratios_PC1        None
             Activity_Ratios_PC2        None
         Profitability_Ratios_PC1       None
         Profitability_Ratios_PC2       None
      Cost_and_Expense_Ratios_PC1       None
      Cost_and_Expense_Ratios_PC2       None

LogisticRegression Model:
Feature Importance from LogisticRegression Model:
                          Feature  Importance
         Profitability_Ratios_PC2    4.046635
             Per_Share_Ratios_PC1    3.474581
              Leverage_Ratios_PC1    2.353163
      Cost_and_Expense_Ratios_PC2    1.087933
         Profitability_Ratios_PC1    0.990705
             Cash_Flow_Ratios_PC2    0.606652
              Leverage_Ratios_PC2    0.038675
                Growth_Ratios_PC2   -0.161846
             Per_Share_Ratios_PC2   -0.203632
Liquidity_and_Coverage_Ratios_PC2   -0.432175

GradientBoosting Model:
Feature Importance from GradientBoosting Model:
                          Feature  Importance
             Leverage_Ratios_PC1    0.657207
Liquidity_and_Coverage_Ratios_PC1    0.081874
      Cost_and_Expense_Ratios_PC1    0.058239
Liquidity_and_Coverage_Ratios_PC2    0.056355
             Cash_Flow_Ratios_PC1    0.028875
         Profitability_Ratios_PC1    0.017441
             Cash_Flow_Ratios_PC2    0.016426
              Activity_Ratios_PC1    0.014149
              Activity_Ratios_PC2    0.012347
      Cost_and_Expense_Ratios_PC2    0.010751

KNN Model:
Feature Importance from KNN Model:
                          Feature Importance
```

```
Liquidity_and_Coverage_Ratios_PC1       None
Liquidity_and_Coverage_Ratios_PC2       None
              Leverage_Ratios_PC1       None
              Leverage_Ratios_PC2       None
              Activity_Ratios_PC1       None
              Activity_Ratios_PC2       None
         Profitability_Ratios_PC1       None
         Profitability_Ratios_PC2       None
      Cost_and_Expense_Ratios_PC1       None
      Cost_and_Expense_Ratios_PC2       None

NaiveBayes Model:
Feature Importance from NaiveBayes Model:
                          Feature Importance
Liquidity_and_Coverage_Ratios_PC1       None
Liquidity_and_Coverage_Ratios_PC2       None
              Leverage_Ratios_PC1       None
              Leverage_Ratios_PC2       None
              Activity_Ratios_PC1       None
              Activity_Ratios_PC2       None
         Profitability_Ratios_PC1       None
         Profitability_Ratios_PC2       None
      Cost_and_Expense_Ratios_PC1       None
      Cost_and_Expense_Ratios_PC2       None


_____________________________________________________________________________
 Total time taken by SVMSMOTE_MICE_3_PCA: 86.48 mins

_____________________________________________________________________________
 Total time taken by all the models : 565.97 mins
Results for ADASYN_AE_3_PCA: {'ANN': {'accuracy': 0.9773846434433704, 'confusion_matrix': array([[2634,  108],
       [  16, 2725]], dtype=int64), 'f1_score': 0.9777538571941156, 'precision': 0.9618778679844687, 'recall': 0.9941
627143378329, 'auc_roc': 0.9931587656014206}, 'RandomForest': {'accuracy': 0.990333758891118, 'confusion_matrix': arr
ay([[2694,   48],
       [   5, 2736]], dtype=int64), 'f1_score': 0.9904072398190045, 'precision': 0.9827586206896551, 'recall': 0.9981
758482305728, 'auc_roc': 0.9997875814515031}, 'XGBoost': {'accuracy': 0.9930694875068393, 'confusion_matrix': array
([[2705,   37],
       [   1, 2740]], dtype=int64), 'f1_score': 0.9931134469010511, 'precision': 0.9866762693554195, 'recall': 0.9996
351696461145, 'auc_roc': 0.999748796605348}, 'SVM': {'accuracy': 0.9182929053437899, 'confusion_matrix': array([[243
1,  311],
       [ 137, 2604]], dtype=int64), 'f1_score': 0.9207920792079208, 'precision': 0.8933104631217839, 'recall': 0.9500
182415176943, 'auc_roc': 0.9517697465426936}, 'LogisticRegression': {'accuracy': 0.8670435892759438, 'confusion_matri
x': array([[2249,  493],
       [ 236, 2505]], dtype=int64), 'f1_score': 0.872974385781495, 'precision': 0.8355570380253502, 'recall': 0.91390
00364830354, 'auc_roc': 0.9060929729309715}, 'GradientBoosting': {'accuracy': 0.9914280503374066, 'confusion_matrix':
array([[2698,   44],
       [   3, 2738]], dtype=int64), 'f1_score': 0.9914901321745428, 'precision': 0.9841840402588066, 'recall': 0.9989
055089383436, 'auc_roc': 0.999739482920165}, 'KNN': {'accuracy': 0.9835856283056721, 'confusion_matrix': array([[265
3,   89],
       [   1, 2740]], dtype=int64), 'f1_score': 0.9838420107719928, 'precision': 0.9685401201838105, 'recall': 0.9996
351696461145, 'auc_roc': 0.9928478880952742}, 'NaiveBayes': {'accuracy': 0.5363851905890935, 'confusion_matrix': arra
y([[ 307, 2435],
       [ 107, 2634]], dtype=int64), 'f1_score': 0.6745198463508323, 'precision': 0.5196291181692642, 'recall': 0.9609
631521342575, 'auc_roc': 0.7015488126248866}}
Results for ADASYN_MICE_3_PCA: {'ANN': {'accuracy': 0.9746442904049617, 'confusion_matrix': array([[2618,  124],
       [  15, 2725]], dtype=int64), 'f1_score': 0.9751297190910717, 'precision': 0.9564759564759565, 'recall': 0.9945
255474452555, 'auc_roc': 0.9906930979039222}, 'RandomForest': {'accuracy': 0.9903319956220358, 'confusion_matrix': ar
ray([[2689,   53],
       [   0, 2740]], dtype=int64), 'f1_score': 0.990421109705404, 'precision': 0.9810239885427855, 'recall': 1.0, 'a
uc_roc': 0.9997555596373258}, 'XGBoost': {'accuracy': 0.9927033929222912, 'confusion_matrix': array([[2702,   40],
       [   0, 2740]], dtype=int64), 'f1_score': 0.9927536231884058, 'precision': 0.9856115107913669, 'recall': 1.0,
'auc_roc': 0.9998087335686563}, 'SVM': {'accuracy': 0.91353520612915, 'confusion_matrix': array([[2431,  311],
       [ 163, 2577]], dtype=int64), 'f1_score': 0.9157782515991472, 'precision': 0.8923130193905817, 'recall': 0.9405
109489051094, 'auc_roc': 0.9508714135880357}, 'LogisticRegression': {'accuracy': 0.850601970083911, 'confusion_matri
x': array([[2198,  544],
       [ 275, 2465]], dtype=int64), 'f1_score': 0.857540441815968, 'precision': 0.8192090395480226, 'recall': 0.89963
50364963503, 'auc_roc': 0.8971524993744243}, 'GradientBoosting': {'accuracy': 0.9905144107989785, 'confusion_matrix':
array([[2691,   51],
       [   1, 2739]], dtype=int64), 'f1_score': 0.9905967450271248, 'precision': 0.9817204301075269, 'recall': 0.9996
350364963503, 'auc_roc': 0.9997785195951594}, 'KNN': {'accuracy': 0.9850419554906968, 'confusion_matrix': array([[266
0,   82],
       [   0, 2740]], dtype=int64), 'f1_score': 0.9852571017619561, 'precision': 0.9709425939050319, 'recall': 1.0,
'auc_roc': 0.9919371283148856}, 'NaiveBayes': {'accuracy': 0.5368478657424298, 'confusion_matrix': array([[ 276, 246
6],
```

```
       [  73, 2667]], dtype=int64), 'f1_score': 0.6775053981963673, 'precision': 0.5195791934541204, 'recall': 0.9733
576642335766, 'auc_roc': 0.7360718240721514}}
Results for KMSMOTE_AE_3_PCA: {'ANN': {'accuracy': 0.9697301239970825, 'confusion_matrix': array([[2659,   83],
       [  83, 2659]], dtype=int64), 'f1_score': 0.9697301239970825, 'precision': 0.9697301239970825, 'recall': 0.9697
301239970825, 'auc_roc': 0.9926386208855841}, 'RandomForest': {'accuracy': 0.9927060539752006, 'confusion_matrix': ar
ray([[2706,   36],
       [   4, 2738]], dtype=int64), 'f1_score': 0.9927483683828862, 'precision': 0.9870223503965393, 'recall': 0.9985
412107950401, 'auc_roc': 0.9997474916752721}, 'XGBoost': {'accuracy': 0.9941648431801605, 'confusion_matrix': array
([[2711,   31],
       [   1, 2741]], dtype=int64), 'f1_score': 0.9941965904969169, 'precision': 0.9888167388167388, 'recall': 0.9996
3530269876, 'auc_roc': 0.9997288045961968}, 'SVM': {'accuracy': 0.937272064186725, 'confusion_matrix': array([[2536,
206],
       [ 138, 2604]], dtype=int64), 'f1_score': 0.9380403458213257, 'precision': 0.9266903914590747, 'recall': 0.9496
71772428884, 'auc_roc': 0.9770516018750388}, 'LogisticRegression': {'accuracy': 0.9086433260393874, 'confusion_matri
x': array([[2467,  275],
       [ 226, 2516]], dtype=int64), 'f1_score': 0.9094523766491958, 'precision': 0.9014690075241849, 'recall': 0.9175
784099197666, 'auc_roc': 0.9475611300242971}, 'GradientBoosting': {'accuracy': 0.9925237053245806, 'confusion_matri
x': array([[2704,   38],
       [   3, 2739]], dtype=int64), 'f1_score': 0.9925711179561515, 'precision': 0.9863161685271876, 'recall': 0.9989
059080962801, 'auc_roc': 0.999436195528827}, 'KNN': {'accuracy': 0.9777534646243617, 'confusion_matrix': array([[263
1,  111],
       [  11, 2731]], dtype=int64), 'f1_score': 0.9781518624641834, 'precision': 0.9609429978888107, 'recall': 0.9959
883296863603, 'auc_roc': 0.9893184389997876}, 'NaiveBayes': {'accuracy': 0.5559810357403355, 'confusion_matrix': arra
y([[ 383, 2359],
       [  76, 2666]], dtype=int64), 'f1_score': 0.6864941418823226, 'precision': 0.530547263681592, 'recall': 0.97228
30051057623, 'auc_roc': 0.8071921047689425}}
Results for KMSMOTE_MICE_3_PCA: {'ANN': {'accuracy': 0.9781181619256017, 'confusion_matrix': array([[2659,   83],
       [  37, 2705]], dtype=int64), 'f1_score': 0.9783001808318263, 'precision': 0.9702295552367288, 'recall': 0.9865
06199854121, 'auc_roc': 0.9932256345759642}, 'RandomForest': {'accuracy': 0.9930707512764405, 'confusion_matrix': arr
ay([[2708,   34],
       [   4, 2738]], dtype=int64), 'f1_score': 0.9931084512150888, 'precision': 0.9877344877344877, 'recall': 0.9985
412107950401, 'auc_roc': 0.9997593290420884}, 'XGBoost': {'accuracy': 0.9936177972283005, 'confusion_matrix': array
([[2709,   33],
       [   2, 2740]], dtype=int64), 'f1_score': 0.9936536718041704, 'precision': 0.9880995311936531, 'recall': 0.9992
706053975201, 'auc_roc': 0.9998380009799744}, 'SVM': {'accuracy': 0.937454412837345, 'confusion_matrix': array([[253
6,  206],
       [ 137, 2605]], dtype=int64), 'f1_score': 0.9382315865298038, 'precision': 0.9267164710067591, 'recall': 0.9500
36469730124, 'auc_roc': 0.976826026884921}, 'LogisticRegression': {'accuracy': 0.9028081692195478, 'confusion_matri
x': array([[2454,  288],
       [ 245, 2497]], dtype=int64), 'f1_score': 0.9035643206079247, 'precision': 0.8965888689407541, 'recall': 0.9106
491611962072, 'auc_roc': 0.9447957083294096}, 'GradientBoosting': {'accuracy': 0.9930707512764405, 'confusion_matri
x': array([[2707,   35],
       [   3, 2739]], dtype=int64), 'f1_score': 0.9931109499637418, 'precision': 0.9873828406633021, 'recall': 0.9989
059080962801, 'auc_roc': 0.9997156371881651}, 'KNN': {'accuracy': 0.9826768781911014, 'confusion_matrix': array([[265
2,   90],
       [   5, 2737]], dtype=int64), 'f1_score': 0.9829412820973245, 'precision': 0.9681641315882561, 'recall': 0.9981
765134938001, 'auc_roc': 0.9916931610876759}, 'NaiveBayes': {'accuracy': 0.5516046681254558, 'confusion_matrix': arra
y([[ 339, 2403],
       [  56, 2686]], dtype=int64), 'f1_score': 0.6859915719576044, 'precision': 0.5278050697583022, 'recall': 0.9795
769511305616, 'auc_roc': 0.8358999803685918}}
Results for SVMSMOTE_AE_3_PCA: {'ANN': {'accuracy': 0.9812180889861415, 'confusion_matrix': array([[2643,   99],
       [   4, 2738]], dtype=int64), 'f1_score': 0.9815379100197168, 'precision': 0.9651039830807191, 'recall': 0.9985
412107950401, 'auc_roc': 0.9946508402402374}, 'RandomForest': {'accuracy': 0.9947118891320205, 'confusion_matrix': ar
ray([[2715,   27],
       [   2, 2740]], dtype=int64), 'f1_score': 0.9947358867308042, 'precision': 0.9902421395012649, 'recall': 0.9992
706053975201, 'auc_roc': 0.9999214610662355}, 'XGBoost': {'accuracy': 0.9930707512764405, 'confusion_matrix': array
([[2707,   35],
       [   3, 2739]], dtype=int64), 'f1_score': 0.9931109499637418, 'precision': 0.9873828406633021, 'recall': 0.9989
059080962801, 'auc_roc': 0.9998229715142413}, 'SVM': {'accuracy': 0.9390955506929248, 'confusion_matrix': array([[253
4,  208],
       [ 126, 2616]], dtype=int64), 'f1_score': 0.9399928135106, 'precision': 0.9263456090651558, 'recall': 0.9540481
400437637, 'auc_roc': 0.9774536068323685}, 'LogisticRegression': {'accuracy': 0.9035375638220278, 'confusion_matrix':
array([[2498,  244],
       [ 285, 2457]], dtype=int64), 'f1_score': 0.9028109498438361, 'precision': 0.9096630877452795, 'recall': 0.8960
612691466083, 'auc_roc': 0.9536440070204896}, 'GradientBoosting': {'accuracy': 0.9928884026258206, 'confusion_matri
x': array([[2708,   34],
       [   5, 2737]], dtype=int64), 'f1_score': 0.9929258117177581, 'precision': 0.9877300613496932, 'recall': 0.9981
765134938001, 'auc_roc': 0.9996596424530002}, 'KNN': {'accuracy': 0.9897884755652808, 'confusion_matrix': array([[268
7,   55],
       [   1, 2741]], dtype=int64), 'f1_score': 0.9898880462260744, 'precision': 0.9803290414878397, 'recall': 0.9996
3530269876, 'auc_roc': 0.9939639404545867}, 'NaiveBayes': {'accuracy': 0.5351932895696572, 'confusion_matrix': array
([[ 313, 2429],
       [ 120, 2622]], dtype=int64), 'f1_score': 0.6729115873219556, 'precision': 0.5191051276974856, 'recall': 0.9562
363238512035, 'auc_roc': 0.7500727532544779}}
Results for SVMSMOTE_MICE_3_PCA: {'ANN': {'accuracy': 0.9828592268417213, 'confusion_matrix': array([[2653,   89],
```

```
       [   5, 2737]], dtype=int64), 'f1_score': 0.983117816091954, 'precision': 0.9685067232837934, 'recall': 0.99817
65134938001, 'auc_roc': 0.9958149588139437}, 'RandomForest': {'accuracy': 0.9925237053245806, 'confusion_matrix': arr
ay([[2707,   35],
       [   6, 2736]], dtype=int64), 'f1_score': 0.9925630328314892, 'precision': 0.9873691808011548, 'recall': 0.9978
118161925602, 'auc_roc': 0.9998945942336862}, 'XGBoost': {'accuracy': 0.9945295404814004, 'confusion_matrix': array
([[2713,   29],
       [   1, 2741]], dtype=int64), 'f1_score': 0.9945573294629898, 'precision': 0.9895306859205776, 'recall': 0.9996
3530269876, 'auc_roc': 0.9998450501984155}, 'SVM': {'accuracy': 0.9341721371261853, 'confusion_matrix': array([[2530,
212],
       [ 149, 2593]], dtype=int64), 'f1_score': 0.9349197764557419, 'precision': 0.9244206773618538, 'recall': 0.9456
601021152443, 'auc_roc': 0.9732490140404472}, 'LogisticRegression': {'accuracy': 0.8920495988329686, 'confusion_matri
x': array([[2485,  257],
       [ 335, 2407]], dtype=int64), 'f1_score': 0.8904920458749538, 'precision': 0.9035285285285285, 'recall': 0.8778
264040846098, 'auc_roc': 0.9438662755281461}, 'GradientBoosting': {'accuracy': 0.9916119620714807, 'confusion_matri
x': array([[2702,   40],
       [   6, 2736]], dtype=int64), 'f1_score': 0.9916636462486408, 'precision': 0.9855907780979827, 'recall': 0.9978
118161925602, 'auc_roc': 0.9998310847656547}, 'KNN': {'accuracy': 0.9896061269146609, 'confusion_matrix': array([[268
6,   56],
       [   1, 2741]], dtype=int64), 'f1_score': 0.989709333814768, 'precision': 0.9799785484447623, 'recall': 0.99963
530269876, 'auc_roc': 0.9943150048333698}, 'NaiveBayes': {'accuracy': 0.536834427425237, 'confusion_matrix': array([[
290, 2452],
       [  88, 2654]], dtype=int64), 'f1_score': 0.6763506625891947, 'precision': 0.5197806502154328, 'recall': 0.9679
066374908826, 'auc_roc': 0.7783898627450667}}
<Figure size 1000x600 with 0 Axes>
<Figure size 1000x600 with 0 Axes>
<Figure size 1000x600 with 0 Axes>
<Figure size 1000x600 with 0 Axes>
<Figure size 1000x600 with 0 Axes>
<Figure size 1000x600 with 0 Axes>
```

In [ ]:

In [ ]: