

ANALYTICS LAB(ABW508D)

Bankruptcy Risk Warning prediction for
Taiwanese companies.

Marian Saad Naguib Saad
P-EM0120/23

Supervisor: Prof. Dr. Noor Hazlina Ahmed.

Contents

.....	1
Table of figures	4
Acknowledgement.....	6
Abstract.....	7
Chapter one.....	8
1.1 Introduction:	8
1.2 Background:.....	8
1.3 problem of statement:	11
1.4 Significant of study:.....	11
1.5 research objectives:.....	12
1.6 research questions:.....	12
1.7 Scope:	12
1.8 Outline:	12
Chapter Two:	14
2.literature review:.....	14
2.1 Bankruptcy prediction:	14
2.2 Supervised machine learning:.....	16
2.3 Classification:	16
2.3.1 Random Forest:.....	16
2.3.2 Adaboost:	17
2.3.3 Support Vector Machine (SVM):.....	17
2.3.4 K-Nearest Neighbor (KNN):	17
2.3.5 Logistic Regression:	18
2.3.6 Naïve Bayes:.....	19
2.3.7 XGBoost:	19
2.3.8 Decision Tree:.....	20
Chapter three.....	21
2.1 Data Retrieval:	22
3.2 Data Cleaning:	30
3.2.1 Handling Missing Values:	30
3.2.2 checking for duplication:	31
3.2.3 Eliminating Unnecessary Features:	31
3.2.4 Outliers Capping:.....	31
3.3 Exploratory Analysis:	32
3.3.1 Statistical Summary:.....	32

3.3.2 Plotting the distribution:	33
3.3.3 Checking for multicollinearity.....	34
3.3.5 Correlation Analysis:	35
3.4 Data preprocessing:	36
3.4.1 Data splitting:	36
3.4.2: Resampling techniques:.....	36
3.4.3: Standardize features:.....	37
3.4.4 Principal component analysis (PCA):.....	38
3.5 Modeling:.....	38
3.6 Evaluating:.....	38
3.6.1: confusion metric:	38
3.6.2 Accuracy:	39
3.6.3 F1 score:	40
3.6.4 Area under the ROC Curve:.....	40
Chapter four:.....	42
4.1 Results for descriptive analysis:	42
4.2 Results for models:	46
4.2.1 Results without resample techniques:.....	46
4.2.2 Results after applying smote:.....	47
4.2.3 Results after applying SMOTE-ENN:.....	48
4.2.4 Champion model:	49
4.3 Discussion:.....	50
Chapter 5	51
5.1 Conclusion:	51
5.2 Limitation:	51
5.3 Recommendations:	52
References:	52
Appendix	54

Table of figures

Figure 1 historical bankruptcy in Taiwan from 2001 to 2008	10
Figure 2 historical bankruptcy in Taiwan from 2009 to 2016	10
Figure 3 historical bankruptcy in Taiwan from 2017 to 2023	11
Figure 4 3-Nearest Neighbor Classification in a 2D feature space.....	18
Figure 5 : the flow chart for bankruptcy prediction	21
Figure 6 Checking Outliers using boxplot	32
Figure 7 Statistical Summary of Numerical Variables	33
Figure 8: histogram for distribution	34
Figure 9 data imbalance	35
Figure 10 scatter plot for correlation	35
Figure 11: data set after SMOTE	37
Figure 12 confusion metrix	39
Figure 13 : ROC Curve Plot from Machine Learning Mastery website.....	41
Figure 14 : Bankruptcy VS Non bankruptcy	42
Figure 15 key influencers of bankruptcy.....	42
Figure 16: R&D for non-bankruptcy companies	43
Figure 17: R&D for bankruptcy companies.....	43
Figure 18 people efficiency and bankruptcy.....	44
Figure 19 : Risk analysis for bankruptcy companies	45
Figure 20 results with SMOTE	47
Figure 21 Results with SMOTEEN.....	48
Figure 22 dashboard using power bi	54

List of Acronyms:

ML: Machine Learning

AI: Artificial Intelligence

RF: Random Forest

LR: Logistic Regression Classifier

KNN: K-Nearest Neighbor Classifier

DTC: Decision Tree Classifier

SVC: Support Vector Classifier

SMOTE: Synthetic Minority Over-sampling Technique

SMOTE-ENN: Synthetic Minority Over-sampling Technique with Edited Nearest Neighbors.

RENN: repetition of ENN editing

PCA: Principal Component Analysis

TP: True Positive

FP: False Positive

TN: True Negative

FN: False Negative

ROC AUC: Area Under the ROC Curve

Acknowledgement

I extend my deepest gratitude to God for providing me with the strength and resilience throughout my academic journey. My heartfelt thanks go to my supervisor, Prof. Dr. Noor Hazline Ahmed, for her unwavering support and guidance. I am also thankful to University Sains Malaysia for fostering an enriching academic environment. The assistance from the DEBI Scholarship has been invaluable, and I am grateful for the support of Eng Ahmed Khaled. I also want to express my gratitude to my family and colleagues. Each of these entities has played a crucial role in the successful completion of my study of bankruptcy prediction.

Abstract

In the dynamic landscape of financial markets, predicting corporate bankruptcy has emerged as a critical area of research and practical application. Corporate bankruptcy happens when an organization being unable to repay its obligations. Accurate prediction of bankruptcy can help stakeholders take a better decision.

This study is applied using Taiwan bankruptcy dataset, which contains historical data from 1999 to 2009. Machine learning algorithms were applied to predict whether or not a company will face bankruptcy. Several models such as Logistic regression and SVM were applied. Additionally, ensemble learning methods are explored to leverage the strengths of multiple models, enhancing overall prediction accuracy. The performance of the models was measured using some metrics such as confusion metric and ROC (AUC).

In this study we used resampling techniques included SMOTE, and SMOTE-ENN. PCA is also used for dimensionality reduction. The champion model after hyper parameter tuning was Logistic Regression model with F1 score 0.97.

The results of this study can be used as a basic guideline for applying machine learning to predict the risk of bankruptcy. This will be significant for financial institutions, investors, and policymakers, offering them a proactive tool for risk mitigation and strategic decision-making.

Keywords: bankruptcy prediction, financial disasters, machine learning, PCA, resampling techniques, SMOTE, SMOTE-ENN, logistic regression, hyper parameter tuning.

Chapter one

This chapter included an introduction, background, aim, problem of statement, questions, and objectives

1.1 Introduction:

Nowadays companies spend on raw materials, salaries, investment, and other factors to gain a competitive advantage in the market. Unfortunately, high costs and low liquidity can cause corporates bankruptcy specially with start ups.as a result stakeholder as well as the overall macroeconomic affected badly. Machine learning can be useful in this area of study to minimize the risk of bankruptcy.(Muslim et al., 2023)

1.2Background:

This point represents an overview of bankruptcy, its history, and the differences between insolvency, and bankruptcy terms.it also discuss the effect of the global financial crisis and covid-19 on bankruptcy. Furthermore, it represents why Taiwan is a suitable country for this research.

Corporate bankruptcy can be associated with Inadequate management and the failure of meeting payment obligations. Financial disaster can happen due to cash shortages or negative net firm value. Excessive debt financing and risky investments can increase the likelihood of financial distress. If a company is unable to repay debts, it may be in default, which can lead to bankruptcy. It has become prevalent worldwide, and policymakers have implemented reforms to address the economic problems caused by increasing bankruptcies. Failure, insolvency, default, and bankruptcy are terms related to unsuccessful business enterprises, each with its own distinctions. Insolvency refers to the inability to repay debts, while bankruptcy is a legal adjudication that arises from this failure. Bankruptcy allows creditors to have control over the assets of the debtor. The process involves appointing a trustee to sell assets and distribute funds to creditors.(Yigit, 2018)

Bankruptcy has evolved from being considered a shameful last resort to being recognized as an important tool for resolving serious financial problems. The rise of debt in recent decades has led to the development of bankruptcy systems in many countries, including the UK and the USA, which have undergone major reforms due to increasing levels of bankruptcy. In the USA, the first modern Bankruptcy Act was introduced in 1898.

Different chapters of the US Bankruptcy Code address specific situations, such as liquidation or reorganization for businesses.(Yigit, 2018)

Bankruptcy has a sever effect on the economic and society. In 2008 after the financial crisis led to 389 US banks into bankruptcy, high instability in the European economy for example bankruptcy Portugal increased by 49%. In the past 90 years over 500 studies have been published using different methods of statistics and machine learning. Machine learning techniques perform better than statistical methods not only in bankruptcy prediction, but also in other fields such as energy and buildings(Brenes et al., 2022).

Covid-19 pandemic has a great effect on bankruptcy rate. During this pandemic consumer sending dropped, and the unemployment rate increased which caused bankruptcy for organizations specially in the struggling sectors such as retail. (J. Wang et al., 2020)

Statistical facts about bankruptcy in Taiwan:

Why Taiwan:

Taiwan is an excellent research destination for bankruptcy studies, given its global tech sector prominence, diverse corporate landscape from large corporations to SMEs, and crucial role in the international electronics supply chain. Additionally, Taiwan's dynamic technology-focused entrepreneurial environment provides valuable insights into the specific challenges confronted by innovative startups during financial distress

As published on the take-profit.org website. In August 2023 companies faced bankruptcy were about 2481.The maximum level was 7810 companies, and the minimum was 1144 companies. The following charts represent the volume of bankruptcy from 2001 to 2023. As we can see, 2008 and 2020 have a high rate due to covid and the global financial crisis.



Figure 1 historical bankruptcy in Taiwan from 2001 to 2008

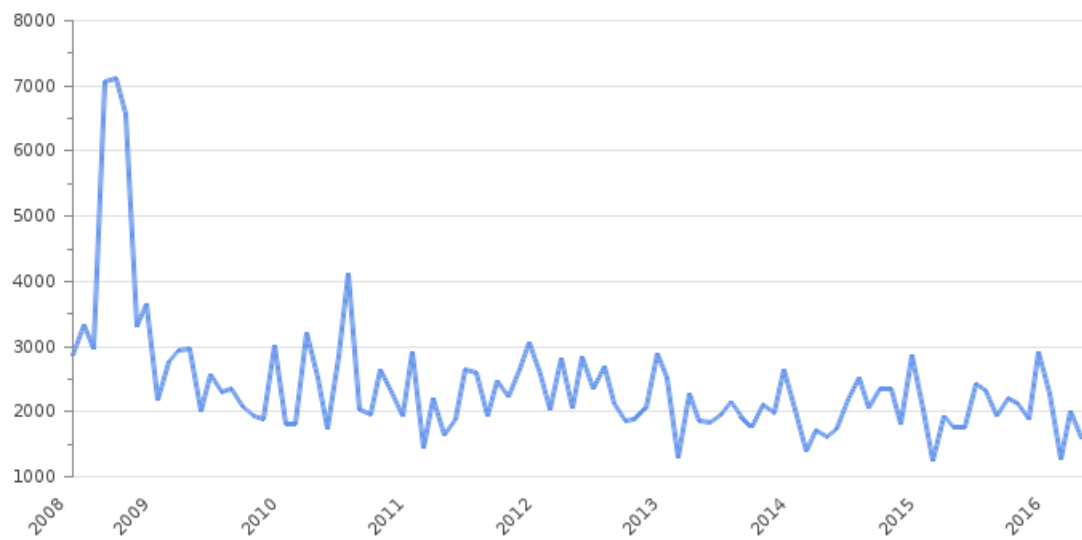


Figure 2 historical bankruptcy in Taiwan from 2009 to 2016



Figure 3 historical bankruptcy in Taiwan from 2017 to 2023

1.3 problem of statement:

In the finance field bankruptcy prediction is considered as an issue. Bankruptcy affects investors, customers, employees, and the overall economy. Also, companies that face the risk of bankruptcy represent burdens on the lending financial institutions. So the main goal is to predict whether the organization will face bankruptcy or not (Devi & Radhika, 2018). Statistical model were widely used to predict bankruptcy, but by the development of the economic and the technology machine learning began to be used (H. Wang & Liu, 2021). Machine learning has proven its efficiency in several disciplines, so machine learning techniques will be used to predict company bankruptcy.

1.4 Significant of study:

the ability to predict bankruptcy with greater accuracy using machine learning holds significant economic and social implications. By mitigating financial risks, promoting informed decision-making, and fostering a more stable financial environment, ML-based bankruptcy prediction has the potential to benefit a wide range of institutions and stakeholders as the following:

1. Financial institution: accurate prediction for bankruptcy can help banks, lenders, and insurance companies to minimize bad debts.
2. Companies: any organization can benefit from this study to monitor its financial position .as well as Companies can take proactive steps in case of early detection of bankruptcy such as cutting costs and changing their strategies.

3. Regulators and policymakers: predicting bankruptcy can identify systematic risk with the financial system which affect the overall macroeconomic.

1.5 research objectives:

- 1.to help government and the financial institutions to determine the financial position of corporates.
- 2.to identify key financial factors which affect bankruptcy in Taiwan.
- 3.to determine the champion model of machine learning
- 4.to evaluate the performance of machine learning models in this study
5. to evaluate the robustness and generalizability of bankruptcy prediction models under different economic conditions.

1.6 research questions:

- 1.What are the key financial factors that contribute to bankruptcy in Taiwan?
2. what is the champion model for this study?
3. how effective machine learning models at predicting bankruptcy?
4. How can the generalizability of machine learning models be monitored to reduce the gap between prediction and reality?
- 5.what are the limitations of this study?

1.7 Scope:

This study focusses on predicting the risk of corporate bankruptcy using machine learning classification algorithms. using a publicly available dataset and selecting the champion model based on performance metrics. This study does not contain the deployment stage.

1.8 Outline:

Chapter one: this chapter explains an overview of this topic and contains introduction, background, problem of statement, aim, scope, objective, and research questions.

Chapter two: this chapter discusses the literature review of this study.

Chapter three: this chapter describes the methodology used with this study which are data retrieval, data cleaning, data preprocessing, model building, and model evaluation.

Chapter four: this chapter presents the results, visualization, and discussion.

Chapter five: this chapter contains conclusion, limitations, and recommendations.

Chapter Two:

This chapter cover the literature review and explanation of machine learning models

2.literature review:

This chapter presents the literature review for this study, focusing on bankruptcy prediction, furthermore it presents machine learning models which were employed with this study.

2.1 Bankruptcy prediction:

Researchers from all around the world have become more and more interested in the topic of bankruptcy prediction during the last fifty years. Finding the most accurate model for predicting company failure has been the focus of numerous scholarly investigations. Since Altman's groundbreaking 1968 introduction of the bankruptcy prediction model, a significant predicting business financial distress has been the subject of a growing amount of study. Authors typically draw a line between failed and non-failed enterprises based on the ultimate failure, which is bankruptcy.

The four phrases failure, insolvency, default, and bankruptcy are genetic concepts that can be used to characterize disastrous business ventures.

- Failure: Realized rate of return on invested capital lower than prevailing rates on equivalent investments, taking risk into consideration.
- insolvency: The state in which a company cannot satisfy its ongoing debts because its liabilities exceed its assets.
- Default: Happens when a business doesn't meet a commitment, particularly one to make loan payments or show up in small claims court.
- Bankruptcy: There are three different forms of bankruptcy: technical bankruptcy, which describes a situation where a company is unable to repay principal and interest on schedule; legal bankruptcy, which literally means that the company files for a declaration of bankruptcy in court; and accounting-bankruptcy, which refers to a situation where a company is just showing negative book net assets(Shi & Li, 2019).

Related work:

Feature selection and dimensionality reduction are significant for bankruptcy prediction models. There are three types of feature selection: filter, wrapper, and embedded methods. Almost of the previous studies focused on applying filter methods (Lin et al., 2019). PCA for dimensionality reduction is widely used in the area of bankruptcy (Lin et al., 2019) (Tsai et al., 2021) (Bărbuță-Mișu & Madaleno, 2020). In general, there are fewer bankrupt banks or companies than there are non-bankrupt banks or businesses. The selection of a suitable metric may be influenced by the imbalance in data. Many studies used AUC(ROC) as a metric for classification problems. Also, they deal with the imbalance data using oversampling and under sampling techniques. We found that 9% applied oversampling, and 10% applied under sampling techniques according to papers published from 2012 to mid-2023 (Dasilas & Rigani, n.d.).

Previous studies applied statistical and machine learning techniques for this topic. We found that logistic regression is the highest statistical model used. The most machine learning and AI models applied were Neural Network, SVM, and Decision Tree. In addition, Adaboost and Random Forest were the most applied ensemble methods (Shi & Li, 2019).

This study compares the performance of three deep learning with six ensemble classifiers in predicting companies' financial failure. The study uses three extremely imbalanced datasets of Spanish, Taiwanese, and Polish companies' data and applies various data balancing techniques to avoid data inconsistency problems. The results show that deep learning methods outperform the ensemble classifiers in predicting financial failure. The study applied SMOTE, Borderline SMOTE, SMOTE-NC, SVM-SMOTE, ADASYN, SMOTE-ENN, SMOTE-Tomek, and K-means SMOTE. With Taiwan data set SMOTE-ENN show its proficiency as a resampling technique (Aljawazneh et al., 2021).

In their comprehensive study, Tsai, Sue, Hu, and Chiu investigated the combined impact of feature selection, instance selection, and ensemble classification techniques on bankruptcy prediction and credit scoring models. The researchers explored 288 combinations of these techniques and evaluated their performance across various datasets, including Taiwan. The results revealed that the choice of data preprocessing approach significantly influenced the predictive performance for the Taiwan dataset, with AP-PCA providing the highest AUC rate ($p < 0.05$). (Tsai et al., 2021)

This study applied LR, NB, SVM, LDA, XG-boost, KNN, RBF, and NN. Many resampling techniques are applied includes: Tomek Links, Edited Nearest Neighbors, RENN, and One-sided Selection. The results shows that naïve base with ENN is the champion model (H. Wang & Liu, 2021).

2.2 Supervised machine learning:

Guiding computers to solve issues with data is the core objective of machine learning. In supervised machine learning, inputs and outputs are mapped by a domain expert who is supervising the system's learning. For applications such as regression and classification, SML attempts to build a model of class labels based on predictor types. It is utilized in a variety of fields, including bioinformatics, speech recognition, and Finance, and requires analyzing data characteristics before selecting an algorithm. The primary benefit of SML is in its capacity to execute tasks autonomously after being taught on data (Shetty et al., 2022).

2.3 Classification:

Classification is a supervised machine learning technique. Classification is a data mining (machine learning) approach that is used to forecast group membership for data instances. Although there are variety of available techniques for machine learning but classification is most widely used technique(Soofi & Awan, 2017).

2.3.1Random Forest:

Random Forest is a popular ensemble classifier that combines multiple randomized decision trees to make predictions with high accuracy and performance. The method of bagging is used to select feature subsets, and the importance of each feature is ranked based on its contribution to the final decision. By aggregating the results of individual trees through majority voting, Random Forest achieves high accuracy and performance while reducing overfitting. Experimental results demonstrate that Random Forest achieves high classification accuracy and outperforms other classifiers such as decision trees and support vector machines in certain tasks(Parmar et al., 2019).

2.3.2 Adaboost:

AdaBoost is an ensemble learning algorithm that combines multiple weak classifiers to create a strong classifier. The algorithm works by iteratively re-weighting the training data, with more weight given to misclassified examples in each iteration. The final classifier is a weighted sum of the weak classifiers, with the weights determined by the accuracy of each classifier on the training data. AdaBoost can fit the training data with no error and still achieve good generalization performance. This is in contrast to the traditional view of AdaBoost as a margin-maximizing algorithm, which suggests that AdaBoost works by maximizing the margins between the decision boundary and the training data points(Wyner et al., 2017).

2.3.3 Support Vector Machine (SVM):

Support Vector Machines (SVMs) have emerged as powerful and robust classification and regression algorithms with applications across various scientific and engineering domains. When applied to imbalanced data sets, the performance of SVMs can be severely affected, especially when the ratio between the majority and minority class is large. The performance of SVM is particularly affected when dealing with imbalanced data sets. Various techniques, such as under-sampling, over-sampling, and evolutionary algorithms, have been used to address the negative effects of imbalanced data sets on SVM classifiers. The SVM algorithm can be used to classify both linear and nonlinear data. A linear SVM is used when the data can be separated by a linear hyperplane in the feature space. The goal of a linear SVM is to find the hyperplane that best separates the two classes in the feature space. A nonlinear SVM is used when the data cannot be separated by a linear hyperplane in the feature space. In this case, the data is transformed into a higher-dimensional feature space where a linear hyperplane can be used to separate the classes. The transformation is done using a kernel function, which maps the data into the higher-dimensional feature space.(Cervantes et al., 2020)

2.3.4 K-Nearest Neighbor (KNN):

Nearest Neighbors Classification determines the class of examples based on the class of their nearest neighbors. It is also referred to as Memory-based Classification because training examples are needed at runtime, and as Lazy Learning because induction is delayed to runtime. The concept is illustrated in Figure 1, which shows a 3-Nearest Neighbor Classifier in a two-dimensional feature space for a two-class problem. For example, q_1 is classified as O because all three of its nearest neighbors are of class O. However, q_2 , with two neighbors of class X and one of class O, requires a decision to be made using simple majority voting or distance weighted voting. (Cunningham & Delany, 2021)

P. Cunningham and S. J. Delany

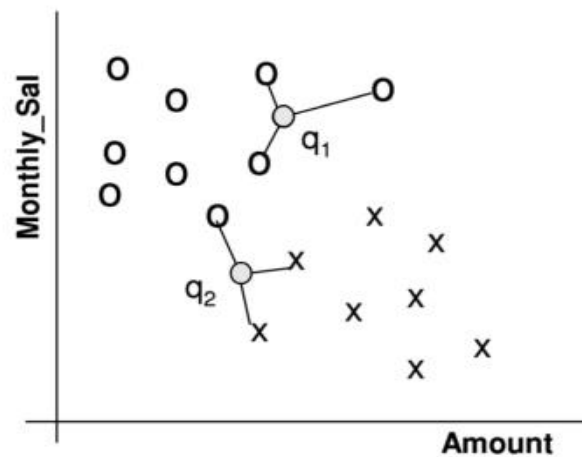


Figure 4 3-Nearest Neighbor Classification in a 2D feature space

2.3.5 Logistic Regression:

Logistic regression is a statistical analysis method used to explore and describe the relationship between a binary or dichotomous outcome and a set of independent predictors. The logistic regression model uses a logit link function to model the probability of a binary event. The model uses the following equation:

$$\log(p/(1-p)) = b_0 + b_1 \cdot x_1 + b_2 \cdot x_2 + \dots,$$

Where b_0 is the average odds for event Y to happen, x_1 , x_2 , etc. are independent predictors, and b_1 , b_2 , etc. are the corresponding regression coefficients. This model can be extended to incorporate ordinal and categorical outcomes with more than two groups. Occurring, and the regression coefficients can be interpreted as odds ratios. (Das, 2021).

2.3.6 Naïve Bayes:

Naïve Bayes is a form of Bayesian Network Classifier based on Bayes' rule. It assumes that the attributes are conditionally independent given the class, which means that the presence or absence of one attribute does not affect the presence or absence of another attribute. This assumption simplifies the calculation of probabilities and makes the algorithm computationally efficient. Naïve Bayes can be used for both categorical and numeric attributes. For categorical attributes, the required probabilities are normally derived from frequency counts stored in arrays whose values are calculated by a single pass through the training data at training time. For numeric attributes, either the data are discretized or probability density estimation is employed.

Naïve Bayes is widely applied in practice due to its desirable features, including computational efficiency, low variance, incremental learning, direct prediction of posterior probabilities, robustness in the face of noise and missing values, and the ability to use probabilities, making it relatively insensitive to noise in the training data.(Webb et al., 2010)

2.3.7XGBoost:

XGBoost is a highly effective and widely used machine learning method that offers a scalable end-to-end tree boosting system. Developed by Tianqi Chen and Carlos Guestrin, XGBoost has gained widespread recognition for its ability to achieve state-of-the-art results on various machine learning challenges. The system introduces a novel sparsity-aware algorithm designed to handle sparse data, along with a weighted quantile sketch procedure that enables approximate tree learning with instance weights. Its impact has been demonstrated in numerous machine learning and data mining challenges, where it has consistently outperformed other methods and has been the preferred choice for many winning solutions. Additionally, XGBoost is known for its scalability, running more than ten times faster than existing popular solutions on a single machine and being capable of scaling to billions of examples in distributed or memory-limited settings. This makes XGBoost a valuable tool for a wide range of applications, including spam classification, advertising, fraud detection, anomaly event detection, and more.(Chen & Guestrin, 2016).

2.3.8 Decision Tree:

Decision tree methodology is a widely used data mining method for establishing classification systems based on multiple covariates or for developing prediction algorithms for a target variable. Decision trees classify a population into branch-like segments that construct an inverted tree with a root node, internal nodes, and leaf nodes. The algorithm is non-parametric and can efficiently deal with large, complicated datasets without imposing a complicated parametric structure. The process of building a decision tree model involves splitting, stopping, and pruning, with the aim of simplifying complex relationships between input variables and target variables by dividing original input variables into significant subgroups (Song & Ying, 2015).

Several statistical algorithms for building decision trees are available, including CART (Classification and Regression Trees), C4.5, CHAID (Chi-Squared Automatic Interaction Detection), and QUEST (Quick, Unbiased, Efficient, Statistical Tree). These algorithms can be implemented using data mining software included in widely available statistical software packages such as SAS Enterprise Miner and IBM SPSS Modeler. Overall, decision tree methodology offers a powerful statistical tool for classification, prediction, interpretation, and data manipulation in various fields (Song & Ying, 2015).

Chapter three

This chapter represents methods conducted with this study from data retrieval to the evaluation of models.

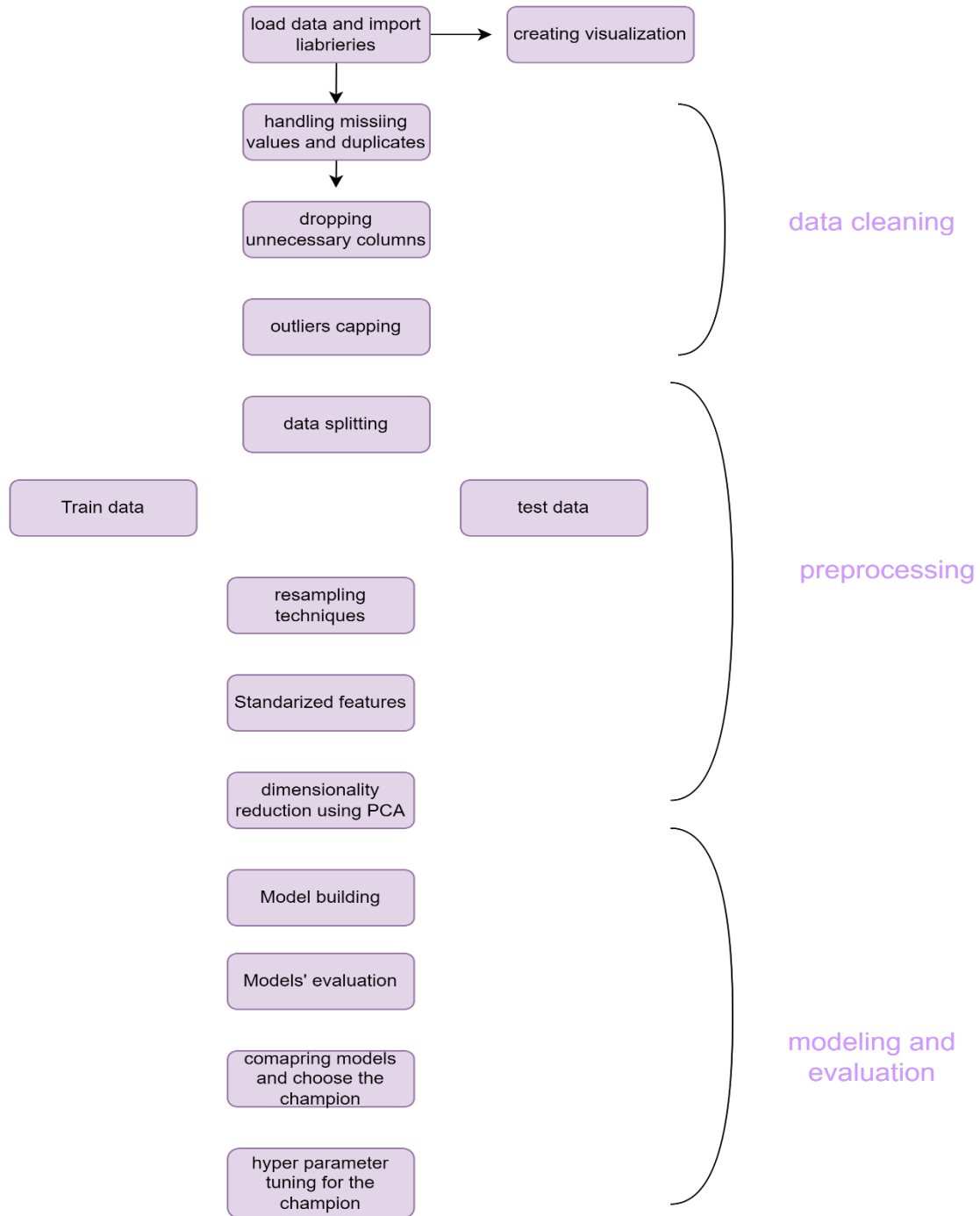


Figure 5 : the flow chart for bankruptcy prediction

2.1 Data Retrieval:

The dataset obtained from Kaggle and Kaggle got it from UCI. It is historical data for Taiwanese corporate bankruptcy. The data were collected from the Taiwan Economic Journal for the years 1999 to 2009. Company bankruptcy was defined based on the business regulations of the Taiwan Stock Exchange. it contains 6819 rows and 96 columns. The first step in the coding process is to install and import all the important libraries in google Colab. After that the data was imported by `read_csv` function(). The description of features will be in the following table:

feature	Description
'Bankrupt?'	whether the corporate 'Bankrupt or not (Yes, No)
' ROA(C) before interest and depreciation before interest'	company's profitability before accounting for interest and depreciation expenses
' ROA(A) before interest and % after tax',	company's profitability before accounting for interest expenses and after accounting for tax expenses
' ROA(B) before interest and depreciation after tax',	company's profitability before accounting for interest and depreciation expenses and after accounting for tax expenses
' Operating Gross Margin':	the profitability of a company's core operations.
' Realized Sales Gross Margin',	the profitability of a company's sales after accounting for the cost of goods sold
' Operating Profit Rate':	the profitability of a company's core operations, calculated as the ratio of operating income (earnings before interest and taxes) to net sales
' Pre-tax net Interest Rate',	the profitability of a company's investments and debt financing activities before accounting for taxes
' After-tax net Interest Rate',	the profitability of a company's investments and debt financing activities after accounting for taxes

' Non-industry income and expenditure/revenue',	the profitability of a company's non-industry-specific income and expenditure
' Continuous interest rate (after tax)',	the profitability of a company's investments and debt financing activities after accounting for taxes, on a continuous basis. It is calculated by dividing the company's net interest income
' Operating Expense Rate',	the efficiency of a company's operating expenses as a percentage of its net sales
' Research and development expense rate',	the efficiency of a company's R&D expenses as a percentage of its net sales.
' Cash flow rate',	the efficiency of a company's cash flow as a percentage of its net sales
' Interest-bearing debt interest rate'	the interest rate on a company's interest-bearing debt (debt that accrues interest, such as loans and bonds).
' Tax rate (A)',	the percentage of a company's profits that is paid in taxes
' Net Value Per Share (B)',	the net value of a company's assets and liabilities per share of common stock
' Net Value Per Share (A)',	the net value of a company's assets and liabilities per share of common stock, after accounting for preferred stock
' Net Value Per Share (C)',	the net value of a company's assets and liabilities per share of common stock, after accounting for preferred stock and minority interests
' Persistent EPS in the Last Four Seasons',	a company's profitability. Persistent EPS refers to a company's earnings per share that has remained stable or consistent over a period of time
' Cash Flow Per Share',	the amount of cash flow generated per share of common stock
' Revenue Per Share (Yuan ¥)',	the amount of revenue generated by a company per share of its stock expressed in Chinese yuan (¥)
' Operating Profit Per Share (Yuan ¥)',	a company's profitability that takes into account the revenue and expenses that are directly related to the company's operations. expressed in Chinese yuan (¥)

' Per Share Net profit before tax (Yuan ¥)',	a company's net income before taxes have been deducted expressed in Chinese yuan (¥)
' Realized Sales Gross Profit Growth Rate',	the increase or decrease in a company's gross profit over a period of time.
' Operating Profit Growth Rate',	the increase or decrease in a company's operating profit over a period of time
' After-tax Net Profit Growth Rate'	the increase or decrease in a company's net profit after taxes over a period of time
' Regular Net Profit Growth Rate',	the increase or decrease in a company's net profit over a period of time The regular net profit growth rate is calculated by comparing the net profit in a current period to the net profit in a previous period and expressing the difference as a percentage. It is a measure of how well a company is increasing its profitability over time.
' Continuous Net Profit Growth Rate'	the increase or decrease in a company's net profit over a period of time "continuous" in this context may refer to the fact that the growth rate is being calculated over a series of consecutive time periods
' Total Asset Growth Rate',	the increase or decrease in a company's total assets over a period of time. Total assets represent all of the resources that a company owns or controls that have monetary value, including cash, investments, property, and equipment.
' Net Value Growth Rate',	the increase or decrease in a company's net worth over a period of time. Net worth, also known as shareholder equity or net assets, represents the difference between a company's total assets and its total liabilities. It represents the portion of a company's assets that is owned by its shareholders.
' Total Asset Return Growth Rate Ratio',	the increase or decrease in the return on a company's total assets over a period of time.
' Cash Reinvestment %',	the percentage of a company's cash flow that is reinvested back into the business
' Current Ratio'	indicates a company's ability to pay its short-term debts and financial obligations

' Quick Ratio'	that indicates a company's ability to pay its short-term debts and financial obligations using its most liquid assets.
' Interest Expense Ratio',	indicates the percentage of a company's income that is used to pay interest on its debts
' Total debt/Total net worth',	indicates the proportion of a company's debt to its net worth. Net worth, also known as shareholder equity or net assets, represents the difference between a company's total assets and its total liabilities. It represents the portion of a company's assets that is owned by its shareholders.
' Debt ratio %'	indicates the percentage of a company's assets that are financed through debt
' Net worth/Assets'	indicates the proportion of a company's assets that are financed through net worth. Net worth, also known as shareholder equity or net assets, represents the difference between a company's total assets and its total liabilities.
' Long-term fund suitability ratio (A)',	that indicates the proportion of a company's long-term funds that are being used for long-term investments
' Borrowing dependency',	measure of how reliant a company is on borrowing to finance its operations and growth. It is typically expressed as a percentage and calculated by dividing a company's total debt by its total capital, which includes both debt and equity
' Contingent liabilities/Net worth',	indicates the proportion of a company's net worth that is represented by contingent liabilities
' Operating profit/Paid-in capital',	indicate that a company is generating a significant amount of profit from its core business operations relative to the amount of capital that has been invested by its shareholders
' Net profit before tax/Paid-in capital',	indicates the proportion of a company's paid-in capital that is represented by its net profit before tax. It is calculated by dividing a company's net profit before tax by its paid-in capital.
' Inventory and accounts	that indicates the proportion of a company's net worth that is represented by its inventory and accounts receivable. It is calculated by dividing a company's inventory and accounts receivable by its net worth.
' Total Asset Turnover',	indicates the efficiency with which a company is using its assets to generate revenue

' Accounts Receivable Turnover',	indicates the efficiency with which a company is collecting payments from its customers.
' Average Collection Days',	indicates the average number of days that it takes a company to collect payments from its customers
' Inventory Turnover Rate (times)',	indicates the efficiency with which a company is managing its inventory
' Fixed Assets Turnover Frequency',	the efficiency with which a company is using its fixed assets to generate revenue. Fixed assets are long-term physical assets that are used in a company's operations, such as buildings, machinery, and equipment.
' Net Worth Turnover Rate (times)',	indicates the efficiency with which a company is using its net worth to generate revenue
' Revenue per person'	that indicates the amount of revenue that a company generates per employee
' Operating profit per person'	that indicates the amount of operating profit that a company generates per employee. Operating profit is a company's profit from its core business operations, before accounting for other expenses such as interest and taxes. It is calculated by subtracting a company's operating expenses from its revenue.
' Allocation rate per person',	a percentage of an investor's cash or capital outlay that goes toward a final investment
' Working Capital to Total Assets',	that represents a company's short-term liquidity
' Quick Assets/Total Assets'	Company can be easily converted into cash. They include cash, cash equivalents (short-term investments that can be quickly sold), and accounts receivable (amounts owed to the company by customers for goods or services that have been delivered or performed but not yet paid for).
' Current Assets/Total Assets'	indicates the proportion of a company's assets that are represented by its current assets. It is calculated by dividing a company's current assets by its total assets.
' Cash/Total Assets'	indicates the proportion of a company's assets that are represented by its cash. It is calculated by dividing a company's cash by its total assets.

' Quick Assets/Current Liability',	indicates the extent to which a company's quick assets are able to cover its current liabilities. It is calculated by dividing a company's quick assets by its current liabilities.
' Cash/Current Liability'	indicates the extent to which a company's cash is able to cover its current liabilities. It is calculated by dividing a company's cash by its current liabilities.
' Current Liability to Assets'	indicates the proportion of a company's assets that are financed by its current liabilities. It is calculated by dividing a company's current liabilities by its total assets.
' Operating Funds to Liability'	indicates the extent to which a company's operating funds are able to cover its liabilities. It is calculated by dividing a company's operating funds by its liabilities.
' Inventory/Working Capital',	indicates the proportion of a company's working capital that is represented by its inventory. It is calculated by dividing a company's inventory by its working capital.
' Inventory/Current Liability'	company's ongoing ability to pay its debts as they are due. Accounts payable is typically one of the largest current liability accounts on a company's financial statements, and it represents unpaid supplier invoices
' Current Liabilities/Liability'	The ratio of current assets to current liabilities is important in determining a company's ongoing ability to pay its debts as they are due.
' Working Capital/Equity'	
' Current Liabilities/Equity'	The ratio of long-term liabilities to current assets is a measure of a company's financial health. A higher ratio suggests that the company has more long-term liabilities than current assets, which may indicate that the company may not be able to cover its current liabilities with its current assets.
' Long-term Liability to Current Assets'	The ratio of long-term liabilities to current assets is a measure of a company's financial health.

' Retained Earnings to Total Assets'	The ratio of retained earnings to total assets is used to measure how much of the total assets of a company are funded by retained earnings or profits.
' Total income/Total expense'	a company's profitability. It is calculated by dividing a company's total income by its total expenses
' Total expense/Assets'	The total expense to assets ratio is a measure of a company's efficiency in using its assets to generate income. It is calculated by dividing a company's total expenses by its total assets
' Current Asset Turnover Rate'	firm's ability of generating sales through its current assets (cash, inventory, accounts receivable, etc.)
' Quick Asset Turnover Rate'	the firm's ability to generate sales through its quick assets (cash and marketable securities)
' Working capital Turnover Rate'	a measure of how well a company is utilizing its working capital to support a given level of sales. Working capital is current assets minus current liabilities
' Cash Turnover Rate'	measure of how quickly a company's cash is being used to generate revenues
' Cash Flow to Sales'	measure of how much cash is available to support a given level of sales.
' Fixed Assets to Assets'	measure of how much of a company's total assets are tied up in fixed assets
' Current Liability to Liability'	a measure of how much of a company's total liabilities are current liabilities
' Current Liability to Equity'	a measure of how much of a company's total liabilities are current liabilities in comparison to its total equity
' Equity to Long-term Liability'	The equity to long-term liability ratio is a measure of how much of a company's total liabilities are long-term liabilities in comparison to its total equity
' Cash Flow to Total Assets'	an efficiency ratio that rates actual cash flows to the company assets without being affected by income recognition or income measurements.
' Cash Flow to Liability'	ratio is a measure of the ability of a company to pay off its liabilities with its operating cash flow. It is calculated by dividing operating cash flow by total liabilities

' CFO to Assets'	a measure of how much of a company's total assets are managed by the CFO in comparison to the total assets. It is calculated by dividing the company's total assets managed by the CFO by its total assets
' Cash Flow to Equity'	the amount of cash a business generates that is available to be potentially distributed to shareholders
' Current Liability to Current Assets',	a measure of a company's liquidity. The ratio is calculated by dividing the total current liabilities by the total current assets.
' Liability-Assets Flag'	financial ratio that shows a company's overall financial health. It is calculated by dividing total liabilities by total assets. If the value of the flag is 1 or more, the company has more liabilities than assets and is in a financially unhealthy position. If the value is less than 1, the company has more assets than liabilities and is in a financially healthy position. This ratio can be used to compare a company to its peers and help investors evaluate its management of liabilities.
' Net Income to Total Assets'	shows the profitability of a company in relation to the total value of its assets. It is calculated by dividing the company's net income by its total assets.
' Total assets to GNP price'	a financial ratio that compares the value of a company's total assets to the price of the country's gross national product
' No-credit Interval'	is a financial metric that indicates the number of days that a company can operate without needing to access noncurrent assets, long-term assets whose full value cannot be obtained within the current accounting year, or additional outside financial resources.
' Gross Profit to Sales'	financial ratio that measures the profitability of a company's sales. It is calculated by dividing the company's gross profit by its total sales.
' Net Income to Stockholder's Equity'	the profitability of a company in relation to the equity of its shareholders. It is calculated by dividing a company's net income by its stockholder's equity.
' Liability to Equity'	financial ratio that compares a company's total liabilities to its shareholder equity
' Degree of Financial Leverage (DFL)',	financial ratio that measures the extent to which a company relies on borrowed funds to finance its operations and assets. It is calculated by dividing a company's total liabilities by its shareholder equity

' Interest Coverage Ratio (Interest expense to EBIT)',	s a financial measure that shows the ability of a company to pay the interest on its outstanding debt. It is calculated by dividing the company's earnings before interest and taxes (EBIT) by its interest expense
' Net Income Flag'	represents an individual's total earnings or pre-tax earnings after factoring deductions and taxes in gross income.
' Equity to Liability'	a financial ratio that compares a company's shareholder equity to its total liabilities. It is calculated by dividing a company's shareholder equity by its total liabilities.

3.2Data Cleaning:

Data preprocessing is essential before applying machine learning models as it ensures the quality of data, increases the efficiency of models, and improves models' generalization.

3.2.1 Handling Missing Values:

To check for missing values in this data the following functions are used:

- 1) data.isna().sum().sum()
- 2) np.isnan(data).sum().sum()
- 3) data.isnull().sum().sum()

Those functions are used to check for missing and null values. We found that there are no missing values in this dataset. The following figure show that:



```

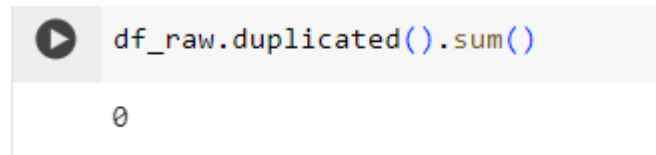
print(df_raw.isna().sum().sum())
print(np.isnan(df_raw).sum().sum())
print(df_raw.isnull().sum().sum())

```

0
0
0

3.2.2 checking for duplication:

The function `data_raw.duplicated().sum()` is used to check for duplication in the whole dataset. The result shows that there are no duplicates in the data. The following figure show that:



```
df_raw.duplicated().sum()
0
```

3.2.3 Eliminating Unnecessary Features:

Using `value_counts()` function we found that “Net Income Flag” feature contain only one unique value , class 1 which refers to that all companies in this dataset achieve negative net income in the last two years. Also, we found that “Liability-Assets Flag” feature is extremely imbalance. So we decide to drop both of them using the following function: `df.drop(columns=[' Net Income Flag', ' Liability-Assets Flag'],inplace=True)`

3.2.4 Outliers Capping:

We used boxplot and IQR to determine whether the data contains outliers or not. The results show that 88 columns contain outliers. We chose to cap the outliers instead of removing them to keep most of information specially with our small dataset.the following figure contain the boxplot for all features.

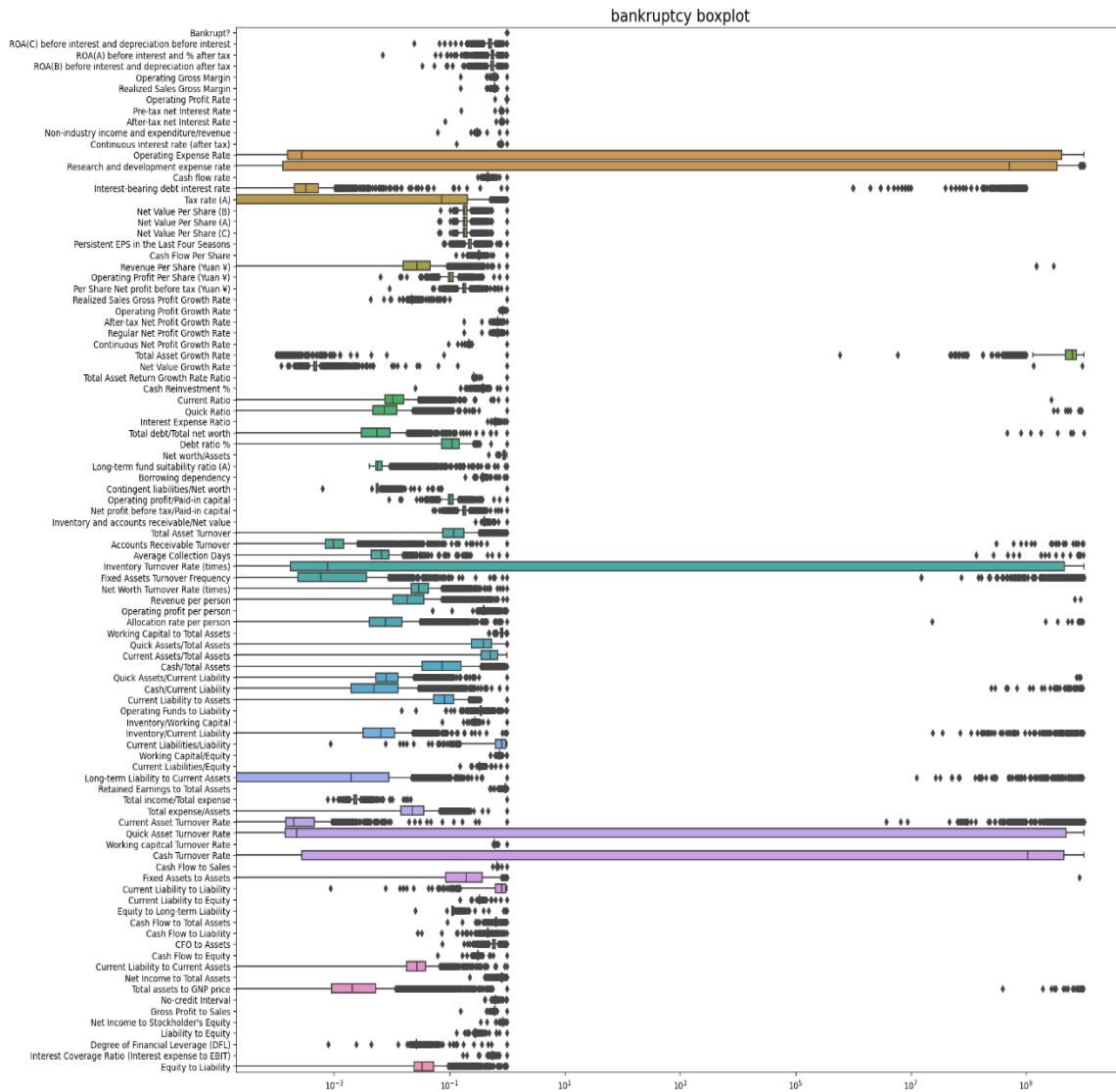


Figure 6 Checking Outliers using boxplot

3.3 Exploratory Analysis:

3.3.1 Statistical Summary:

It is critical to understand summary statistics provide a quick snapshot of patterns, trends, and potential outliers. Since all features are numerical the function `df.describe()` is used.


```
df.describe()
```

	Bankrupt?	ROA(C) before interest and depreciation before interest	ROA(A) before interest and % after tax	ROA(B) before interest and depreciation after tax	Operating Gross Margin	Realized Sales Gross Margin	Operating Profit Rate	Pre-tax net Interest Rate
count	6819.000000	6819.000000	6819.000000	6819.000000	6819.000000	6819.000000	6819.000000	6819.000000
mean	0.032263	0.505180	0.558625	0.553589	0.607948	0.607929	0.998755	0.797190
std	0.176710	0.060686	0.065620	0.061595	0.016934	0.016916	0.013010	0.012869
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.476527	0.535543	0.527277	0.600445	0.600434	0.998969	0.797386
50%	0.000000	0.502706	0.559802	0.552278	0.605997	0.605976	0.999022	0.797464
75%	0.000000	0.535563	0.589157	0.584105	0.613914	0.613842	0.999095	0.797579
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

8 rows x 96 columns

4

Figure 7 Statistical Summary of Numerical Variables

3.3.2 Plotting the distribution:

Plotting distribution for each feature is essential to see whether the data is normally distributed or skewed to the left or right. Using the histogram, we plot the distribution and found some skewed features as the next graph show:

```
df.hist(bins = 50, figsize = (40, 20));
```

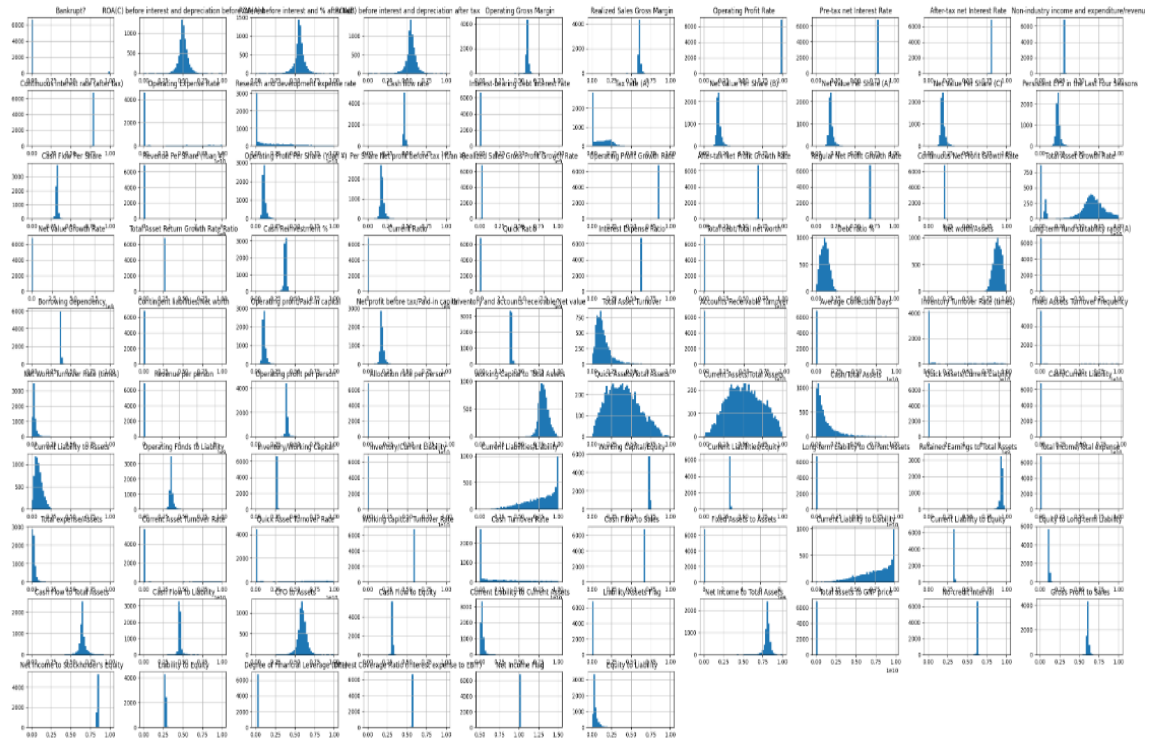


Figure 8: histogram for distribution

3.3.3 Checking for multicollinearity

we address the challenge of multicollinearity, a phenomenon where high correlations among independent variables hinder models in prediction. To detect and mitigate multicollinearity, we employ the variance inflation factor (VIF) method, a widely accepted technique for assessing the strength of correlations between independent variables. The VIF is calculated by regressing each variable against all others, and a higher VIF indicates increased correlation. A VIF value of 1 implies no correlation, while values exceeding 10 signify notable multicollinearity (Sundus et al., 2022). When we apply VIF to our data, we found 53 features get score higher than 10 which prove that the data face multicollinearity.

3.3.4 Checking for data balance:

When we check for the balance of the data, we found it highly imbalance. Class 1 is only 3.23%. Resampling should be consider with the data.

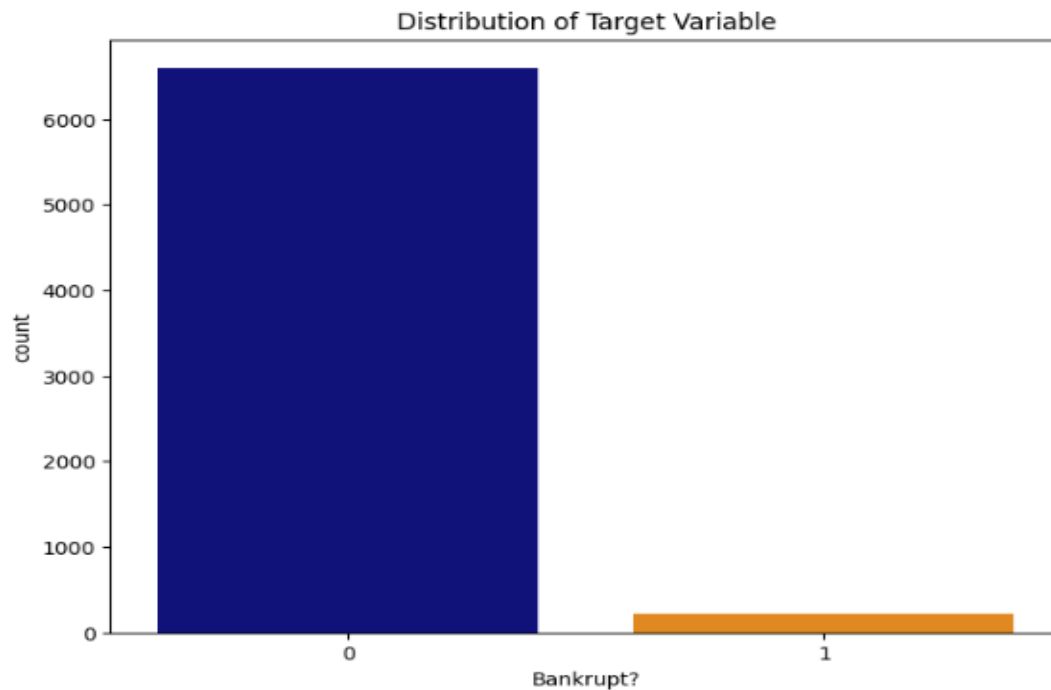


Figure 9 data imbalance

3.3.5 Correlation Analysis:

Using the scatter plot, we see the relation between each feature and the target column in the data. The scatter is plotted using `df.corr()['Bankrupt?']` function.

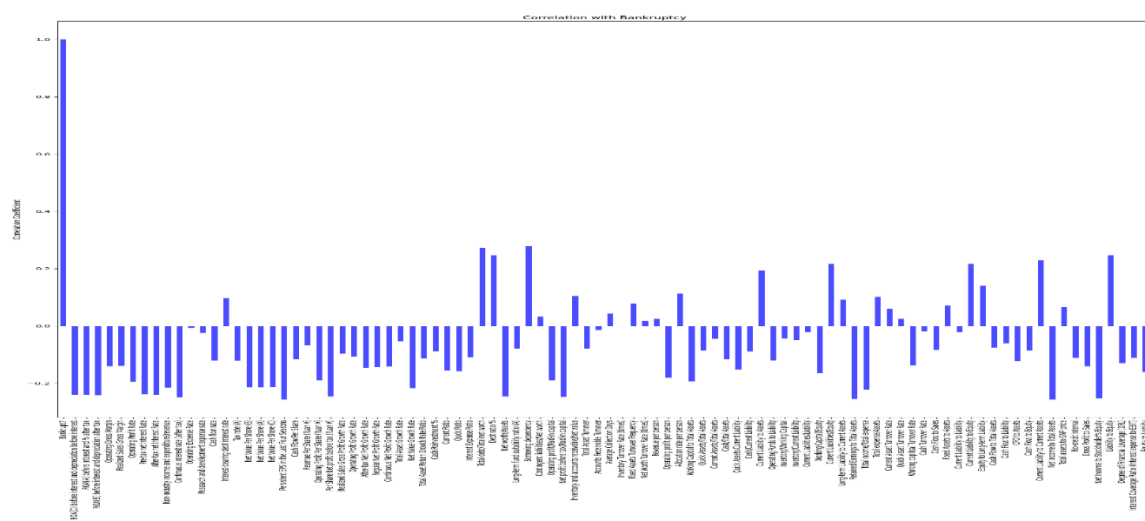


Figure 10 scatter plot for correlation

3.4 Data preprocessing:

3.4.1 Data splitting:

Effective data splitting is crucial in machine learning, as it enables the creation of robust models by dividing datasets into training and testing sets. This practice ensures that models generalize well to new, unseen data, ultimately enhancing their predictive performance. In this study `train_test_split` function is used from scikit learn library.

Data set	Percentage	Instance
Train	80%	5455
Test	20%	1364

3.4.2: Resampling techniques:

In order to address imbalance class resampling techniques are used such as: SMOTE, and SMOTE-ENN.

SMOTE:

SMOTE, or Synthetic Minority Over-sampling Technique, is a powerful tool in the realm of machine learning. SMOTE combats this issue by generating synthetic samples for the minority class, effectively balancing the dataset. SMOTE enhances the model's ability to recognize patterns within the minority class, leading to improved classification accuracy. In our study we set `sampling_strategy = 0.5`. we also fit SMOTE on training data only.

```
Original dataset shape Counter({0: 5279, 1: 176}) - Total Rows: 5455  
Resampled dataset shape Counter({0: 5279, 1: 2639}) - Total Rows: 7918
```

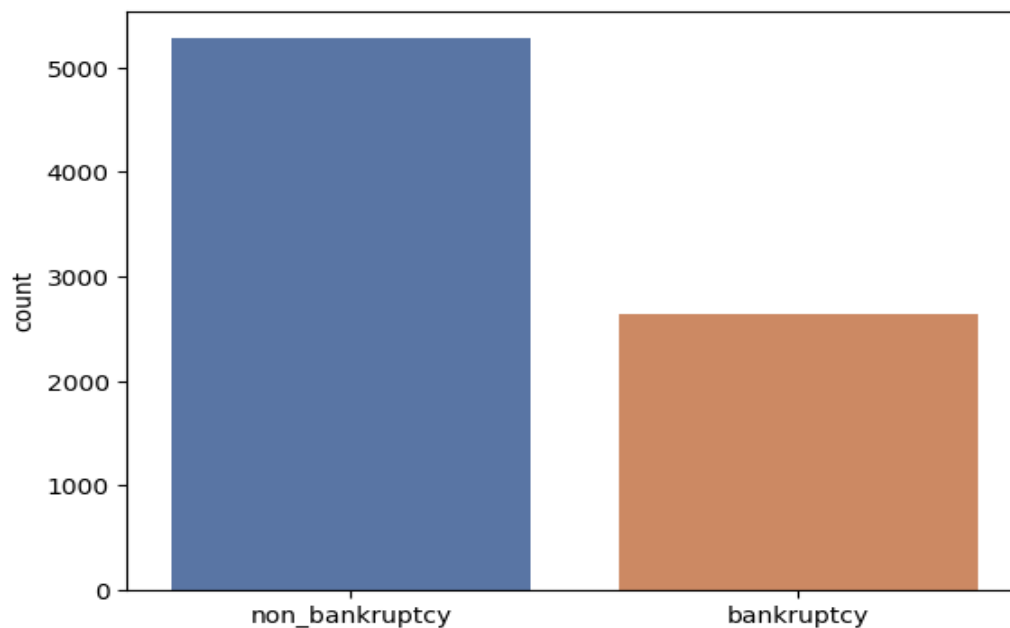


Figure 11: data set after SMOTE

SMOTE-ENN:

As we try to enhance and explain other techniques SMOTE-ENN is applied. it's a combination of two techniques: SMOTE (Synthetic Minority Over-sampling Technique) and ENN (Edited Nearest Neighbors). SMOTE-ENN is commonly used to address imbalances in datasets by oversampling the minority class using SMOTE and simultaneously cleaning the dataset by removing noise and redundant examples with ENN. This combination aims to enhance the overall performance of machine learning models in handling imbalanced datasets. By generating synthetic instances for the minority class and removing potentially misleading examples, SMOTE-ENN contributes to creating a more balanced and representative dataset, thereby improving the robustness and accuracy of classification models.

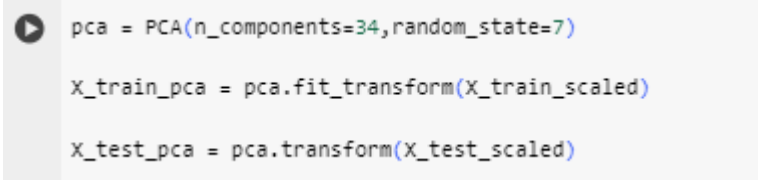
3.4.3: Standardize features:

StandardScaler is a data normalization technique in machine learning, rescaling features to have a mean of 0 and a standard deviation of 1. Applied through tools like Scikit-learn's

StandardScaler. This method, integrated with train-test splitting functions. Also, it is recommended to apply scaling before PCA(Meraj & Shetty, 2023).

3.4.4 Principal component analysis (PCA):

Principal Component Analysis (PCA) is a dimensionality reduction technique that identifies the directions of maximum variance in a dataset, producing principal components or eigenvectors.(Meraj & Shetty, 2023). PCA consider as a solution to multicollinearity in regression, transforming variables into uncorrelated principal components for dimensionality reduction.(Chan et al., 2022). In this study desired_variance set to be 0.95, which lead us to choose n_components to be 34.after that PCA fit to the train scaled data and transformed to the test scaled data.

```
pca = PCA(n_components=34,random_state=7)  
X_train_pca = pca.fit_transform(X_train_scaled)  
X_test_pca = pca.transform(X_test_scaled)
```

3.5 Modeling:

In machine learning, modeling refers to the process of creating an algorithm that captures patterns and relationships within data, enabling the system to make predictions. This involves selecting an appropriate algorithm, training it on a labeled dataset, and fine-tuning its parameters to optimize performance. In this study single models are applied such as: SVM, KNN, DT, logistic regression and Naïve bayes. Furthermore, Random Forest and Adaboost which consider ensemble techniques are applied.

3.6 Evaluating:

Evaluating the performance of a machine learning model is a crucial step in assessing its effectiveness and reliability. Model evaluation involves assessing how well the model generalizes to unseen data. In this study we used confusion metric, and ROC(AUC). We also calculate scores for F1, recall, precision, and accuracy.

3.6.1: confusion metric:

A confusion matrix is a tabular representation used to evaluate the performance of a classification model by breaking down predictions into categories: true positives, true negatives, false positives, and false negatives. It provides a clear snapshot of how well a model distinguishes between classes and where errors occur. From the confusion matrix, various performance metrics can be derived, including accuracy, precision, recall, and the

F1 score. This tool is essential for assessing the overall effectiveness of a classification model.

		Actual	
		POSITIVE	NEGATIVE
Predicted	POSITIVE	Count of TP	Count of FP
	NEGATIVE	Count of FN	Count of TN

Figure 12 confusion metrix

TP	True Positive is the number of times the model predicts the positive class accurately
FN	False Negative is the number of times the model does not predict the negative class accurately.
FP	False Positive is the number of times the model does not predict the positive class accurately
TN	True Negative is the number of times the model predicts the negative class accurately.

3.6.2 Accuracy:

Accuracy represents the proportion of correct predictions within an algorithm's entire set of predictions, and a superior model generally achieves a higher accuracy score. However, when dealing with imbalanced class distributions in the dataset, accuracy may not be a dependable metric. The accuracy calculation involves dividing the sum of true positives and true negatives by the total count of true positives, true negatives, false positives, and false negatives, as illustrated in the formula.

$$\text{Accuracy} = \frac{(TP+TN)}{TP+FP+TN+FN} \quad (3.1)$$

3.6.3 F1 score:

Precision and recall constitute the fundamental elements of the F1 score, which seeks to combine these two metrics into a unified evaluation. Notably, the F1 score is specifically crafted to maintain effectiveness in scenarios involving imbalanced data. The calculation of the F1 score follows the formulas (2), (3), and (4) presented below.

$$\text{Precision} = \frac{TP}{TP + FP} \quad (3.2)$$

$$\text{Recall} = \frac{TP}{TP + FN} \quad (3.3)$$

$$\text{F1 Score} = \frac{2 \times (\text{Precision} \times \text{Recall})}{\text{Precision} + \text{Recall}} \quad (3.4)$$

The F1 score serves as a weighted average of precision and recall, where a value of 1 indicates optimal performance, and 0 indicates the least favorable outcome.

3.6.4 Area under the ROC Curve:

The ROC curve visually illustrates a model's performance, while the Area Under the Curve (AUC) provides a quantitative measure summarizing this performance. A higher x-axis value on the ROC curve signifies an increased number of false positives compared to true negatives. This graphical representation depicts the True Positive Rate and False Positive Rate of a classifier, with optimal performance positioned at the top-left corner of the curve. Interpreting AUC involves understanding it as the probability that a randomly selected positive instance will be ranked higher than a randomly chosen negative instance by the model.

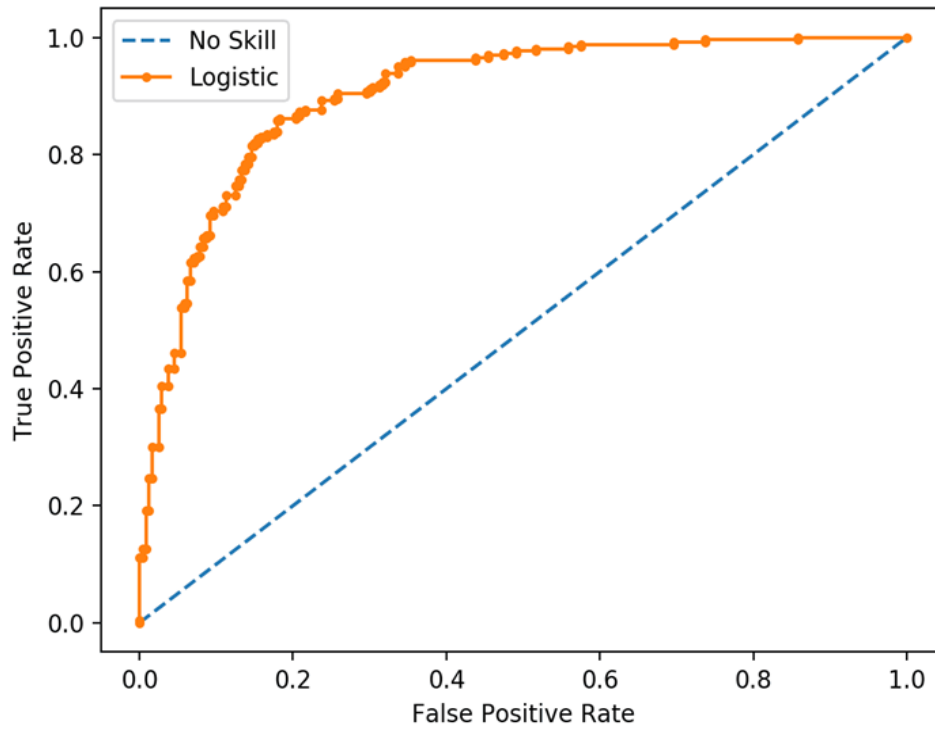


Figure 13 : ROC Curve Plot from Machine Learning Mastery website

This curve helps us to choose the best model and eliminate the rest.

$$\text{FPR} = \frac{FP}{(FP+TN)} \quad (3.5) \quad \text{TPR} = \frac{TP}{(TP+FN)} \quad (3.6)$$

Where,

False Positive Rate (FP) is equal to the X-axis

True Positive Rate (TP) is equal to the Y-axis.

3.7 Hyper parameter tuning:

Hyper parameter tuning is essential with any machine learning project. In this study we applied Random Search hyperparameter optimization technique which search for the best set of parameters to optimize the performance. Random search is often more computationally efficient than other methods such as: grid search.

Chapter four:

This chapter represent results we obtained from visualization and machine learning. It also contains discussion.

4.1Results for descriptive analysis:

Bankruptcy VS. Non-bankruptcy:

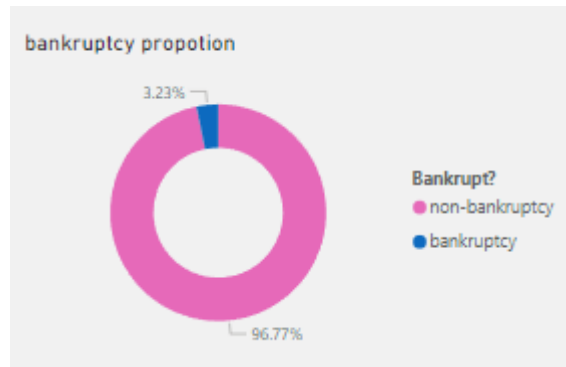


Figure 14 : Bankruptcy VS Non bankruptcy

This pie chart highlights that the proportion of bankruptcy companies and non-bankruptcy companies are 96.77%, and 3.23% respectively.

Dead Man Walking: A Company's Path to Bankrupt!:

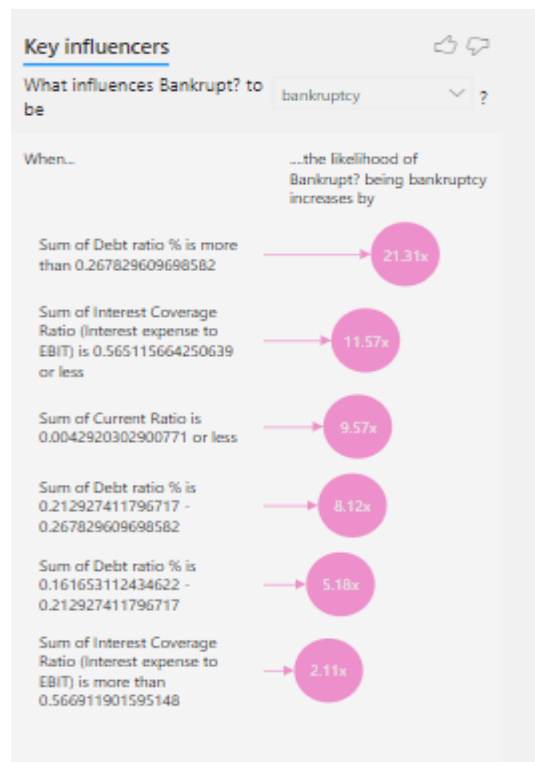


Figure 15key influencers of bankruptcy

three core factors impacting bankruptcy risk: debt burden, interest coverage, and liquidity. The company's high debt-to-asset ratio, low interest coverage ratio, and insufficient current ratio suggest a significant risk of bankruptcy.

R&D Expenditure:



Figure 17: R&D for bankruptcy companies

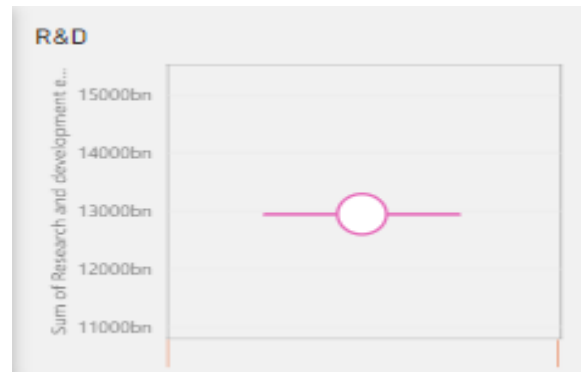


Figure 16: R&D for non-bankruptcy companies

Struggling companies near bankruptcy spent \$360 billion on research and development (R&D), while financially secure ones invested a massive \$13,000 billion in R&D. This huge gap highlights how financially healthy companies focus more on innovation. Financial well-being is crucial for allocating resources to new ideas and staying competitive. The significant difference in R&D spending paints a clear picture of the challenges faced by financially distressed companies in keeping up with their more prosperous counterparts in terms of innovation.

The Efficiency Equation: Where People Push Profits or Plunge into Bankruptcy:

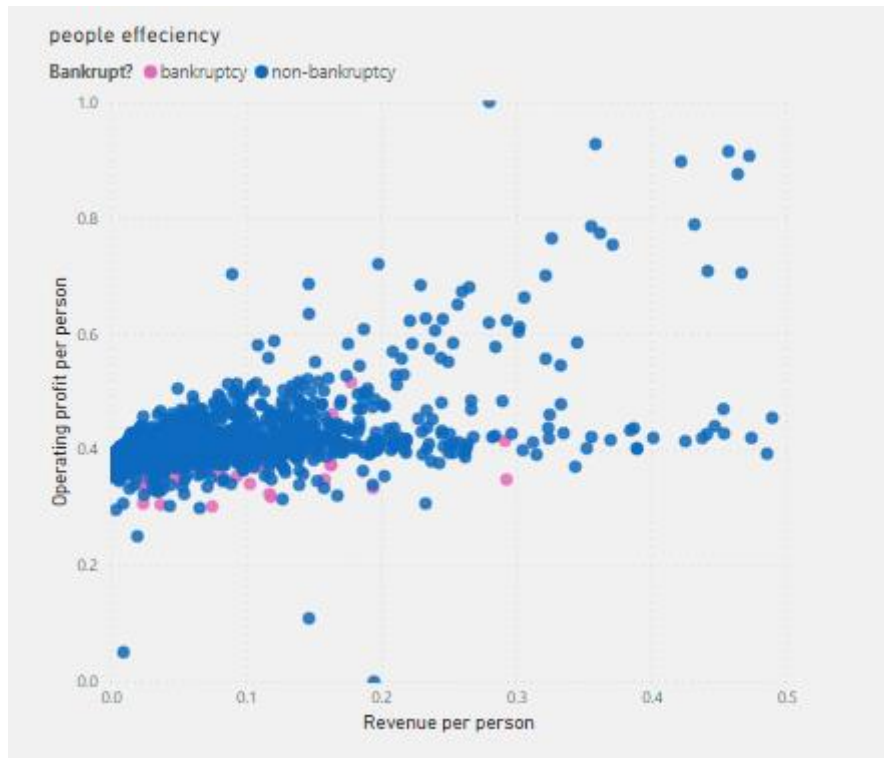


Figure 18 people efficiency and bankruptcy

The data points in the scatter plot show a general trend:

- Companies with higher operating profit per person, people efficiency, and revenue per person are more likely to be blue (not bankrupt).
- Companies with lower operating profit per person, people efficiency, and revenue per person are more likely to be red (bankrupt).

However, there are also some exceptions to this trend. There are a few red dots (bankrupt companies) in the upper right part of the chart, where companies typically have higher profitability and efficiency. This suggests that other factors, besides the three shown in the scatter plot, can also play a role in bankruptcy risk.

Risk Analysis:



Figure 19 : Risk analysis for bankruptcy companies

This heatmap explains the most factors that affect the risk of bankruptcy.

4.2 Results for models:

4.2.1 Results without resample techniques:

Firstly, models were built after using PCA without any resampling techniques. the following table show results of all metrics for models:

Model	AUC	F1 (weighted average)	Precision	Recall	Accuracy
Random Forest	0.55	0.96	0.8	0.09	0.97
Logistic Regression	0.64	0.97	0.62	0.3	0.97
AdaBoost	0.59	0.96	0.31	0.2	0.96
SVM	0.53	0.96	0.75	0.07	0.97
KNN	0.63	0.96	0.55	0.27	0.97
Decision Tree	0.69	0.96	0.37	0.41	0.96
Naive Bayes	0.77	0.96	0.34	0.57	0.95
XGBoost	0.62	0.97	0.73	0.25	0.97

Figure 4.1 results without resample techniques

- Logistic regression, and XG-boost have the highest fl score
- Naïve bayes achieved highest recall
- Random forest achieved the highest precision
- Naïve Bayes achieved highest ROC(AUC).
- We can see many models achieve 0.97 score for accuracy.

After that, resampling techniques such as: SMOTE, and SMOTE-ENN are applied trying to address imbalance in the data. As we have mentioned that class 1 is only 3.25%. in the following pages we will present the results using resample techniques.

4.2.2 Results after applying smote:

In this study we applied smote on the training data only, using resample strategy=0.5, and random_state=42. SMOTE helped in raise the minority class from 176 instance to 2639 instance. The following table shows results.

Model	AUC	F1 (weighted average)	Precision	Recall	Accuracy
Random Forest	0.64	0.96	0.45	0.30	0.97
Logistic Regression	0.82	0.94	0.24	0.73	0.92
AdaBoost	0.73	0.94	0.21	0.52	0.92
SVM	0.76	0.96	0.35	0.55	0.95
KNN	0.83	0.94	0.25	0.73	0.92
Decision Tree	0.68	0.94	0.22	0.41	0.93
Naive Bayes	0.76	0.94	0.23	0.59	0.92
XGBoost	0.73	0.96	0.39	0.48	0.96

Figure 20 results with SMOTE

- We can see that after applying smote the recall increased for all models
- All models have an accuracy of over 90%, with Random Forest, Logistic Regression, SVM, and Decision Tree having the highest accuracy of 97%.
- Random Forest has the highest F1 score of 0.96.
- XGBoost has the highest AUC of 0.83.
- Logistic Regression has the highest precision of 0.45.
- SVM has the highest recall of 0.73.

4.2.3 Results after applying SMOTE-ENN:

Although smote is widely used, smote may introduce noise, particularly when synthetic examples lie far from the true decision boundary. In response to this challenge, SMOTE-ENN integrates SMOTE with Edited Nearest Neighbors, aiming to refine the oversampling process. By combining synthetic sample generation with the removal of noisy instances through ENN. The following table show results after applying SMOTE-ENN:

Model	AUC	F1 (weighted average)	Precision	Recall	Accuracy
Random Forest	0.73	0.96	0.45	0.39	0.96
Logistic Regression	0.73	0.95	0.32	0.55	0.95
AdaBoost	0.73	0.94	0.22	0.59	0.92
SVM	0.73	0.93	0.15	0.39	0.91
KNN	0.73	0.94	0.25	0.64	0.93
Decision Tree	0.73	0.94	0.24	0.75	0.92
Naive Bayes	0.73	0.93	0.22	0.70	0.91
XGBoost	0.73	0.96	0.41	0.48	0.96
Model	0.73	0.93	0.21	0.70	0.90

Figure 21 Results with SMOTENN

- all of the models have an accuracy of over 90%. Random Forest, XGBoost, and Logistic Regression have the highest accuracy of 96%.
- Random Forest also has the highest F1 score of 0.96, while XGBoost has the highest AUC of 0.83.
- Logistic Regression has the highest precision of 0.45, while SVM has the highest recall of 0.75.

4.2.4 Champion model:

In our research we chose to focus on F1 score for models' evaluation. Logistic regression and XG-boost get 0.97 f1 score, and we chose logistic regression without resampling techniques to be the champion model it achieves higher recall and ROC(AUC) score comparing with XGBoost model. which get the same f1 score. After applying hyper parameter tuning, only the precision increased. The following table represent its scores:

	ROC(AUC)	F1 (weighted average)	Precision	Recall	Accuracy
Before hyper parameter tuning	0.64	0.97	0.62	0.3	0.97
After hyper parameter tuning	0.65	0.97	0.72	0.3	0.97

we can see that after applying hyper parameter tuning, roc(auc) score increased, and also the precision has a remarkable increase.

4.3 Discussion:

In our study we applied data analysis and machine learning to predict bankruptcy.

Bankruptcy happens when an organization being unable to repay its obligations.

Bankruptcy can affect stakeholders such as: investors, customers, employees,..etc. it can also consider as a risk for lending institutions, and the overall economy. Predicting bankruptcy can be useful for government, lending instructions, and credit assess institutions.

Taiwan serves as a compelling research location for bankruptcy studies due to its economic significance in the global technology sector, diverse corporate landscape ranging from large multinational corporations to SMEs, and the country's pivotal role in the international electronics supply chain. Furthermore, the vibrant entrepreneurial ecosystem in Taiwan, particularly in the technology sector, offers a unique perspective on the challenges faced by innovative startups in times of financial distress, contributing to a comprehensive understanding of bankruptcy dynamics in a globally connected economy.

After applying machine learning, we found that logistic regression model was the champion model achieving f1 score equal 0.97. hyperparameter tuning using Randomized SearchCV is applied to get the best parameters for logistic regression.

In this study, in the preprocessing stage, outliers are capped, features are scaled, and dimensionality reduction using PCA is applied as the data contain 96 features. Resampling techniques includes SMOTE, and SMOTE-ENN helped to address imbalance class issue achieving higher recall for almost of models.

Using power bi for data analysis, we found that bankruptcy assessment using debt ratio, interest coverage ratio, and liquidity ratio is essential. R&D is an important factor to consider to be innovative and able to compete in the market, so that companies can avoid failing in bankruptcy. Hiring skilled personnel is also essential as human capital is the most important factor in any organization.

At the end, this study can be useful for financial institution to assess bankruptcy and avoid bad debt. It is also essential for any institution to assess its financial position

Chapter 5

This chapter represent recommendations, limitation, and conclusion.

5.1 Conclusion:

Prediction of bankruptcy is essential for many organizations, and all of the stakeholders. this study on bankruptcy has provided valuable insights into predicting financial distress. The champion model, Logistic Regression emerged as a robust tool for this purpose. By exploring various factors and employing advanced machine learning techniques. The predictive capabilities of the LR model underscore its effectiveness in identifying early signs of bankruptcy.in addition to visualization analysis provide essential insights. We also tried to address class imbalance using SMOTE, and SMOTE-ENN. However, it is important to acknowledge the study's limitations, such as the absence of economic factors in the data and the limited exploration of alternative classifiers. Future research endeavors should strive to address these limitations, and further exploration into deep learning methodologies could enhance the predictive precision of bankruptcy models.

5.2 Limitation:

Several limitations are inherent in this study that warrant consideration. Firstly, it is crucial to acknowledge the absence of economic factors within the dataset, thereby limiting the comprehensive understanding of the studied phenomenon. Additionally, the dataset exhibited an inherent imbalance, potentially affecting the robustness and generalizability of the findings and there many resampling techniques do not cover in this study. The study utilized a specific set of classifiers, and it is important to note that alternative methods were not exhaustively explored. Future research could benefit from a more comprehensive exploration of classifier options to discern the most suitable model for the given dataset. Moreover, while the current study employed machine learning techniques, the exploration of deep learning methodologies was not within its scope. This represents an avenue for further investigation, as the application of deep learning could provide valuable insights and potentially enhance the predictive capabilities of the models.

5.3 Recommendations:

- Any organization interested in assessing whether a company will face bankruptcy or not, should consider 3 factors: debt ratio, interest coverage ratio, and liquidity.
- It is recommended for organizations to invest in their human capital, as people efficiency may lead to bankruptcy.
- R&D is essential to be innovative, stay competitive, and avoid bankruptcy.
- Utilizing machine learning to assess bankruptcy: Logistic regression model performs well in this study, so it is recommended to be used.
- Future research can apply more resampling techniques to address the problem of class imbalance, using other feature selection or dimensionality reduction techniques, and applying other machine learning classifiers.
- Future research should consider economic factors as this data contain only financial factors.

References:

- Aljawazneh, H., Mora, A., García-Sánchez, P., & Castillo-Valdivieso, P. (2021). Comparing the performance of deep learning methods to predict companies' financial failure. *IEEE Access*, 9, 97010–97038.
- Bărbuță-Mișu, N., & Madaleno, M. (2020). Assessment of bankruptcy risk of large companies: European countries evolution analysis. *Journal of Risk and Financial Management*, 13(3), 58.
- Brenes, R. F., Johannssen, A., & Chukhrova, N. (2022). An intelligent bankruptcy prediction model using a multilayer perceptron. *Intelligent Systems with Applications*, 200136.
- Cervantes, J., Garcia-Lamont, F., Rodríguez-Mazahua, L., & Lopez, A. (2020). A comprehensive survey on support vector machine classification: Applications, challenges and trends. *Neurocomputing*, 408, 189–215.
- Chan, J. Y.-L., Leow, S. M. H., Bea, K. T., Cheng, W. K., Phoong, S. W., Hong, Z.-W., & Chen, Y.-L. (2022). Mitigating the multicollinearity problem and its machine learning approach: A review. *Mathematics*, 10(8), 1283.
- Chen, T., & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. *Proceedings of the 22nd Acm Sigkdd International Conference on Knowledge Discovery and Data Mining*, 785–794.
- Cunningham, P., & Delany, S. J. (2021). K-Nearest neighbour classifiers-A Tutorial. *ACM Computing Surveys (CSUR)*, 54(6), 1–25.
- Dasilas, A., & Rigani, A. (n.d.). Machine Learning Techniques in Bankruptcy Prediction: A Systematic Literature Review. *Available at SSRN 4577883*.
- Devi, S. S., & Radhika, Y. (2018). A survey on machine learning and statistical techniques in bankruptcy prediction. *International Journal of Machine Learning and Computing*, 8(2), 133–139.

- Lin, W.-C., Lu, Y.-H., & Tsai, C.-F. (2019). Feature selection in single and ensemble learning-based bankruptcy prediction models. *Expert Systems*, 36(1), e12335.
- Meraj, S. B. S., & Shetty, B. (2023). Successful Data Mining: With Dimension Reduction. *International Conference on Applications of Machine Intelligence and Data Analytics (ICAMIDA 2022)*, 11–22.
- Muslim, M. A., Dasril, Y., Javed, H., Abror, W. F., Pertiwi, D. A. A., Mustaqim, T., & others. (2023). An Ensemble Stacking Algorithm to Improve Model Accuracy in Bankruptcy Prediction. *Journal of Data Science and Intelligent Systems*.
- Parmar, A., Katariya, R., & Patel, V. (2019). A review on random forest: An ensemble classifier. *International Conference on Intelligent Data Communication Technologies and Internet of Things (ICICI) 2018*, 758–763.
- Shetty, S. H., Shetty, S., Singh, C., & Rao, A. (2022). Supervised Machine Learning: Algorithms and Applications. *Fundamentals and Methods of Machine and Deep Learning: Algorithms, Tools and Applications*, 1–16.
- Shi, Y., & Li, X. (2019). An overview of bankruptcy prediction models for corporate firms: A systematic literature review. *Intangible Capital*, 15(2), 114–127.
- Song, Y.-Y., & Ying, L. (2015). Decision tree methods: Applications for classification and prediction. *Shanghai Archives of Psychiatry*, 27(2), 130.
- Soofi, A. A., & Awan, A. (2017). Classification techniques in machine learning: Applications and issues. *Journal of Basic & Applied Sciences*, 13(1), 459–465.
- Sundus, K. I., Hammo, B. H., Al-Zoubi, M. B., & Al-Omari, A. (2022). Solving the multicollinearity problem to improve the stability of machine learning algorithms applied to a fully annotated breast cancer dataset. *Informatics in Medicine Unlocked*, 33, 101088. <https://doi.org/10.1016/j.imu.2022.101088>
- Tsai, C.-F., Sue, K.-L., Hu, Y.-H., & Chiu, A. (2021). Combining feature selection, instance selection, and ensemble classification techniques for improved financial distress prediction. *Journal of Business Research*, 130, 200–209.
- Wang, H., & Liu, X. (2021). Undersampling bankruptcy prediction: Taiwan bankruptcy data. *Plos One*, 16(7), e0254030.
- Wang, J., Yang, J., Iverson, B. C., & Kluender, R. (2020). Bankruptcy and the COVID-19 Crisis. *Available at SSRN 3690398*.
- Webb, G. I., Keogh, E., & Miikkulainen, R. (2010). Naïve Bayes. *Encyclopedia of Machine Learning*, 15(1), 713–714.
- Wyner, A. J., Olson, M., Bleich, J., & Mease, D. (2017). Explaining the success of adaboost and random forests as interpolating classifiers. *The Journal of Machine Learning Research*, 18(1), 1558–1590.
- Yigit, F. (2018). Bankruptcy: An examination of different approaches. *Global Approaches in Financial Economics, Banking, and Finance*, 293–308.

Appendix

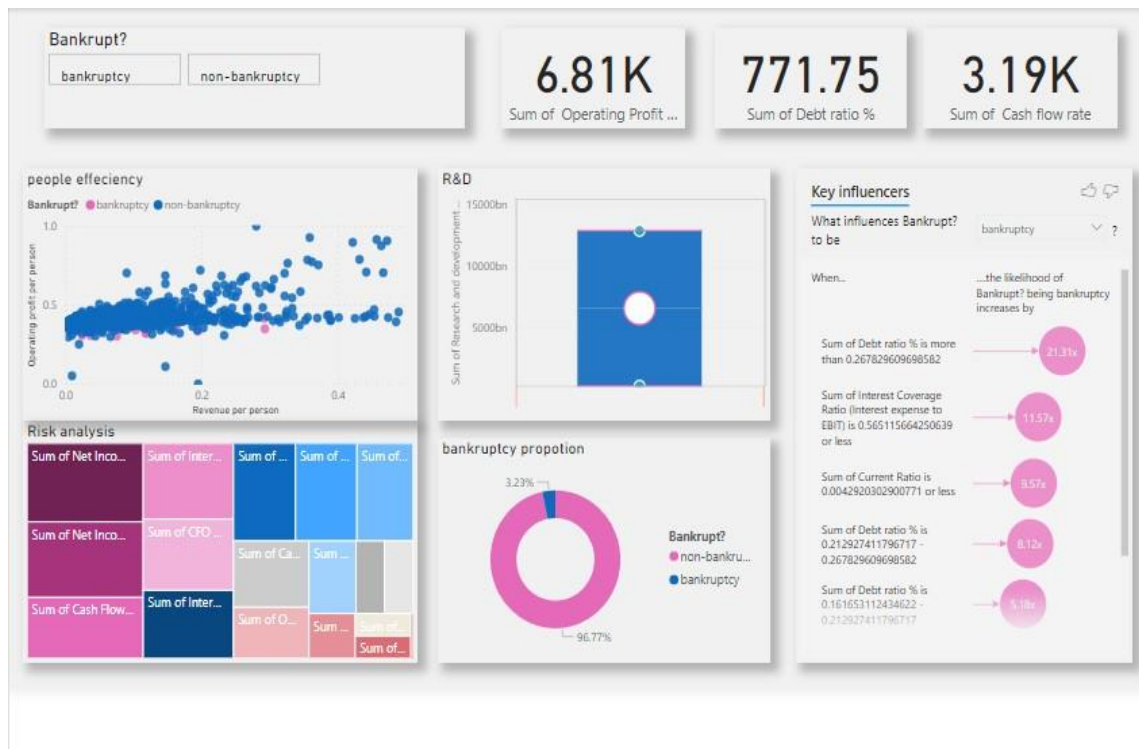


Figure 22 dashboard using power bi

□ Data acquization

```
from google.colab import drive
drive.mount('/content/drive')
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content

□ import liab

```
#import liabrieries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

#spilting data
from sklearn.model_selection import train_test_split

#SMOTE & undersample
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler

#Import library for VIF
from statsmodels.stats.outliers_influence import variance_inflation_factor

# liaberieries dealing with SCALING
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

#metrics
from sklearn import metrics
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score, accuracy_score, class
from mlxtend.evaluate import confusion_matrix
from mlxtend.plotting import plot_confusion_matrix
from sklearn.pipeline import make_pipeline

#MODELS
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
import xgboost as xgb
from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import cross_val_score, train_test_split
```

□ read data

```
df= pd.read_csv("/content/drive/MyDrive/مستند من Marian Saad")
df
```

		ROA(C)	ROA(A)	ROA(B)	Operatin
	Bankrupt?	before interest and depreciation	before interest and % tax after	before interest and depreciation after tax	Gros Margi
0	1	0.370594	0.424389	0.405750	0.60145
1	1	0.464291	0.538214	0.516730	0.61023
2	1	0.426071	0.499019	0.472295	0.60145
3	1	0.399844	0.451265	0.457733	0.58354
4	1	0.465022	0.538432	0.522298	0.59878
...
6814	0	0.493687	0.539468	0.543230	0.60445
6815	0	0.475162	0.538269	0.524172	0.59830
6816	0	0.472725	0.533744	0.520638	0.61044
6817	0	0.506264	0.559911	0.554045	0.60785
6818	0	0.493053	0.570105	0.549548	0.62740

6819 rows x 96 columns

□ Data Exploration

□ Description

```
df.describe()
```

		ROA(C)	ROA(A)	ROA(B)	
	Bankrupt?	before interest and depreciation	before interest and % after tax	before interest and depreciation after tax	
count	6819.000000	6819.000000	6819.000000	6819.000000	681
mean	0.032263	0.505180	0.558625	0.553589	
std	0.176710	0.060686	0.065620	0.061595	
min	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.476527	0.535543	0.527277	
50%	0.000000	0.502706	0.559802	0.552278	
75%	0.000000	0.535563	0.589157	0.584105	
max	1.000000	1.000000	1.000000	1.000000	

8 rows x 96 columns

df.info()

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 6819 entries, 0 to 6818
```

```
Data columns (total 96 columns):
```

#	Column	Non-Null Count	Dtype
0	Bankrupt?	6819 non-null	int64
1	ROA(C) before interest and depreciation before interest	6819 non-null	float64
2	ROA(A) before interest and % after tax	6819 non-null	float64
3	ROA(B) before interest and depreciation after tax	6819 non-null	float64

4	Operating Gross Margin	6819	non-null	float64
5	Realized Sales Gross Margin	6819	non-null	float64
6	Operating Profit Rate	6819	non-null	float64
7	Pre-tax net Interest Rate	6819	non-null	float64
8	After-tax net Interest Rate	6819	non-null	float64
9	Non-industry income and expenditure/revenue	6819	non-null	float64
10	Continuous interest rate (after tax)	6819	non-null	float64
11	Operating Expense Rate	6819	non-null	float64
12	Research and development expense rate	6819	non-null	float64
13	Cash flow rate	6819	non-null	float64
14	Interest-bearing debt interest rate	6819	non-null	float64
15	Tax rate (A)	6819	non-null	float64
16	Net Value Per Share (B)	6819	non-null	float64
17	Net Value Per Share (A)	6819	non-null	float64
18	Net Value Per Share (C)	6819	non-null	float64
19	Persistent EPS in the Last Four Seasons	6819	non-null	float64
20	Cash Flow Per Share	6819	non-null	float64
21	Revenue Per Share (Yuan ¥)	6819	non-null	float64
22	Operating Profit Per Share (Yuan ¥)	6819	non-null	float64
23	Per Share Net profit before tax (Yuan ¥)	6819	non-null	float64
24	Realized Sales Gross Profit Growth Rate	6819	non-null	float64
25	Operating Profit Growth Rate	6819	non-null	float64
26	After-tax Net Profit Growth Rate	6819	non-null	float64
27	Regular Net Profit Growth Rate	6819	non-null	float64
28	Continuous Net Profit Growth Rate	6819	non-null	float64
29	Total Asset Growth Rate	6819	non-null	float64
30	Net Value Growth Rate	6819	non-null	float64
31	Total Asset Return Growth Rate Ratio	6819	non-null	float64
32	Cash Reinvestment %	6819	non-null	float64
33	Current Ratio	6819	non-null	float64
34	Quick Ratio	6819	non-null	float64
35	Interest Expense Ratio	6819	non-null	float64
36	Total debt/Total net worth	6819	non-null	float64
37	Debt ratio %	6819	non-null	float64
38	Net worth/Assets	6819	non-null	float64
39	Long-term fund suitability ratio (A)	6819	non-null	float64
40	Borrowing dependency	6819	non-null	float64
41	Contingent liabilities/Net worth	6819	non-null	float64
42	Operating profit/Paid-in capital	6819	non-null	float64
43	Net profit before tax/Paid-in capital	6819	non-null	float64
44	Inventory and accounts receivable/Net value	6819	non-null	float64
45	Total Asset Turnover	6819	non-null	float64
46	Accounts Receivable Turnover	6819	non-null	float64
47	Average Collection Days	6819	non-null	float64
48	Inventory Turnover Rate (times)	6819	non-null	float64
49	Fixed Assets Turnover Frequency	6819	non-null	float64
50	Net Worth Turnover Rate (times)	6819	non-null	float64
51	Revenue per person	6819	non-null	float64
52	Operating profit per person	6819	non-null	float64

```
df.describe(include='all')
```

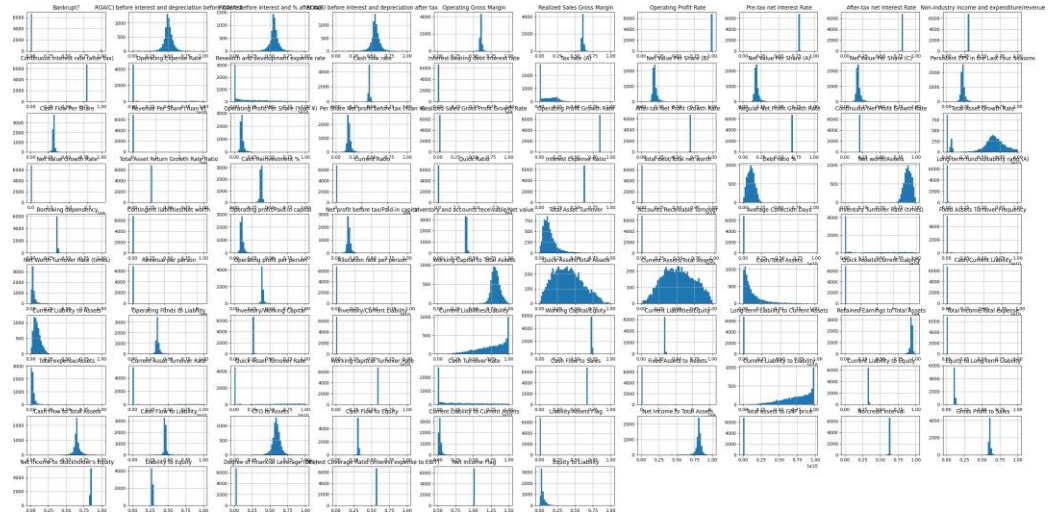
	Bankrupt?	ROA(C) before interest and depreciation	ROA(A) before interest and % aftertax	ROA(B) before Ointerest and depreciation aftertax	
count	6819.000000	6819.000000	6819.000000	6819.000000	681
mean	0.032263	0.505180	0.558625	0.553589	
std	0.176710	0.060686	0.065620	0.061595	
min	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.476527	0.535543	0.527277	
50%	0.000000	0.502706	0.559802	0.552278	
75%	0.000000	0.535563	0.589157	0.584105	
max	1.000000	1.000000	1.000000	1.000000	

8 rows x 96 columns

- Plotting the distribution

```
# prompt: draw histogram for each feature in the data
```

```
df.hist(bins = 50, figsize = (40, 20));
```



- Data preprocessing

```
df= df.copy()
```

- ❑ Dropping unuseful columns

```
df[' Net Income Flag'].value_counts()
```

```
1      6819
Name: Net Income Flag, dtype: int64
```

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

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

this two columns consider as unnecessary. they are categorical columns which are encoded. the first one contain only one value. the second column has a very large class imbalance and we can drop both of them

```
df.drop(columns=[' Net Income Flag', ' Liability-Assets Flag'],inplace=True)
```

df

		ROA(C)	ROA(A)	ROA(B)	Operatin
	Bankrupt?	before interest and depreciation	before interest and % after tax	before interest and depreciation after tax	Gros Margi
0	1	0.370594	0.424389	0.405750	0.60145
1	1	0.464291	0.538214	0.516730	0.61023
2	1	0.426071	0.499019	0.472295	0.60145
3	1	0.399844	0.451265	0.457733	0.58354
4	1	0.465022	0.538432	0.522298	0.59878
...
6814	0	0.493687	0.539468	0.543230	0.60445
6815	0	0.475162	0.538269	0.524172	0.59830
6816	0	0.472725	0.533744	0.520638	0.61044
6817	0	0.506264	0.559911	0.554045	0.60785
6818	0	0.493053	0.570105	0.549548	0.62740

6819 rows x 94 columns

□ checking for null values and duplications.

```
print(df.isna().sum().sum())
print(np.isnan(df).sum().sum())
print(df.isnull().sum().sum())
```

```
0
0
0
```

```
df.isnull().sum()
```

```
Bankrupt? 0
ROA(C) before interest and depreciation before interest 0
ROA(A) before interest and % after tax 0
ROA(B) before interest and depreciation after tax 0
Operating Gross Margin 0
..
Net Income to Stockholder's Equity 0
Liability to Equity 0
Degree of Financial Leverage (DFL) 0
Interest Coverage Ratio (Interest expense to EBIT) 0
Equity to Liability 0
Length: 94, dtype: int64
```

```
df.duplicated().sum()
```

```
0
```

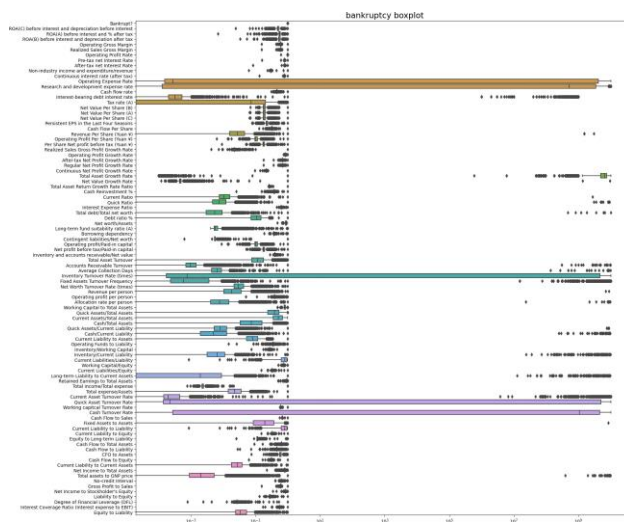
```
df
```

		ROA(C)	ROA(A)	ROA(B)	Operatin
	Bankrupt?	before interest and depreciation	before interest and % tax after	before interest and depreciation after tax	Gros Margi
0	1	0.370594	0.424389	0.405750	0.60145
1	1	0.464291	0.538214	0.516730	0.61023
2	1	0.426071	0.499019	0.472295	0.60145
3	1	0.399844	0.451265	0.457733	0.58354
4	1	0.465022	0.538432	0.522298	0.59878
...
6814	0	0.493687	0.539468	0.543230	0.60445
6815	0	0.475162	0.538269	0.524172	0.59830
6816	0	0.472725	0.533744	0.520638	0.61044
6817	0	0.506264	0.559911	0.554045	0.60785
6818	0	0.493053	0.570105	0.549548	0.62740

6819 rows x 94 columns

□ outliers capping

```
plt.figure(figsize=(20,20))
ax=sns.boxplot(data=df,orient="h")
ax.set_title("bankruptcy boxplot",fontsize="18")
ax.set(xscale="log")
plt.show()
```



```
def detect_outliers_iqr(column):
    q1 = np.percentile(column, 25)
    q3 = np.percentile(column, 75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    outliers = column[(column < lower_bound) | (column > upper_bound)]
    return outliers

outlier_columns = []
for column in df.iloc[:, 1:].columns:
    outliers = detect_outliers_iqr(df.iloc[:, 1:][column])
    if len(outliers) > 0:
        outlier_columns.append(column)

outlier_columns = []
for column in df.iloc[:, 1:].columns:
    outliers = detect_outliers_iqr(df.iloc[:, 1:][column])
    if len(outliers) > 0:
        outlier_columns.append(column)

print("Columns with outliers:")
for column in outlier_columns:
    print(column)
```

Columns with outliers:

ROA(C) before interest and depreciation before interest
ROA(A) before interest and % after tax
ROA(B) before interest and depreciation after tax
Operating Gross Margin
Realized Sales Gross Margin
Operating Profit Rate
Pre-tax net Interest Rate
After-tax net Interest Rate
Non-industry income and expenditure/revenue
Continuous interest rate (after tax)
Research and development expense rate
Cash flow rate
Interest-bearing debt interest rate
Tax rate (A)
Net Value Per Share (B)
Net Value Per Share (A)
Net Value Per Share (C)
Persistent EPS in the Last Four Seasons
Cash Flow Per Share
Revenue Per Share (Yuan ¥)
Operating Profit Per Share (Yuan ¥)
Per Share Net profit before tax (Yuan ¥)
Realized Sales Gross Profit Growth Rate
Operating Profit Growth Rate
After-tax Net Profit Growth Rate
Regular Net Profit Growth Rate
Continuous Net Profit Growth Rate
Total Asset Growth Rate
Net Value Growth Rate
Total Asset Return Growth Rate Ratio
Cash Reinvestment %
Current Ratio
Quick Ratio
Interest Expense Ratio
Total debt/Total net worth
Debt ratio %
Net worth/Assets
Long-term fund suitability ratio (A)
Borrowing dependency
Contingent liabilities/Net worth
Operating profit/Paid-in capital
Net profit before tax/Paid-in capital
Inventory and accounts receivable/Net value
Total Asset Turnover
Accounts Receivable Turnover
Average Collection Days
Fixed Assets Turnover Frequency
Net Worth Turnover Rate (times)
Revenue per person
Operating profit per person
Allocation rate per person
Working Capital to Total Assets
Quick Assets/Total Assets
Cash/Total Assets
Quick Assets/Current Liability
Cash/Current Liability
Current Liability to Assets

```

# Define the function to detect outliers using the IQR method
def detect_outliers_iqr(column):
    q1 = np.percentile(column, 25)
    q3 = np.percentile(column, 75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    outliers = column[(column < lower_bound) | (column > upper_bound)]
    return outliers

# Initialize a counter for columns with outliers
outlier_columns_count = 0

# Iterate over each column in the DataFrame and check for outliers
for column in df.iloc[:, 1:].columns:
    outliers = detect_outliers_iqr(df.iloc[:, 1:][column])
    if len(outliers) > 0:
        outlier_columns_count += 1

# Print the count of columns with outliers
print("Number of columns with outliers:", outlier_columns_count)

# Initialize a counter for columns without outliers
columns_without_outliers_count = 0

# Iterate over each column in the DataFrame and check for outliers
for column in df.iloc[:, 1:].columns:
    outliers = detect_outliers_iqr(df.iloc[:, 1:][column])
    if len(outliers) == 0:
        columns_without_outliers_count += 1

# Print the count of columns without outliers
print("Number of columns without outliers:", columns_without_outliers_count)

    Number of columns with outliers: 88
    Number of columns without outliers: 5

# Define the function to cap outliers using the IQR method
def cap_outliers_iqr(column):
    q1 = np.percentile(column, 25)
    q3 = np.percentile(column, 75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    capped_column = np.where(column < lower_bound, lower_bound, column)
    capped_column = np.where(column > upper_bound, upper_bound, capped_column)
    return capped_column

# Iterate over each column in the DataFrame and cap outliers
for column in df.columns[1:]:
    df[column] = cap_outliers_iqr(df[column])

# Print the modified DataFrame with capped outliers
df_cleaned = df
df_cleaned

```


		ROA(C)	ROA(A)	ROA(B)	Operatin
	Bankrupt?	before interest and depreciation	before interest and % tax after	before interest and depreciation after tax	Gros Margi
0	1	0.387973	0.455122	0.442034	0.60145
1	1	0.464291	0.538214	0.516730	0.61023
2	1	0.426071	0.499019	0.472295	0.60145
3	1	0.399844	0.455122	0.457733	0.58354
4	1	0.465022	0.538432	0.522298	0.59878
...
6814	0	0.493687	0.539468	0.543230	0.60445
6815	0	0.475162	0.538269	0.524172	0.59830
6816	0	0.472725	0.533744	0.520638	0.61044
6817	0	0.506264	0.559911	0.554045	0.60785
6818	0	0.493053	0.570105	0.549548	0.62740

6819 rows x 94 columns

```
def detect_outliers_iqr(column):
    q1 = np.percentile(column, 25)
    q3 = np.percentile(column, 75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    outliers = column[(column < lower_bound) | (column > upper_bound)]
    return outliers

# Initialize a counter for columns with outliers
outlier_columns_count = 0

# Iterate over each column in the DataFrame and check for outliers
for column in df_cleaned.columns:
    outliers = detect_outliers_iqr(df_cleaned[column])
    if len(outliers) > 0:
        outlier_columns_count += 1

# Print the count of columns with outliers
print("Number of columns with outliers:", outlier_columns_count)

Number of columns with outliers: 1

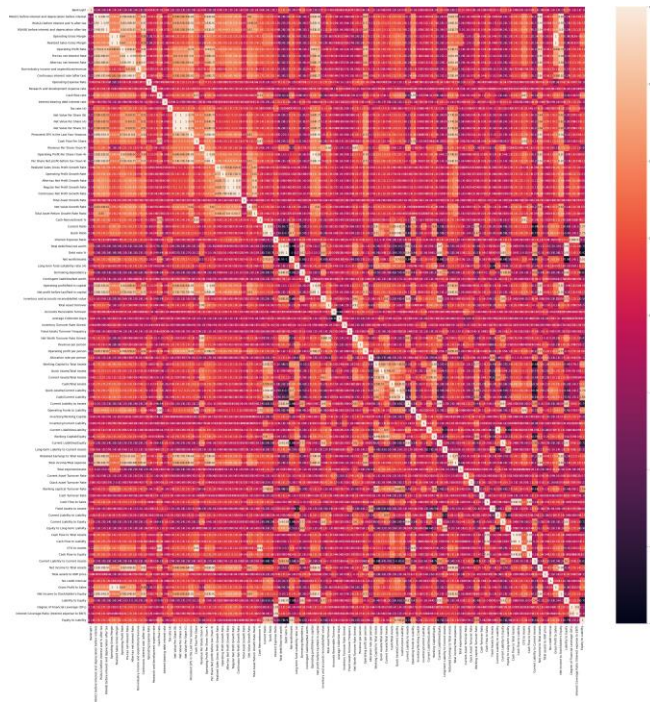
df_cleaned
```

		ROA(C)	ROA(A)	ROA(B)	Operatin
	Bankrupt?	before interest and depreciation	before interest and % tax after	before interest and depreciation after tax	Gros Margi
0	1	0.387973	0.455122	0.442034	0.60145
1	1	0.464291	0.538214	0.516730	0.61023
2	1	0.426071	0.499019	0.472295	0.60145
3	1	0.399844	0.455122	0.457733	0.58354
4	1	0.465022	0.538432	0.522298	0.59878
...
6814	0	0.493687	0.539468	0.543230	0.60445
6815	0	0.475162	0.538269	0.524172	0.59830
6816	0	0.472725	0.533744	0.520638	0.61044
6817	0	0.506264	0.559911	0.554045	0.60785
6818	0	0.493053	0.570105	0.549548	0.62740

6819 rows x 94 columns

□ Checking for Multicolunarity

```
df_cleaned.corr()
plt.figure(figsize=(40,40))
sns.heatmap(df_cleaned.corr(),annot=True)
plt.show()
```



```
def calc_vif(X):

    # Calculating VIF
    vif = pd.DataFrame()
    vif["variables"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

    return(vif)

X = df_cleaned.iloc[:,1:]
vif_result = calc_vif(X)
vif_result
```

	variables	VIF
0	ROA(C) before interest and depreciation befor	1.114779e+02
1	ROA(A) before interest and % after tax	1.361600e+02
2	ROA(B) before interest and depreciation after	1.645285e+02
3	Operating Gross Margin	1.493994e+07
4	Realized Sales Gross Margin	7.048890e+02
...
88	Net Income to Stockholder's Equity	3.297765e+01
89	Liability to Equity	9.358246e+01
90	Degree of Financial Leverage (DFL)	6.283449e+00
91	Interest Coverage Ratio (Interest expense to	9.305971e+00
92	Equity to Liability	2.206786e+0193

rows x 2 columns

```
# Sort features based on VIF values in descending order
sorted_vif_result = vif_result.sort_values(by='VIF', ascending=False)
```

```
# Display the sorted result
print("Sorted VIF values:")
print(sorted_vif_result)
```

Sorted VIF values:

	variables	VIF
63	Current Liabilities/Liability	1.730250e+10
76	Current Liability to Liability	8.121926e+09
36	Debt ratio %	2.363178e+07
37	Net worth/Assets	2.216773e+07
65	Current Liabilities/Equity	2.033086e+07
..
47	Inventory Turnover Rate (times)	1.289733e+00
73	Cash Turnover Rate	1.211162e+00
13	Interest-bearing debt interest rate	1.189972e+00
11	Research and development expense rate	1.169626e+00
28	Total Asset Growth Rate	1.162967e+00

[93 rows x 2 columns]

```
# Count features with VIF higher than 10
high_vif_features = vif_result[vif_result['VIF'] > 10]
count_high_vif_features = len(high_vif_features)

print("Number of features with VIF > 10:", count_high_vif_features)
print("\nFeatures with VIF > 10:")
print(high_vif_features)
```

Number of features with VIF > 10: 53

Features with VIF > 10:

	variables	VIF
0	ROA(C) before interest and depreciation befor...	1.114779e+02
1	ROA(A) before interest and % after tax	1.361600e+02
2	ROA(B) before interest and depreciation after...	1.645285e+02
3	Operating Gross Margin	1.493994e+07
4	Realized Sales Gross Margin	7.048890e+02
5	Operating Profit Rate	1.612337e+01
6	Pre-tax net Interest Rate	9.460162e+01
7	After-tax net Interest Rate	1.038006e+02
9	Continuous interest rate (after tax)	4.228036e+01
12	Cash flow rate	1.990130e+01
15	Net Value Per Share (B)	1.689846e+03
16	Net Value Per Share (A)	3.322708e+03
17	Net Value Per Share (C)	1.660584e+03
18	Persistent EPS in the Last Four Seasons	8.661834e+01
19	Cash Flow Per Share	1.196747e+01
20	Revenue Per Share (Yuan ¥)	1.573538e+01

21	Operating Profit Per Share (Yuan ¥)	8.675020e+02
22	Per Share Net profit before tax (Yuan ¥)	1.183813e+02
25	After-tax Net Profit Growth Rate	1.527574e+02
26	Regular Net Profit Growth Rate	1.536113e+02
31	Cash Reinvestment %	1.171284e+01
32	Current Ratio	3.727791e+01
33	Quick Ratio	3.008873e+01
35	Total debt/Total net worth	4.192465e+01
36	Debt ratio %	2.363178e+07
37	Net worth/Assets	2.216773e+07
41	Operating profit/Paid-in capital	8.610412e+02
42	Net profit before tax/Paid-in capital	1.801319e+02
43	Inventory and accounts receivable/Net value	1.532803e+01
44	Total Asset Turnover	2.401073e+01
49	Net Worth Turnover Rate (times)	2.528351e+01
53	Working Capital to Total Assets	3.123345e+02
54	Quick Assets/Total Assets	1.801246e+01
55	Current Assets/Total Assets	3.055080e+02
57	Quick Assets/Current Liability	5.110057e+01
59	Current Liability to Assets	2.379598e+02
60	Operating Funds to Liability	2.405329e+01
63	Current Liabilities/Liability	1.730250e+10
64	Working Capital/Equity	1.964810e+01
65	Current Liabilities/Equity	2.033086e+07
72	Working capital Turnover Rate	1.021077e+01
76	Current Liability to Liability	8.121926e+09
77	Current Liability to Equity	2.009854e+07
79	Cash Flow to Total Assets	6.386893e+01
80	Cash Flow to Liability	1.758343e+01
81	CF0 to Assets	3.458205e+01
82	Cash Flow to Equity	2.714723e+01
83	Current Liability to Current Assets	1.748758e+01
84	Net Income to Total Assets	1.352145e+02
87	Gross Profit to Sales	1.500773e+07
88	Net Income to Stockholder's Equity	3.297765e+01
89	Liability to Equity	9.358246e+01
92	Equity to Liability	2.206786e+01

□ splitting data

```
from pickle import DEFAULT_PROTOCOL
X = df_cleaned.drop('Bankrupt?', axis = 1)
y = df_cleaned['Bankrupt?'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
print(X_train.shape)
print(X_test.shape)

(5455, 93)
(1364, 93)
```

□ Standard Scaler

```
s_scaler = StandardScaler()
s_scaler.fit(X_train)
X_train_scaled = s_scaler.transform(X_train)
X_test_scaled = s_scaler.transform(X_test)

X_train_scaled = pd.DataFrame(X_train_scaled, columns=X_train.columns)
X_test_scaled
```

	ROA(C)	ROA(A)	ROA(B)	Operating	Realized
	before interest and depreciation	before interest and % tax	before interest and depreciation after tax	Gross Margin	Sales Gross Margin
0	-0.218792	-0.194214	-0.189539	-0.862987	-0.863196
1	0.574079	0.619855	0.385417	-0.790499	-0.802602
2	0.417979	0.542272	0.247556	-0.097841	-0.095676
3	-0.688993	-0.460836	-0.659763	-1.156293	-1.157412
4	-2.214769	-2.099082	-2.237420	2.451974	2.446732
...
5450	0.880566	0.081149	0.786177	-0.518671	-0.517812
5451	-0.456748	-0.635670	-0.481292	-0.173013	-0.169062
5452	2.290220	2.199641	2.300246	0.605556	-0.412110
5453	1.598243	1.650281	1.382507	0.184726	0.186421
5454	-0.078873	0.135784	-0.077326	0.108211	0.100243

5455 rows x 93 columns

```
X_test_scaled = pd.DataFrame(X_test_scaled, columns=X_test.columns)
X_test_scaled
```

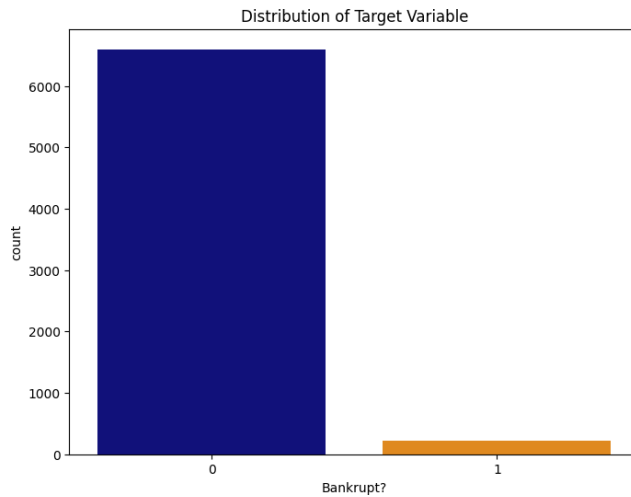
	ROA(C)	ROA(A)	ROA(B)	Operating	Realized
	before interest and depreciation	before interest and % tax	before interest and depreciation after tax	Gross Margin	Sales Gross Margin
0	-0.034137	-0.074016	-0.036716	0.149824	0.152758
1	-0.371084	-0.242294	-0.468468	-0.836140	-0.836265
2	0.964327	1.176045	1.106784	0.006192	0.011372
3	2.325913	2.199641	2.300246	2.077455	2.086368
4	-1.061157	-0.999542	-1.171667	2.451974	2.446732
...
1359	-0.390120	0.240684	-0.418239	-1.097229	-1.097491
1360	-0.442470	-0.411664	-0.403277	-0.585118	-0.576386
1361	0.338978	0.420982	0.195190	-0.809292	-0.809335
1362	-0.481495	-0.249943	-0.463124	-1.035480	-1.042957
1363	-1.063060	-0.761331	-0.893807	-1.156964	-1.158085

1364 rows x 93 columns

□ checking for data balance

```
# Create a count plot to visualize the distribution of the target variable
plt.figure(figsize=(8, 6))
```

```
sns.countplot(x='Bankrupt?', data=df_cleaned, palette=['darkblue', 'darkorange'])  
plt.title('Distribution of Target Variable')  
plt.show()
```



```
df_cleaned['Bankrupt?'].value_counts()
```

```
0    6599
1     220
Name: Bankrupt?, dtype: int64
```

```
class_counts = df_cleaned['Bankrupt?'].value_counts(normalize=True)
```

```
# Display the percentage of each class with '%' symbol
for label, percentage in class_counts.items():
    print(f"Class {label}: {percentage * 100:.2f}%")
```

```
Class 0: 96.77%
Class 1: 3.23%
```

Bankruptcy column is very imbalance. only 3.2% for class one. So let's check Anomaly case with this dataset using PCA

□ Anomaly Detection


```

# *Converting Dataset using StandardScaler:-*
X = df
Y = df['Bankrupt?'].map({0:1,1:-1})

from sklearn.preprocessing import StandardScaler
st = StandardScaler()

X_1 = st.fit_transform(X)
X_1

# *Train and Test Split of the Dataset:-*

from sklearn.model_selection import train_test_split
train_X,test_X,train_Y,test_Y = train_test_split(X_1,Y,test_size=0.25, random_state=0, stratify=Y)

train_X.shape, test_X.shape

train_Y.shape, test_Y.shape

test_Y.value_counts()

train_Y.value_counts()

# *Building Isolation Forest Model in UnSupervised Setting:-*

from sklearn.ensemble import IsolationForest

clf=IsolationForest(n_estimators=100, max_samples='auto', \
                    max_features=1.0, bootstrap=False, n_jobs=-1, random_state=0, verbose=0, contam
clf.fit(train_X)

pred_train = clf.predict(train_X)

np.unique(pred_train, return_counts=True)

from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score

target_names = ['bankrupt_company', 'normal_company']
print(classification_report(train_Y, pred_train, target_names=target_names))
print ("AUC: ", "{:.1%}".format(roc_auc_score(train_Y, pred_train)))
cm = confusion_matrix(train_Y, pred_train )

print(cm)

pred_test = clf.predict(test_X)

target_names2 = ['bankrupt_company', 'normal_company']
print(classification_report(test_Y, pred_test, target_names=target_names2))
print ("AUC: ", "{:.1%}".format(roc_auc_score(test_Y, pred_test)))
cm2 = confusion_matrix(test_Y, pred_test)

print(cm2)

```

	precision	recall	f1-score	support
bankrupt_company	0.30	0.31	0.31	165
normal_company	0.98	0.98	0.98	4949
accuracy			0.95	5114
macro avg	0.64	0.64	0.64	5114
weighted avg	0.96	0.95	0.95	5114

```

AUC:  64.3%
[[ 51 114]
 [118 4831]]

```

	precision	recall	f1-score	support
bankrupt_company	0.35	0.44	0.39	55
normal_company	0.98	0.97	0.98	1650
accuracy			0.96	1705
macro avg	0.66	0.70	0.68	1705

```

weighted avg      0.96      0.96      0.96      1705

AUC:  70.5%
[[ 24  31]
 [ 45 1605]]

```

Since the precision and recall value is low that mean that our model struggles with class 1. let's visualize the dataset using PCA to investigate the problem

```

df2 = df.iloc[:,1:].copy()

X = df2
Y = df['Bankrupt?']

bankrupt_index = df[df['Bankrupt?']==1].index
bankrupt_index

from sklearn.decomposition import PCA
pca = PCA(2)
pca.fit(X)

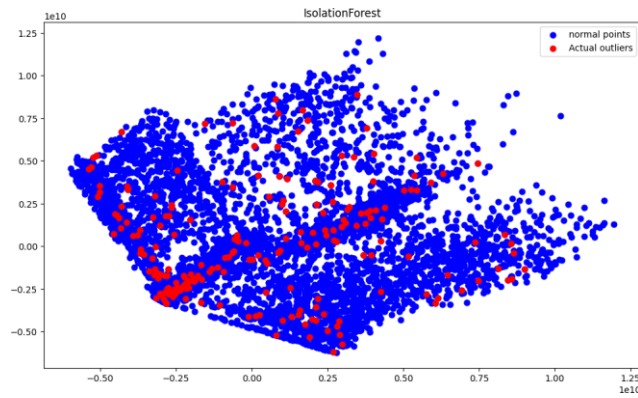
res=pd.DataFrame(pca.transform(X))

Z = np.array(res)
figsize=(12, 7)
plt.figure(figsize=figsize)
plt.title("IsolationForest")
plt.contourf( Z, cmap=plt.cm.Blues_r)

b1 = plt.scatter(res[0], res[1], c='blue',
                  s=40,label="normal points")

b1 = plt.scatter(res.iloc[bankrupt_index,0],res.iloc[bankrupt_index,1], c='red',
                  s=40, edgecolor="red",label="Actual outliers")
plt.legend(loc="upper right")
plt.show()

```



```

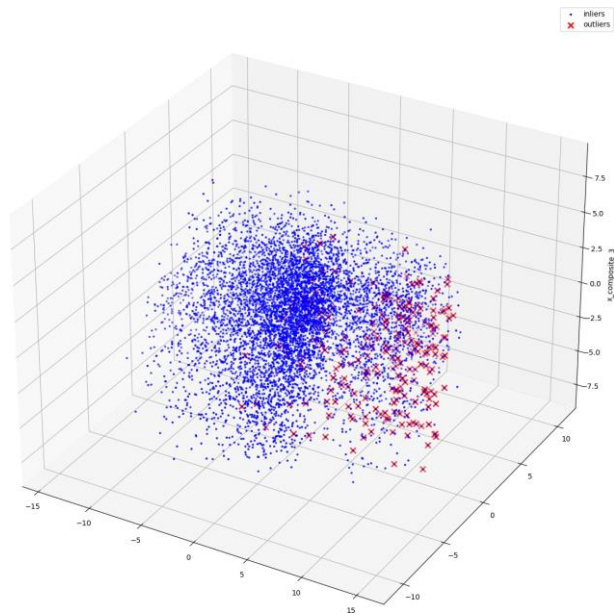
pca = PCA(n_components=3) # Reduce to k=3 dimensions
scaler = StandardScaler()
#normalize the metrics
X = scaler.fit_transform(X)
X_reduce = pca.fit_transform(X)

fig = plt.figure( figsize=(20,16))
ax = fig.add_subplot(111, projection='3d')
ax.set_zlabel("x_composite_3")

# Plot the compressed data points
ax.scatter(X_reduce[:, 0], X_reduce[:, 1], zs=X_reduce[:, 2], s=4, lw=1, label="inliers",c="blue")

# Plot x's for the ground truth outliers
ax.scatter(X_reduce[bankrupt_index,0],X_reduce[bankrupt_index,1], X_reduce[bankrupt_index,2],
          lw=2, s=60, marker="x", c="red", label="outliers")
ax.legend()
plt.show()

```



Classes are intertwined & overlapping. Thus, bankruptcy isn't an anomaly. Bankrupt instances, simply, happen to be sparse in the data set. So we will smote the data

□ checking for linearity

```
df.corr()['Bankrupt?']
```

Bankrupt?	1.000000
ROA(C) before interest and depreciation before interest	-0.239543
ROA(A) before interest and % after tax	-0.239500
ROA(B) before interest and depreciation after tax	-0.241993
Operating Gross Margin	-0.140042

```

Net Income to Stockholder's Equity      ...
Liability to Equity                      -0.251917
Degree of Financial Leverage (DFL)       0.246176
Interest Coverage Ratio (Interest expense to EBIT) -0.129292
to Liability                             -0.110761 Equity
Name: Bankrupt?, Length: 94, dtype: float64

correlations=df.corr()['Bankrupt?']
import matplotlib.pyplot as plt

# Plotting the correlations
correlations.plot(kind='bar', figsize=(20, 20), color='blue', alpha=0.7)
plt.title('Correlation with Bankruptcy')
plt.xlabel('Features')
plt.ylabel('Correlation Coefficient')
plt.show()

```



```

pca=PCA()
pca.fit(X_train_scaled)
exp_variance = pca.explained_variance_ratio_
cum_exp_variance = np.cumsum(exp_variance)
print(cum_exp_variance)

[0.25611217 0.39117652 0.48017277 0.52845127 0.57518833 0.61341973
 0.64527925 0.67584747 0.70396104 0.728935 0.74830102 0.76706242
 0.78262061 0.79682056 0.80962961 0.82059638 0.83124919 0.841297
 0.85108414 0.86071555 0.86951514 0.87794252 0.8862575 0.89372901
 0.90109899 0.90819919 0.91466356 0.92097425 0.9267167 0.9322052
 0.93717566 0.94191977 0.94633757 0.95064578 0.95480151 0.95828371
 0.961037 0.96358662 0.9659644 0.96810755 0.97015185 0.97208275
 0.97392075 0.97560808 0.97728318 0.97887444 0.98030773 0.98171679
 0.98307848 0.98438333 0.98557478 0.98670059 0.98778848 0.98883064
 0.98979599 0.99068851 0.99154461 0.99234583 0.99308448 0.99379436
 0.99443751 0.99503402 0.99559056 0.99608173 0.99653316 0.99695369
 0.99734963 0.99771466 0.99805122 0.99833875 0.99860789 0.99884162
 0.99903044 0.99920872 0.99934159 0.99946468 0.99956772 0.99966835
 0.99974217 0.99980513 0.99986547 0.99990647 0.99993624 0.9999652
 0.99997824 0.99998892 0.99999391 0.99999843 1. 1.
 1. 1. 1. ]

# Set your desired explained variance threshold (e.g., 95%)
desired_variance = 0.95

# Find the number of components that meet the threshold
num_components_threshold = np.argmax(cum_exp_variance >= desired_variance) + 1

# Print the cumulative explained variance
print("Cumulative Explained Variance:")
print(cum_exp_variance)

# Print the number of components needed to reach the desired variance
print(f'\nNumber of components for {desired_variance * 100}% explained variance: {num_components_thresh}

Cumulative Explained Variance:
[0.25611217 0.39117652 0.48017277 0.52845127 0.57518833 0.61341973
 0.64527925 0.67584747 0.70396104 0.728935 0.74830102 0.76706242
 0.78262061 0.79682056 0.80962961 0.82059638 0.83124919 0.841297
 0.85108414 0.86071555 0.86951514 0.87794252 0.8862575 0.89372901
 0.90109899 0.90819919 0.91466356 0.92097425 0.9267167 0.9322052
 0.93717566 0.94191977 0.94633757 0.95064578 0.95480151 0.95828371
 0.961037 0.96358662 0.9659644 0.96810755 0.97015185 0.97208275
 0.97392075 0.97560808 0.97728318 0.97887444 0.98030773 0.98171679
 0.98307848 0.98438333 0.98557478 0.98670059 0.98778848 0.98883064
 0.98979599 0.99068851 0.99154461 0.99234583 0.99308448 0.99379436
 0.99443751 0.99503402 0.99559056 0.99608173 0.99653316 0.99695369
 0.99734963 0.99771466 0.99805122 0.99833875 0.99860789 0.99884162
 0.99903044 0.99920872 0.99934159 0.99946468 0.99956772 0.99966835
 0.99974217 0.99980513 0.99986547 0.99990647 0.99993624 0.9999652
 0.99997824 0.99998892 0.99999391 0.99999843 1. 1.
 1. 1. 1. ]

Number of components for 95.0% explained variance: 34

pca = PCA(n_components=34,random_state=7)

X_train_pca = pca.fit_transform(X_train_scaled)

X_test_pca = pca.transform(X_test_scaled)

```

□ Random forest

```

rf_model = RandomForestClassifier(random_state = 0, n_estimators = 100)
rf_model.fit(X_train_pca, y_train)

```

```

▼ RandomForestClassifier
RandomForestClassifier(random_state=0)

```

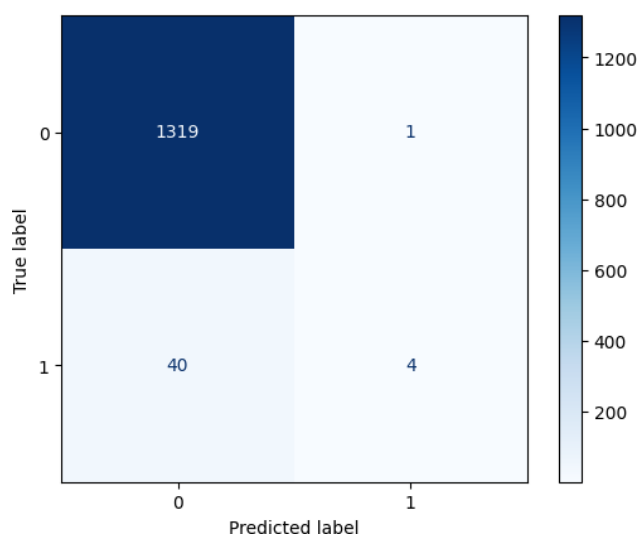
```

rf_y_pred = rf_model.predict(X_test_pca)

print(classification_report(y_test, rf_y_pred))
ConfusionMatrixDisplay.from_predictions(y_test, rf_y_pred, cmap='Blues')
plt.grid(False)

```

	precision	recall	f1-score	support
0	0.97	1.00	0.98	1320
1	0.80	0.09	0.16	44
accuracy			0.97	1364
macro avg	0.89	0.55	0.57	1364
weighted avg	0.97	0.97	0.96	1364



```

auc = round(metrics.roc_auc_score(y_test,rf_y_pred), 2)
f1 = round(metrics.f1_score(y_test, rf_y_pred, average='weighted'), 2)
precision = round(metrics.precision_score(y_test, rf_y_pred), 2)
recall = round(metrics.recall_score(y_test,rf_y_pred), 2)
accuracy = round(metrics.accuracy_score(y_test,rf_y_pred), 2)
print("auc",auc)
print("f1",f1)
print("precision",precision)
print("recall",recall)

auc 0.55
f1 0.96
precision 0.8
recall 0.09

```

□ logistic regression

```

from sklearn.linear_model import LogisticRegression
# Create a logistic regression model
logreg_model = LogisticRegression(max_iter=1000,random_state=42)

# Train the model
logreg_model.fit(X_train_pca , y_train)

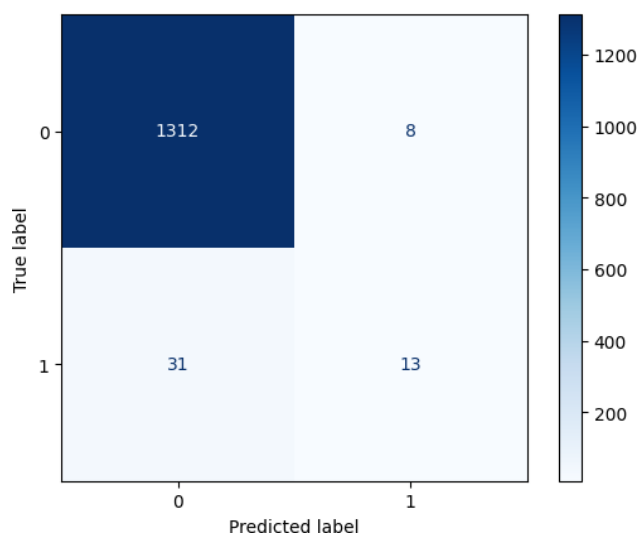
# Make predictions
log_y_pred = logreg_model.predict(X_test_pca)

```



```
print(classification_report(y_test, log_y_pred))
ConfusionMatrixDisplay.from_predictions(y_test, log_y_pred, cmap='Blues')
plt.grid(False)
```

	precision	recall	f1-score	support
0	0.98	0.99	0.99	1320
1	0.62	0.30	0.40	44
accuracy			0.97	1364
macro avg	0.80	0.64	0.69	1364
weighted avg	0.97	0.97	0.97	1364



```
auc = round(metrics.roc_auc_score(y_test,log_y_pred), 2)
f1 = round(metrics.f1_score(y_test, log_y_pred, average='weighted'), 2)
precision = round(metrics.precision_score(y_test, log_y_pred), 2)
recall = round(metrics.recall_score(y_test,log_y_pred), 2)
accuracy = round(metrics.accuracy_score(y_test,log_y_pred), 2)
print("auc",auc)
print("f1",f1)
print("precision",precision)
print("recall",recall)

auc 0.64
f1 0.97
precision 0.62
recall 0.3
```

□ Adaboost

```
# Create a weak learner (Decision Tree in this case)
base_classifier = DecisionTreeClassifier(max_depth=1)

# Create an AdaBoost classifier with the weak learner
adaboost_classifier = AdaBoostClassifier(base_classifier, n_estimators=50, random_state=42)

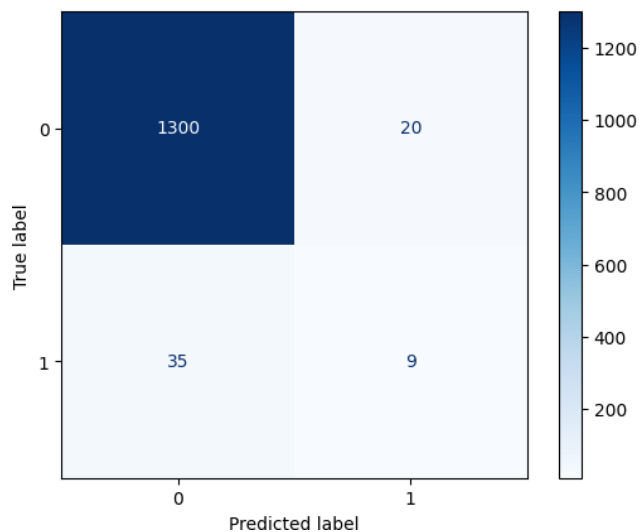
# Train the AdaBoost classifier on the training data
adaboost_classifier.fit(X_train_pca, y_train)
```

```
> AdaBoostClassifier
> estimator: DecisionTreeClassifier
  > DecisionTreeClassifier
```

```
# Make predictions on the test set
Adaboost_y_pred = adaboost_classifier.predict(X_test_pca)

print(classification_report(y_test, Adaboost_y_pred))
ConfusionMatrixDisplay.from_predictions(y_test, Adaboost_y_pred, cmap='Blues')
plt.grid(False)
```

	precision	recall	f1-score	support
0	0.97	0.98	0.98	1320
1	0.31	0.20	0.25	44
accuracy			0.96	1364
macro avg	0.64	0.59	0.61	1364
weighted avg	0.95	0.96	0.96	1364



```
auc = round(metrics.roc_auc_score(y_test,Adaboost_y_pred), 2)
f1 = round(metrics.f1_score(y_test, Adaboost_y_pred, average='weighted'), 2)
precision = round(metrics.precision_score(y_test,Adaboost_y_pred), 2)
recall = round(metrics.recall_score(y_test,Adaboost_y_pred), 2)
accuracy = round(metrics.accuracy_score(y_test,Adaboost_y_pred), 2)
print("auc",auc)
print("f1",f1)
print("precision",precision)
print("recall",recall)

auc 0.59
f1 0.96
precision 0.31
recall 0.2
```

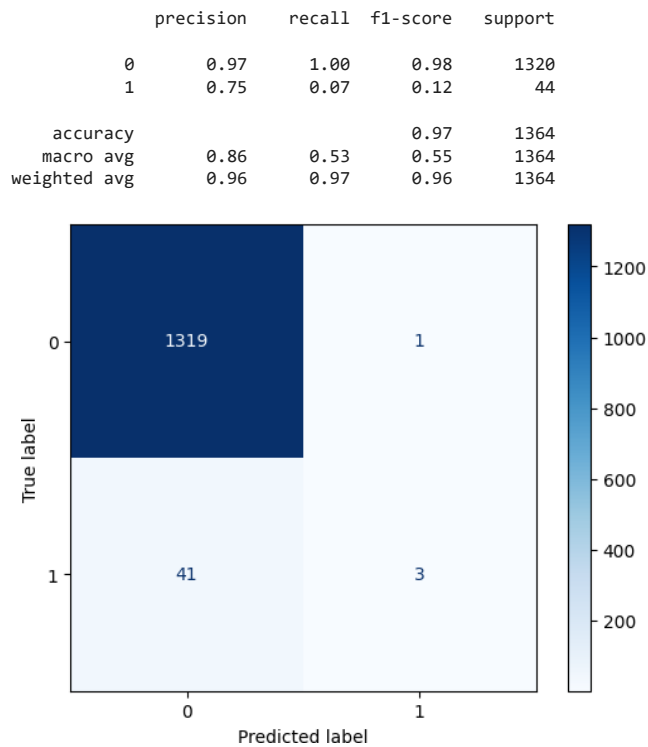
□ SVM

```
# Create an SVM classifier
svm_classifier = SVC(kernel='rbf', C=1.0, random_state=42)

# Train the SVM classifier on the training data
svm_classifier.fit(X_train_pca, y_train)

# Make predictions on the test set
SVC_y_pred = svm_classifier.predict(X_test_pca)
```

```
print(classification_report(y_test, SVC_y_pred))
ConfusionMatrixDisplay.from_predictions(y_test, SVC_y_pred, cmap='Blues')
plt.grid(False)
```



```
auc = round(metrics.roc_auc_score(y_test,SVC_y_pred), 2)
f1 = round(metrics.f1_score(y_test, SVC_y_pred, average='weighted'), 2)
precision = round(metrics.precision_score(y_test,SVC_y_pred), 2)
recall = round(metrics.recall_score(y_test,SVC_y_pred), 2)
accuracy = round(metrics.accuracy_score(y_test,SVC_y_pred), 2)
print("auc",auc)
print("f1",f1)
print("precision",precision)
print("recall",recall)

auc 0.53
f1 0.96
precision 0.75
recall 0.07
```

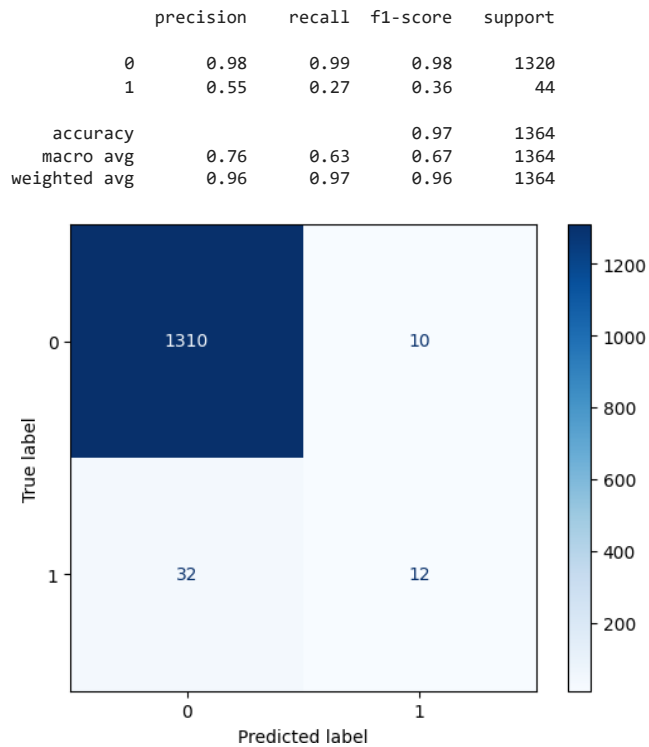
□ KNN

```
# Create a KNN classifier with k=3 (you can adjust the value of k)
knn_classifier = KNeighborsClassifier(n_neighbors=3)

# Train the KNN classifier on the training data
knn_classifier.fit(X_train_pca, y_train)

# Make predictions on the test set
knn_y_pred = knn_classifier.predict(X_test_pca)

print(classification_report(y_test, knn_y_pred))
ConfusionMatrixDisplay.from_predictions(y_test, knn_y_pred, cmap='Blues')
plt.grid(False)
```



```

auc = round(metrics.roc_auc_score(y_test,knn_y_pred), 2)
f1 = round(metrics.f1_score(y_test, knn_y_pred, average='weighted'), 2)
precision = round(metrics.precision_score(y_test,knn_y_pred), 2)
recall = round(metrics.recall_score(y_test,knn_y_pred), 2)
accuracy = round(metrics.accuracy_score(y_test,knn_y_pred), 2)
print("auc",auc)
print("f1",f1)
print("precision",precision)
print("recall",recall)

auc 0.63
f1 0.96
precision 0.55
recall 0.27

```

□ Decsion Tree

```

# Initialize the Decision Tree classifier
decision_tree_classifier = DecisionTreeClassifier(random_state=42)

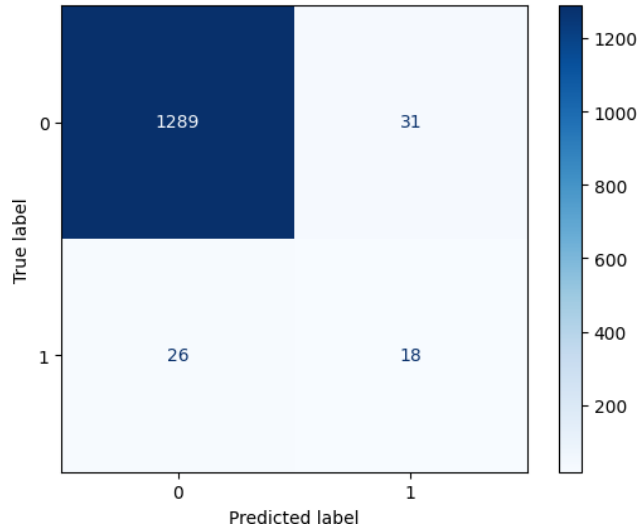
# Train the model on the training set
decision_tree_classifier.fit(X_train_pca, y_train)

# Make predictions on the testing set
DT_y_pred = decision_tree_classifier.predict(X_test_pca)

print(classification_report(y_test, DT_y_pred))
ConfusionMatrixDisplay.from_predictions(y_test, DT_y_pred, cmap='Blues')
plt.grid(False)

```

	precision	recall	f1-score	support
0	0.98	0.98	0.98	1320
1	0.37	0.41	0.39	44
accuracy			0.96	1364
macro avg	0.67	0.69	0.68	1364
weighted avg	0.96	0.96	0.96	1364



```

auc = round(metrics.roc_auc_score(y_test,DT_y_pred), 2)
f1 = round(metrics.f1_score(y_test, DT_y_pred, average='weighted'), 2)
precision = round(metrics.precision_score(y_test,DT_y_pred), 2)
recall = round(metrics.recall_score(y_test,DT_y_pred), 2)
accuracy = round(metrics.accuracy_score(y_test,DT_y_pred), 2)
print("auc",auc)
print("f1",f1)
print("precision",precision)
print("recall",recall)

auc 0.69
f1 0.96
precision 0.37
recall 0.41

```

□ Naive Bayes

```

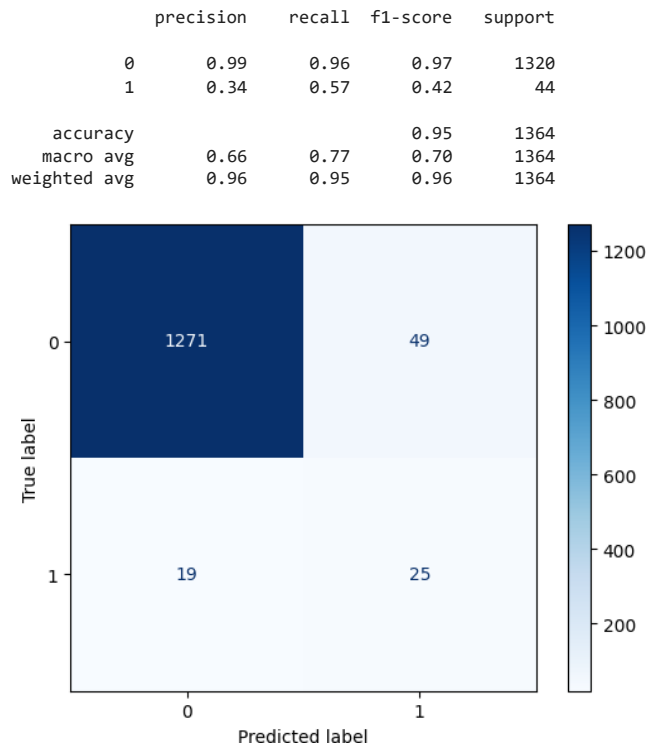
# Initialize the Gaussian Naive Bayes classifier
naive_bayes_classifier = GaussianNB()

# Train the model on the training set
naive_bayes_classifier.fit(X_train_pca, y_train)

# Make predictions on the testing set
Naive_y_pred = naive_bayes_classifier.predict(X_test_pca)

print(classification_report(y_test, Naive_y_pred))
ConfusionMatrixDisplay.from_predictions(y_test, Naive_y_pred, cmap='Blues')
plt.grid(False)

```



```

auc = round(metrics.roc_auc_score(y_test,Naive_y_pred), 2)
f1 = round(metrics.f1_score(y_test, Naive_y_pred, average='weighted'), 2)
precision = round(metrics.precision_score(y_test,Naive_y_pred), 2)
recall = round(metrics.recall_score(y_test,Naive_y_pred), 2)
accuracy = round(metrics.accuracy_score(y_test,Naive_y_pred), 2)
print("auc",auc)
print("f1",f1)
print("precision",precision)
print("recall",recall)

auc 0.77
f1 0.96
precision 0.34
recall 0.57

```

□ xgboost after hyperparameter tuning

```

pca = PCA(n_components=34,random_state=7)

X_train_pca_2 = pca.fit_transform(X_train_scaled)

X_test_pca_2 = pca.transform(X_test_scaled)

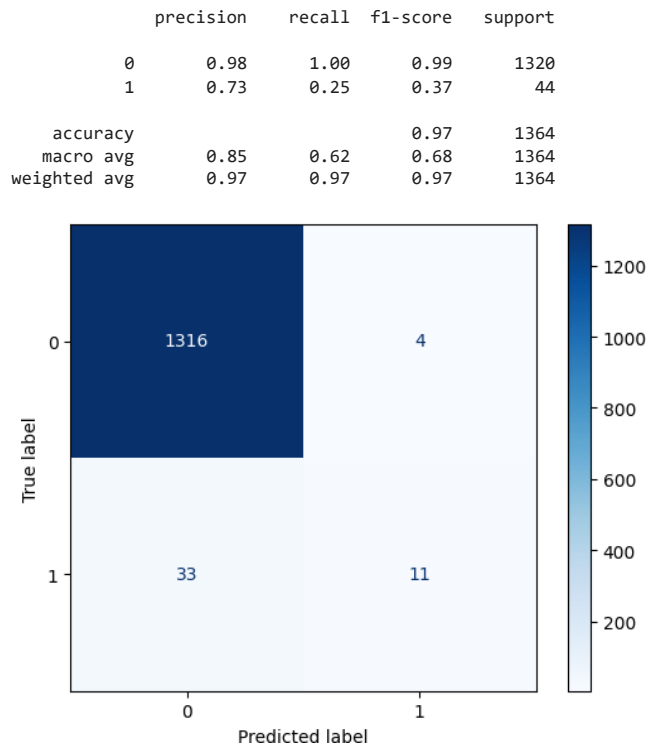
# Initialize the XGBoost classifier
xgb_classifier = xgb.XGBClassifier(learning_rate= 0.049049280704634754, max_depth= 5,n_estimators=188,s

# Train the model on the training set
xgb_classifier.fit(X_train_pca_2, y_train)

# Make predictions on the testing set
XG_y_pred = xgb_classifier.predict(X_test_pca_2)

print(classification_report(y_test, XG_y_pred))
ConfusionMatrixDisplay.from_predictions(y_test, XG_y_pred, cmap='Blues')
plt.grid(False)

```



```

auc = round(metrics.roc_auc_score(y_test,XG_y_pred), 2)
f1 = round(metrics.f1_score(y_test,XG_y_pred, average='weighted'), 2)
precision = round(metrics.precision_score(y_test,XG_y_pred), 2)
recall = round(metrics.recall_score(y_test,XG_y_pred), 2)
accuracy = round(metrics.accuracy_score(y_test,XG_y_pred), 2)
print("auc",auc)
print("f1",f1)
print("precision",precision)
print("recall",recall)

auc 0.62
f1 0.97
precision 0.73
recall 0.25

```

□ XGBoost

```

# Initialize the XGBoost classifier
xgb_classifier = xgb.XGBClassifier(random_state=42)

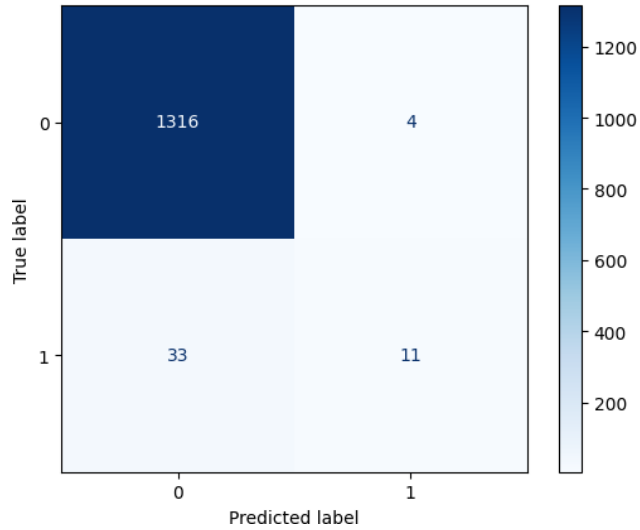
# Train the model on the training set
xgb_classifier.fit(X_train_pca, y_train)

# Make predictions on the testing set
XG_y_pred = xgb_classifier.predict(X_test_pca)

print(classification_report(y_test, XG_y_pred))
ConfusionMatrixDisplay.from_predictions(y_test, XG_y_pred, cmap='Blues')
plt.grid(False)

```

	precision	recall	f1-score	support
0	0.98	1.00	0.99	1320
1	0.73	0.25	0.37	44
accuracy			0.97	1364
macro avg	0.85	0.62	0.68	1364
weighted avg	0.97	0.97	0.97	1364



```

auc = round(metrics.roc_auc_score(y_test,XG_y_pred), 2)
f1 = round(metrics.f1_score(y_test,XG_y_pred, average='weighted'), 2)
precision = round(metrics.precision_score(y_test,XG_y_pred), 2)
recall = round(metrics.recall_score(y_test,XG_y_pred), 2)
accuracy = round(metrics.accuracy_score(y_test,XG_y_pred), 2)
print("auc",auc)
print("f1",f1)
print("precision",precision)
print("recall",recall)

auc 0.62
f1 0.97
precision 0.73
recall 0.25

```

□ Comparison


```

from tabulate import tabulate
from sklearn import metrics

models = [rf_model, logreg_model, adaboost_classifier, svm_classifier, knn_classifier, decision_tree_classifier, naive_bayes_classifier]
model_names = ['random forest', 'Logistic Regression', 'AdaBoost', 'SVM', 'KNN', 'Decision Tree', 'Naive Bayes']

results = []

for model, model_name in zip(models, model_names):
    auc = metrics.roc_auc_score(y_test, model.predict(X_test_pca))
    f1 = metrics.f1_score(y_test, model.predict(X_test_pca), average='weighted')
    precision = metrics.precision_score(y_test, model.predict(X_test_pca))
    recall = metrics.recall_score(y_test, model.predict(X_test_pca))
    accuracy = metrics.accuracy_score(y_test, model.predict(X_test_pca))

    results.append([model_name, round(auc, 2), round(f1, 2), round(precision, 2), round(recall, 2), round(accuracy, 2)])

# Display the results in a table
headers = ['Model', 'AUC', 'F1', 'Precision', 'Recall', 'Accuracy']
table = tabulate(results, headers, tablefmt='grid')

print(table)

```

Model	AUC	F1	Precision	Recall	Accuracy
random forest	0.55	0.96	0.8	0.09	0.97
Logistic Regression	0.65	0.97	0.72	0.3	0.97
AdaBoost	0.59	0.96	0.31	0.2	0.96
SVM	0.53	0.96	0.75	0.07	0.97
KNN	0.63	0.96	0.55	0.27	0.97
Decision Tree	0.69	0.96	0.37	0.41	0.96
Naive Bayes	0.77	0.96	0.34	0.57	0.95
XGBoost	0.62	0.97	0.73	0.25	0.97

```

import xgboost as xgb
from sklearn.model_selection import RandomizedSearchCV
import scipy.stats as stats

# Define the hyperparameter distributions
param_dist = {
    'max_depth': stats.randint(3, 10),
    'learning_rate': stats.uniform(0.01, 0.1),
    'subsample': stats.uniform(0.5, 0.5),
    'n_estimators': stats.randint(50, 200)
}

# Create the XGBoost model object
xgb_model = xgb.XGBClassifier()

# Create the RandomizedSearchCV object
random_search = RandomizedSearchCV(xgb_model, param_distributions=param_dist, n_iter=10, cv=5, scoring='roc_auc')

# Fit the RandomizedSearchCV object to the training data
random_search.fit(X_train_pca, y_train)

# Print the best set of hyperparameters and the corresponding score
print("Best set of hyperparameters: ", random_search.best_params_)
print("Best score: ", random_search.best_score_)

Best set of hyperparameters: {'learning_rate': 0.09333313240873221, 'max_depth': 7, 'n_estimators': 159}
Best score: 0.15904761904761905

```

□ Smote

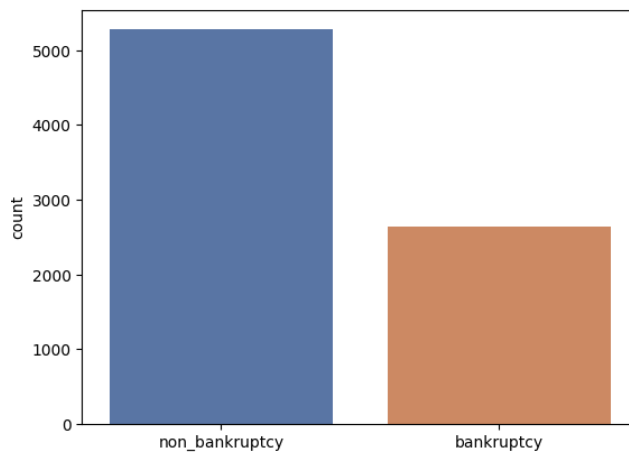
```
smote = SMOTE(sampling_strategy = 0.5, random_state=42)

X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

ax = sns.countplot(x = y_train_smote, palette='deep');
ax.set_xticks(ticks=[0,1]);
ax.set_xticklabels(labels=['non_bankruptcy', 'bankruptcy']);

from collections import Counter
print('Original dataset shape %s' % Counter(y_train), '- Total Rows:', len(y_train))
print('Resampled dataset shape %s' % Counter(y_train_smote), '- Total Rows:', len(y_train_smote))
```

Original dataset shape Counter({0: 5279, 1: 176}) - Total Rows:
Resampled dataset shape Counter({0: 5279, 1: 2639}) - Total Row



□ Scaling

```
s_scaler = StandardScaler()
s_scaler.fit(X_train_smote)
X_train_scaled = s_scaler.transform(X_train_smote)
X_test_scaled = s_scaler.transform(X_test)

X_train_scaled = pd.DataFrame(X_train_scaled, columns=X_train.columns)
X_train_scaled
```

	ROA(C)	ROA(A)	ROA(B)	Operating	Realized
	before interest and depreciation	before interest and % tax after	before interest and depreciation after tax	Gross Margin	Sales Gross Margin
0	0.171416	0.194222	0.203106	-0.623088	-0.624828
1	0.899790	0.944107	0.729786	-0.550474	-0.564096
2	0.756388	0.872641	0.603500	0.143403	0.144447
3	-0.260537	-0.051378	-0.227635	-0.916910	-0.919717
4	-1.662199	-1.560459	-1.672822	2.697703	2.692670
...
7913	-0.760347	-0.335709	-0.559281	-0.607215	-0.608897
7914	-0.261643	-0.140970	-0.236853	-0.528221	-0.529616
7915	-1.479501	-1.525698	-1.534615	-0.988753	-0.934158
7916	-0.799788	-0.629822	-0.845231	1.040667	1.044969
7917	-1.726467	-1.560459	-1.672822	-0.363601	-0.364398

7918 rows x 93 columns

```
X_test_scaled = pd.DataFrame(X_test_scaled, columns=X_test.columns)
X_test_scaled
```

	ROA(C)	ROA(A)	ROA(B)	Operating	Realized
	before interest and depreciation	before interest and % tax after	before interest and depreciation after tax	Gross Margin	Sales Gross Margin
0	0.341049	0.304943	0.343097	0.391504	0.393449
1	0.031512	0.149933	-0.052402	-0.596194	-0.597836
2	1.258293	1.456444	1.390583	0.247619	0.251740
3	2.509121	2.399336	2.483834	2.322525	2.331482
4	-0.602427	-0.547611	-0.696556	2.697703	2.692670
...
1359	0.014024	0.594831	-0.006391	-0.857742	-0.859659
1360	-0.034068	-0.006083	0.007315	-0.344731	-0.337362
1361	0.683813	0.760913	0.555531	-0.569300	-0.570844
1362	-0.069918	0.142887	-0.047507	-0.795885	-0.805000
1363	-0.604176	-0.328181	-0.442027	-0.917583	-0.920392

1364 rows x 93 columns

□ PCA

```
pca = PCA(n_components=34, random_state=7)
```

```
X_train_pca_smote = pca.fit_transform(X_train_scaled)
X_test_pca_smote = pca.transform(X_test_scaled)
```

□ models

□ Random Forest

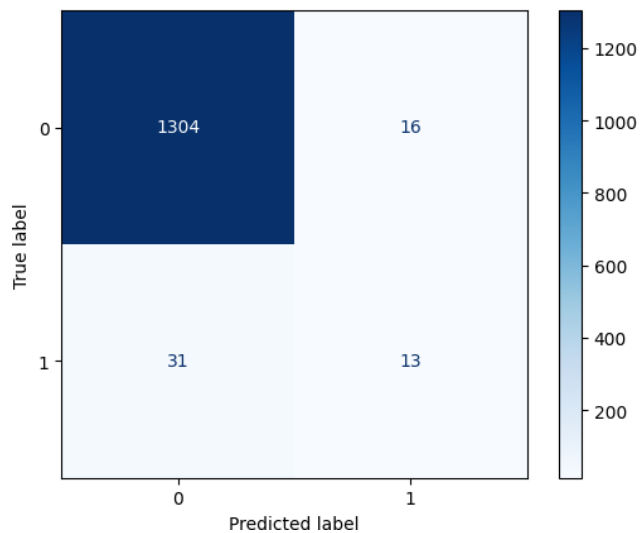
```
rf_smote_model = RandomForestClassifier(random_state = 0, n_estimators = 100)
rf_smote_model.fit(X_train_pca_smote, y_train_smote)
```

```
▼ RandomForestClassifier
RandomForestClassifier(random_state=0)
```

```
rf_smote_y_pred = rf_smote_model.predict(X_test_pca_smote)
```

```
print(classification_report(y_test, rf_smote_y_pred))
ConfusionMatrixDisplay.from_predictions(y_test, rf_smote_y_pred, cmap='Blues')
plt.grid(False)
```

	precision	recall	f1-score	support
0	0.98	0.99	0.98	1320
1	0.45	0.30	0.36	44
accuracy			0.97	1364
macro avg	0.71	0.64	0.67	1364
weighted avg	0.96	0.97	0.96	1364



```
auc = round(metrics.roc_auc_score(y_test, rf_smote_y_pred), 2)
f1 = round(metrics.f1_score(y_test, rf_smote_y_pred, average='weighted'), 2)
precision = round(metrics.precision_score(y_test, rf_smote_y_pred), 2)
recall = round(metrics.recall_score(y_test, rf_smote_y_pred), 2)
accuracy = round(metrics.accuracy_score(y_test, rf_smote_y_pred), 2)
print("auc", auc)
print("f1", f1)
print("precision", precision)
print("recall", recall)
```

```
auc 0.64
f1 0.96
precision 0.45
recall 0.3
```

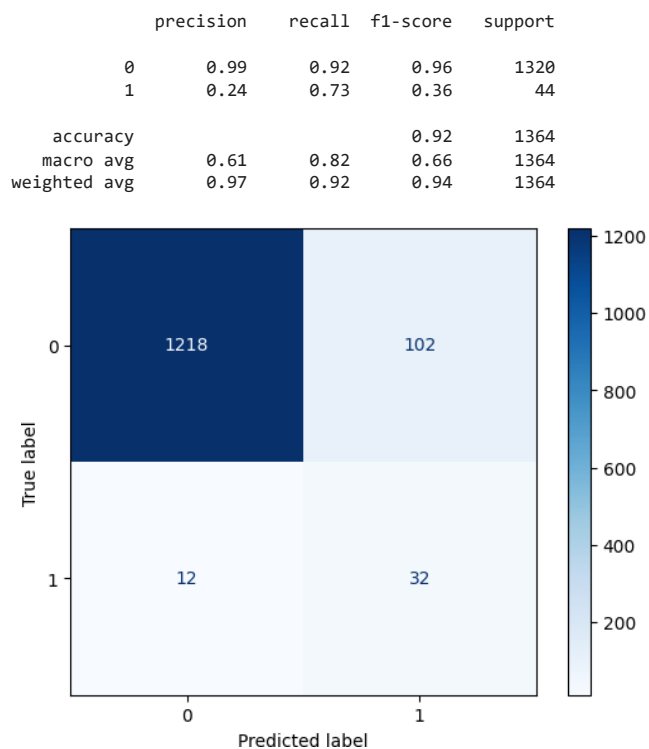
□ logistic regression

```
from sklearn.linear_model import LogisticRegression
# Create a logistic regression model
logreg_model_smote = LogisticRegression(max_iter=1000, random_state=42)

# Train the model
logreg_model_smote.fit(X_train_pca_smote, y_train_smote)

# Make predictions
log_smote_y_pred = logreg_model_smote.predict(X_test_pca_smote)

print(classification_report(y_test, log_smote_y_pred))
ConfusionMatrixDisplay.from_predictions(y_test, log_smote_y_pred, cmap='Blues')
plt.grid(False)
```



```
auc = round(metrics.roc_auc_score(y_test, log_smote_y_pred), 2)
f1 = round(metrics.f1_score(y_test, log_smote_y_pred, average='weighted'), 2)
precision = round(metrics.precision_score(y_test, log_smote_y_pred), 2)
recall = round(metrics.recall_score(y_test, log_smote_y_pred), 2)
accuracy = round(metrics.accuracy_score(y_test, log_smote_y_pred), 2)
print("auc", auc)
print("f1", f1)
print("precision", precision)
print("recall", recall)
print("accuracy", accuracy)

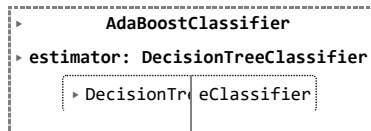
auc 0.82
f1 0.94
precision 0.24
recall 0.73
accuracy 0.92
```

□ Adaboost

```
# Create a weak learner (Decision Tree in this case)
base_classifier_smote = DecisionTreeClassifier(max_depth=1)

# Create an AdaBoost classifier with the weak learner
adaboost_classifier_smote= AdaBoostClassifier(base_classifier_smote, n_estimators=50, random_state=42)

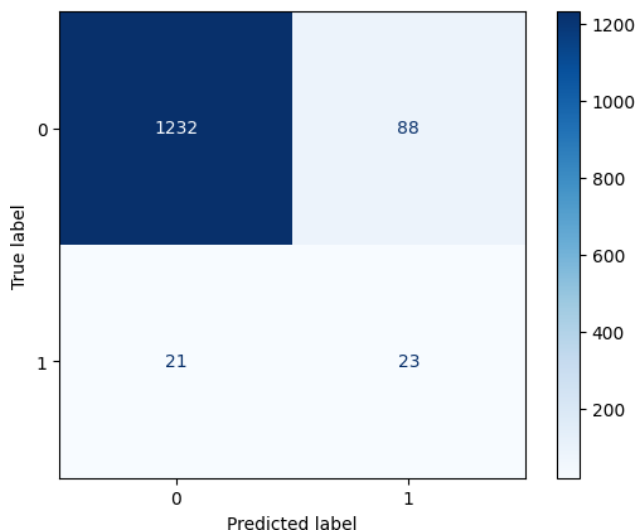
# Train the AdaBoost classifier on the training data
adaboost_classifier_smote.fit(X_train_pca_smote, y_train_smote)
```



```
Adaboost_y_pred_smote = adaboost_classifier_smote.predict(X_test_pca_smote)
```

```
print(classification_report(y_test, Adaboost_y_pred_smote))
ConfusionMatrixDisplay.from_predictions(y_test, Adaboost_y_pred_smote, cmap='Blues')
plt.grid(False)
```

	precision	recall	f1-score	support
0	0.98	0.93	0.96	1320
1	0.21	0.52	0.30	44
accuracy			0.92	1364
macro avg	0.60	0.73	0.63	1364
weighted avg	0.96	0.92	0.94	1364



```
auc = round(metrics.roc_auc_score(y_test,Adaboost_y_pred_smote), 2)
f1 = round(metrics.f1_score(y_test, Adaboost_y_pred_smote, average='weighted'), 2)
precision = round(metrics.precision_score(y_test,Adaboost_y_pred_smote), 2)
recall = round(metrics.recall_score(y_test,Adaboost_y_pred_smote), 2)
accuracy = round(metrics.accuracy_score(y_test,Adaboost_y_pred_smote), 2)
print("auc",auc)
print("f1",f1)
print("precision",precision)
print("recall",recall)

auc 0.73
f1 0.94
precision 0.21
recall 0.52
```

□ SVM

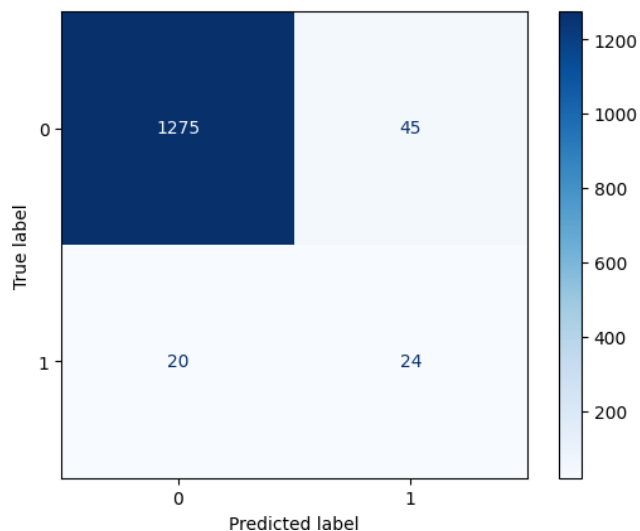
```
# Create an SVM classifier
svm_classifier_smote= SVC(kernel='rbf', C=1.0, random_state=42)

# Train the SVM classifier on the training data
svm_classifier_smote.fit(X_train_pca_smote, y_train_smote)

# Make predictions on the test set
SVC_y_pred_smote = svm_classifier_smote.predict(X_test_pca_smote)

print(classification_report(y_test, SVC_y_pred_smote))
ConfusionMatrixDisplay.from_predictions(y_test, SVC_y_pred_smote, cmap='Blues')
plt.grid(False)
```

	precision	recall	f1-score	support
0	0.98	0.97	0.98	1320
1	0.35	0.55	0.42	44
accuracy			0.95	1364
macro avg	0.67	0.76	0.70	1364
weighted avg	0.96	0.95	0.96	1364



```
auc = round(metrics.roc_auc_score(y_test,SVC_y_pred_smote), 2)
f1 = round(metrics.f1_score(y_test, SVC_y_pred_smote, average='weighted'), 2)
precision = round(metrics.precision_score(y_test,SVC_y_pred_smote), 2)
recall = round(metrics.recall_score(y_test,SVC_y_pred_smote), 2)
accuracy = round(metrics.accuracy_score(y_test,SVC_y_pred_smote), 2)
print("auc",auc)
print("f1",f1)
print("precision",precision)
print("recall",recall)

auc 0.76
f1 0.96
precision 0.35
recall 0.55
```

□ KNN

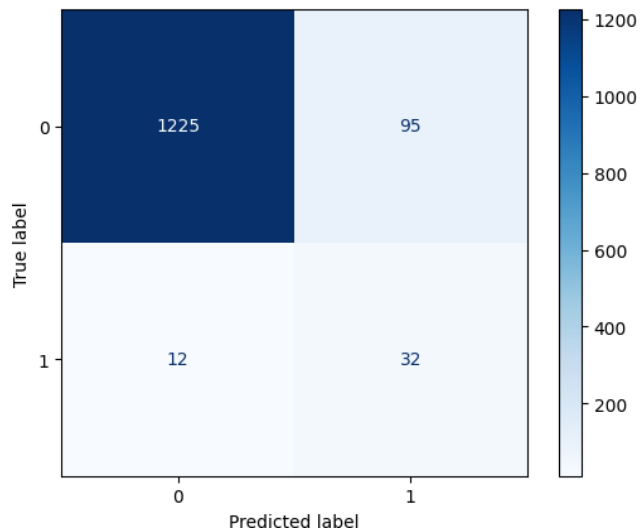

```
# Create a KNN classifier with k=3 (you can adjust the value of k)
knn_classifier_smote = KNeighborsClassifier(n_neighbors=3)

# Train the KNN classifier on the training data
knn_classifier_smote.fit(X_train_pca_smote, y_train_smote)

# Make predictions on the test set
knn_y_pred_smote = knn_classifier_smote.predict(X_test_pca_smote)

print(classification_report(y_test, knn_y_pred_smote))
ConfusionMatrixDisplay.from_predictions(y_test, knn_y_pred_smote, cmap='Blues')
plt.grid(False)
```

	precision	recall	f1-score	support
0	0.99	0.93	0.96	1320
1	0.25	0.73	0.37	44
accuracy			0.92	1364
macro avg	0.62	0.83	0.67	1364
weighted avg	0.97	0.92	0.94	1364



```
auc = round(metrics.roc_auc_score(y_test,knn_y_pred_smote), 2)
f1 = round(metrics.f1_score(y_test, knn_y_pred_smote, average='weighted'), 2)
precision = round(metrics.precision_score(y_test,knn_y_pred_smote), 2)
recall = round(metrics.recall_score(y_test,knn_y_pred_smote), 2)
accuracy = round(metrics.accuracy_score(y_test,knn_y_pred_smote), 2)
print("auc",auc)
print("f1",f1)
print("precision",precision)
print("recall",recall)

auc 0.83
f1 0.94
precision 0.25
recall 0.73
```

□ Decision Tree

```

# Initialize the Decision Tree classifier
decision_tree_classifier_smote = DecisionTreeClassifier(random_state=42)

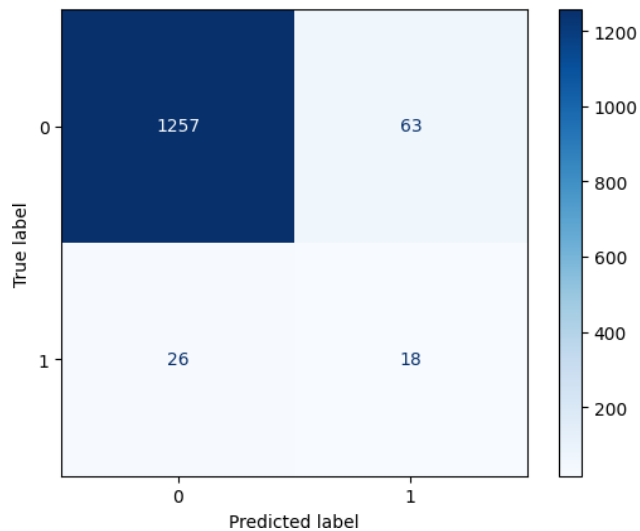
# Train the model on the training set
decision_tree_classifier_smote.fit(X_train_pca_smote, y_train_smote)

# Make predictions on the testing set
DT_y_pred_smote = decision_tree_classifier_smote.predict(X_test_pca_smote)

print(classification_report(y_test, DT_y_pred_smote))
ConfusionMatrixDisplay.from_predictions(y_test, DT_y_pred_smote, cmap='Blues')
plt.grid(False)

```

	precision	recall	f1-score	support
0	0.98	0.95	0.97	1320
1	0.22	0.41	0.29	44
accuracy			0.93	1364
macro avg	0.60	0.68	0.63	1364
weighted avg	0.96	0.93	0.94	1364



```

auc = round(metrics.roc_auc_score(y_test,DT_y_pred_smote), 2)
f1 = round(metrics.f1_score(y_test, DT_y_pred_smote, average='weighted'), 2)
precision = round(metrics.precision_score(y_test,DT_y_pred_smote), 2)
recall = round(metrics.recall_score(y_test,DT_y_pred_smote), 2)
accuracy = round(metrics.accuracy_score(y_test,DT_y_pred_smote), 2)
print("auc",auc)
print("f1",f1)
print("precision",precision)
print("recall",recall)

auc 0.68
f1 0.94
precision 0.22
recall 0.41

```

□ Naive Bayes

```

# Initialize the Gaussian Naive Bayes classifier
naive_bayes_classifier_smote = GaussianNB()

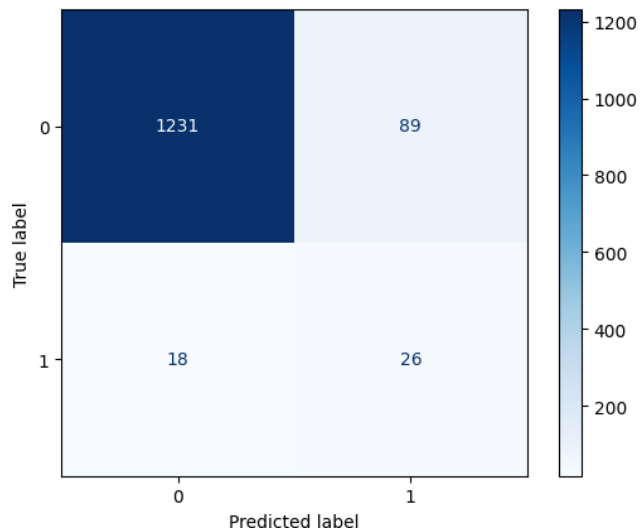
# Train the model on the training set
naive_bayes_classifier_smote.fit(X_train_pca_smote, y_train_smote)

# Make predictions on the testing set
Naive_y_pred_smote = naive_bayes_classifier_smote.predict(X_test_pca_smote)

print(classification_report(y_test, Naive_y_pred_smote))
ConfusionMatrixDisplay.from_predictions(y_test, Naive_y_pred_smote, cmap='Blues')
plt.grid(False)

```

	precision	recall	f1-score	support
0	0.99	0.93	0.96	1320
1	0.23	0.59	0.33	44
accuracy			0.92	1364
macro avg	0.61	0.76	0.64	1364
weighted avg	0.96	0.92	0.94	1364



```

auc = round(metrics.roc_auc_score(y_test,Naive_y_pred_smote), 2)
f1 = round(metrics.f1_score(y_test, Naive_y_pred_smote, average='weighted'), 2)
precision = round(metrics.precision_score(y_test,Naive_y_pred_smote), 2)
recall = round(metrics.recall_score(y_test,Naive_y_pred_smote), 2)
accuracy = round(metrics.accuracy_score(y_test,Naive_y_pred_smote), 2)
print("auc",auc)
print("f1",f1)
print("precision",precision)
print("recall",recall)

auc 0.76
f1 0.94
precision 0.23
recall 0.59

```

□ XGboost

```

# Initialize the XGBoost classifier
xgb_classifier_smote = xgb.XGBClassifier(learning_rate= 0.049049280704634754, max_depth= 5,n_estimators

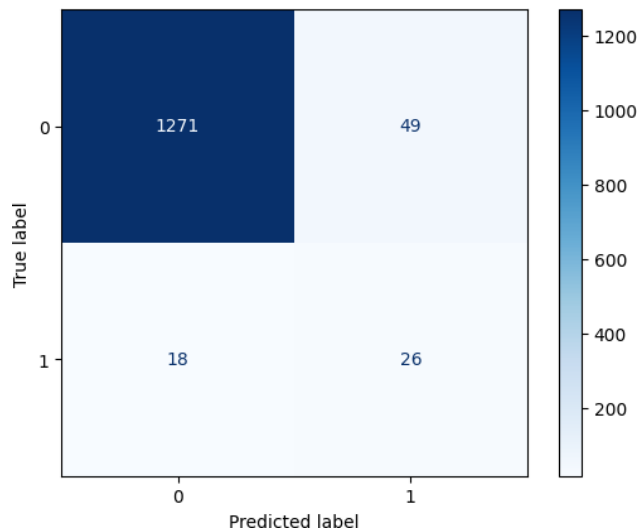
# Train the model on the training set
xgb_classifier_smote.fit(X_train_pca_smote, y_train_smote)

# Make predictions on the testing set
XG_y_pred_smote = xgb_classifier_smote.predict(X_test_pca_smote)

print(classification_report(y_test, XG_y_pred_smote))
ConfusionMatrixDisplay.from_predictions(y_test, XG_y_pred_smote, cmap='Blues')
plt.grid(False)

```

	precision	recall	f1-score	support
0	0.99	0.96	0.97	1320
1	0.35	0.59	0.44	44
accuracy			0.95	1364
macro avg	0.67	0.78	0.71	1364
weighted avg	0.97	0.95	0.96	1364



```

auc = round(metrics.roc_auc_score(y_test,XG_y_pred_smote), 2)
f1 = round(metrics.f1_score(y_test,XG_y_pred_smote, average='weighted'), 2)
precision = round(metrics.precision_score(y_test,XG_y_pred_smote), 2)
recall = round(metrics.recall_score(y_test,XG_y_pred_smote), 2)
accuracy = round(metrics.accuracy_score(y_test,XG_y_pred_smote), 2)
print("auc",auc)
print("f1",f1)
print("precision",precision)
print("recall",recall)

auc 0.78
f1 0.96
precision 0.35
recall 0.59

```

□ comparison

```

from tabulate import tabulate
from sklearn import metrics

models = [rf_smote_model, logreg_model_smote, adaboost_classifier_smote, svm_classifier_smote, knn_class]
model_names = ['random forest', 'Logistic Regression', 'AdaBoost', 'SVM', 'KNN', 'Decision Tree', 'Naive']

results = []

for model, model_name in zip(models, model_names):
    auc = metrics.roc_auc_score(y_test, model.predict(X_test_pca_smote))
    f1 = metrics.f1_score(y_test, model.predict(X_test_pca_smote), average='weighted')
    precision = metrics.precision_score(y_test, model.predict(X_test_pca_smote))
    recall = metrics.recall_score(y_test, model.predict(X_test_pca_smote))
    accuracy = metrics.accuracy_score(y_test, model.predict(X_test_pca_smote))

    results.append([model_name, round(auc, 2), round(f1, 2), round(precision, 2), round(recall, 2), rou

# Display the results in a table
headers = ['Model', 'AUC', 'F1', 'Precision', 'Recall', 'Accuracy']
table = tabulate(results, headers, tablefmt='grid')

print(table)

```

Model	AUC	F1	Precision	Recall	Accuracy
random forest	0.64	0.96	0.45	0.3	0.97
Logistic Regression	0.82	0.94	0.24	0.73	0.92
AdaBoost	0.73	0.94	0.21	0.52	0.92
SVM	0.76	0.96	0.35	0.55	0.95
KNN	0.83	0.94	0.25	0.73	0.92
Decision Tree	0.68	0.94	0.22	0.41	0.93
Naive Bayes	0.76	0.94	0.23	0.59	0.92
XGBoost	0.78	0.96	0.35	0.59	0.95

□ SMOTE EEN

```

from imblearn.combine import SMOTEENN
# Create a SMOTE-ENN object
smote_enn = SMOTEENN(sampling_strategy = 0.5, random_state=42)

# Apply SMOTE-ENN to the training data
X_train_smoteenn, y_train_smoteenn = smote_enn.fit_resample(X_train, y_train)

s_scaler = StandardScaler()
s_scaler.fit(X_train_smoteenn)
X_train_scaled_smoteenn = s_scaler.transform(X_train_smoteenn)
X_test_scaled_smoteenn = s_scaler.transform(X_test)

X_train_scaled_smoteenn = pd.DataFrame(X_train_scaled_smoteenn, columns=X_train.columns)
X_test_scaled_smoteenn = pd.DataFrame(X_test_scaled_smoteenn, columns=X_test.columns)

```

	ROA(C)	ROA(A)	ROA(B)	Operating	Realized
	before interest and depreciation	before interest and % tax after	before interest and depreciation after tax	Gross Margin	Sales Gross Margin
0	0.121357	0.146370	0.152795	-0.638606	-0.640130
1	0.841453	0.886990	0.673419	-0.566448	-0.579780
2	0.699681	0.816408	0.548585	0.123059	0.124307
3	-0.305687	-0.096196	-0.272994	-0.930578	-0.933165
4	-1.691418	-1.586632	-1.701563	2.661273	2.656506
...
6591	-0.799816	-0.377014	-0.600826	-0.622833	-0.624299
6592	-0.306780	-0.184680	-0.282105	-0.544336	-0.545517
6593	-1.510797	-1.552301	-1.564946	-1.001968	-0.947515
6594	-0.838809	-0.667493	-0.883488	1.014672	1.019167
6595	-1.754956	-1.586632	-1.701563	-0.380753	-0.381338

6596 rows x 93 columns

```
X_test_scaled_smoteeen = pd.DataFrame(X_test_scaled, columns=X_test.columns)
X_test_scaled_smoteeen
```

	ROA(C)	ROA(A)	ROA(B)	Operating	Realized
	before interest and depreciation	before interest and % tax after	before interest and depreciation after tax	Gross Margin	Sales Gross Margin
0	0.341049	0.304943	0.343097	0.391504	0.393449
1	0.031512	0.149933	-0.052402	-0.596194	-0.597836
2	1.258293	1.456444	1.390583	0.247619	0.251740
3	2.509121	2.399336	2.483834	2.322525	2.331482
4	-0.602427	-0.547611	-0.696556	2.697703	2.692670
...
1359	0.014024	0.594831	-0.006391	-0.857742	-0.859659
1360	-0.034068	-0.006083	0.007315	-0.344731	-0.337362
1361	0.683813	0.760913	0.555531	-0.569300	-0.570844
1362	-0.069918	0.142887	-0.047507	-0.795885	-0.805000
1363	-0.604176	-0.328181	-0.442027	-0.917583	-0.920392

1364 rows x 93 columns

□ pca

```
pca = PCA(n_components=34,random_state=7)
```

```
X_train_pca_smoteeen = pca.fit_transform(X_train_scaled_smoteeen)
X_test_pca_smoteeen = pca.transform(X_test_scaled_smoteeen)
```

□ random forest

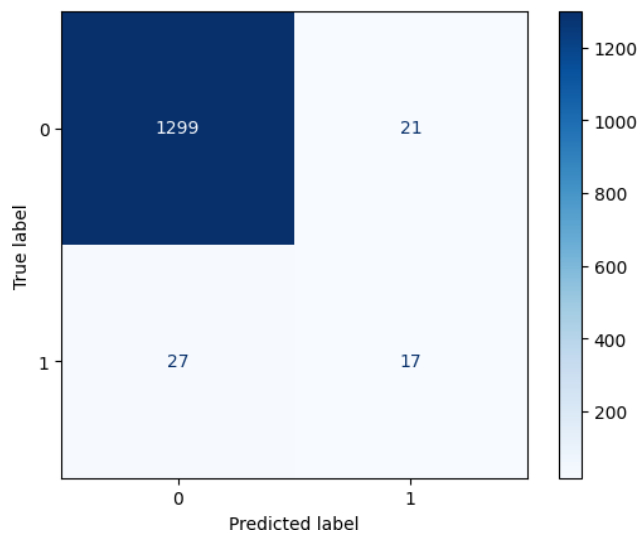
```
rf_smoteeen_model = RandomForestClassifier(random_state = 0, n_estimators = 100)
rf_smoteeen_model.fit(X_train_pca_smoteeen, y_train_smoteeen)
```

```
▼ RandomForestClassifier
RandomForestClassifier(random_state=0)
```

```
rf_smoteeen_y_pred = rf_smoteeen_model.predict(X_test_pca_smoteeen)
```

```
print(classification_report(y_test, rf_smoteeen_y_pred))
ConfusionMatrixDisplay.from_predictions(y_test, rf_smoteeen_y_pred, cmap='Blues')
plt.grid(False)
```

	precision	recall	f1-score	support
0	0.98	0.98	0.98	1320
1	0.45	0.39	0.41	44
accuracy			0.96	1364
macro avg	0.71	0.69	0.70	1364
weighted avg	0.96	0.96	0.96	1364



```
auc = round(metrics.roc_auc_score(y_test, rf_smoteeen_y_pred), 2)
f1 = round(metrics.f1_score(y_test, rf_smoteeen_y_pred, average='weighted'), 2)
precision = round(metrics.precision_score(y_test, rf_smoteeen_y_pred), 2)
recall = round(metrics.recall_score(y_test, rf_smoteeen_y_pred), 2)
accuracy = round(metrics.accuracy_score(y_test, rf_smoteeen_y_pred), 2)
print("auc", auc)
print("f1", f1)
print("precision", precision)
print("recall", recall)
print("accuracy", accuracy)
```

```
auc 0.69
f1 0.96
precision 0.45
recall 0.39
accuracy 0.96
```

□ logistic regression


```

from sklearn.linear_model import LogisticRegression
# Create a logistic regression model
logreg_model_smoteeen = LogisticRegression(max_iter=1000, random_state=42)

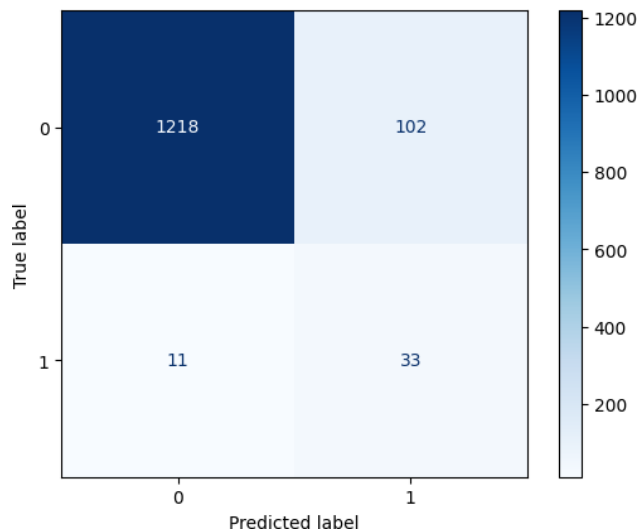
# Train the model
logreg_model_smoteeen.fit(X_train_pca_smoteeen, y_train_smoteeen)

# Make predictions
log_smoteeen_y_pred = logreg_model_smoteeen.predict(X_test_pca_smoteeen)

print(classification_report(y_test, log_smoteeen_y_pred))
ConfusionMatrixDisplay.from_predictions(y_test, log_smoteeen_y_pred, cmap='Blues')
plt.grid(False)

```

	precision	recall	f1-score	support
0	0.99	0.92	0.96	1320
1	0.24	0.75	0.37	44
accuracy			0.92	1364
macro avg	0.62	0.84	0.66	1364
weighted avg	0.97	0.92	0.94	1364



```

auc = round(metrics.roc_auc_score(y_test,log_smoteeen_y_pred), 2)
f1 = round(metrics.f1_score(y_test, log_smoteeen_y_pred, average='weighted'), 2)
precision = round(metrics.precision_score(y_test,log_smoteeen_y_pred), 2)
recall = round(metrics.recall_score(y_test,log_smoteeen_y_pred), 2)
accuracy = round(metrics.accuracy_score(y_test,log_smoteeen_y_pred), 2)
print("auc",auc)
print("f1",f1)
print("precision",precision)
print("recall",recall)
print("accuracy",accuracy)

auc 0.84
f1 0.94
precision 0.24
recall 0.75
accuracy 0.92

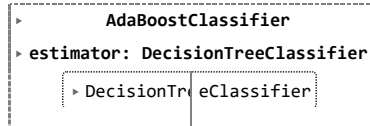
```

□ Adaboost

```
# Create a weak learner (Decision Tree in this case)
base_classifier_smoteen = DecisionTreeClassifier(max_depth=1)

# Create an AdaBoost classifier with the weak learner
adaboost_classifier_smoteen= AdaBoostClassifier(base_classifier_smoteen, n_estimators=50, random_stat

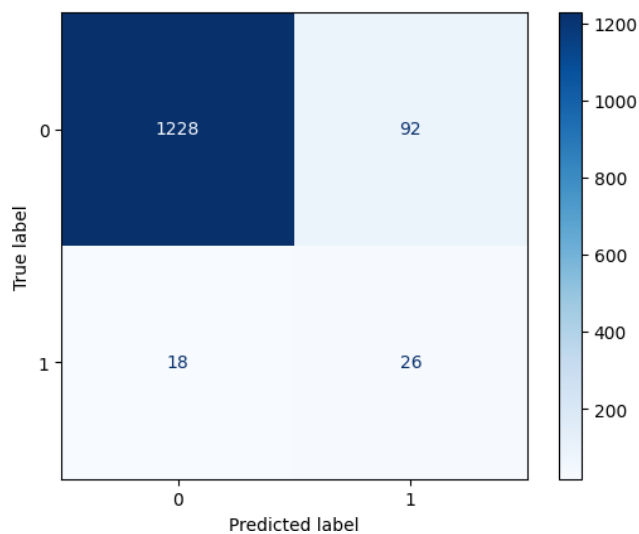
# Train the AdaBoost classifier on the training data
adaboost_classifier_smoteen.fit(X_train_pca_smoteen, y_train_smoteen)
```



```
Adaboost_y_pred_smoteen = adaboost_classifier_smoteen.predict(X_test_pca_smoteen)

print(classification_report(y_test, Adaboost_y_pred_smoteen))
ConfusionMatrixDisplay.from_predictions(y_test, Adaboost_y_pred_smoteen, cmap='Blues')
plt.grid(False)
```

	precision	recall	f1-score	support
0	0.99	0.93	0.96	1320
1	0.22	0.59	0.32	44
accuracy			0.92	1364
macro avg	0.60	0.76	0.64	1364
weighted avg	0.96	0.92	0.94	1364



```
auc = round(metrics.roc_auc_score(y_test,Adaboost_y_pred_smoteen), 2)
f1 = round(metrics.f1_score(y_test,Adaboost_y_pred_smoteen, average='weighted'), 2)
precision = round(metrics.precision_score(y_test,Adaboost_y_pred_smoteen), 2)
recall = round(metrics.recall_score(y_test,Adaboost_y_pred_smoteen), 2)
accuracy = round(metrics.accuracy_score(y_test,Adaboost_y_pred_smoteen), 2)
print("auc",auc)
print("f1",f1)
print("precision",precision)
print("recall",recall)
print("accuracy",accuracy)

auc 0.76
f1 0.94
precision 0.22
recall 0.59
accuracy 0.92
```

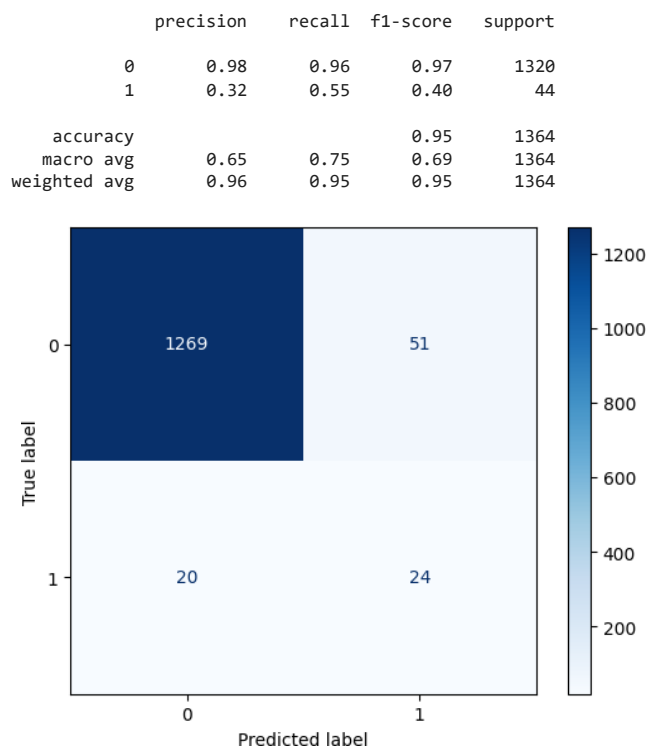
□ SVM

```
# Create an SVM classifier
svm_classifier_smoteeen= SVC(kernel='rbf', C=1.0, random_state=42)

# Train the SVM classifier on the training data
svm_classifier_smoteeen.fit(X_train_pca_smoteeen, y_train_smoteeen)

# Make predictions on the test set
SVC_y_pred_smoteeen = svm_classifier_smoteeen.predict(X_test_pca_smoteeen)

print(classification_report(y_test, SVC_y_pred_smoteeen))
ConfusionMatrixDisplay.from_predictions(y_test, SVC_y_pred_smoteeen, cmap='Blues')
plt.grid(False)
```



```
auc = round(metrics.roc_auc_score(y_test,SVC_y_pred_smoteeen), 2)
f1 = round(metrics.f1_score(y_test,SVC_y_pred_smoteeen, average='weighted'), 2)
precision = round(metrics.precision_score(y_test,SVC_y_pred_smoteeen), 2)
recall = round(metrics.recall_score(y_test,SVC_y_pred_smoteeen), 2)
accuracy = round(metrics.accuracy_score(y_test,SVC_y_pred_smoteeen), 2)
print("auc",auc)
print("f1",f1)
print("precision",precision)
print("recall",recall)

auc 0.75
f1 0.95
precision 0.32
recall 0.55
```

□ Decision Tree

```

# Initialize the Decision Tree classifier
decision_tree_classifier_smoteeen = DecisionTreeClassifier(random_state=42)

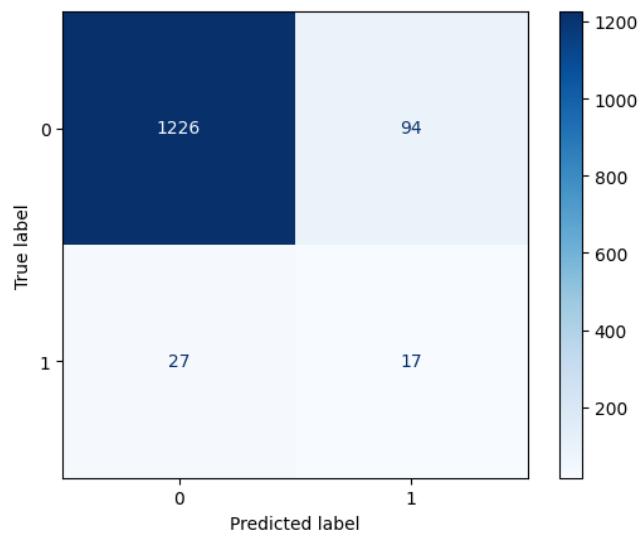
# Train the model on the training set
decision_tree_classifier_smoteeen.fit(X_train_pca_smoteeen, y_train_smoteeen)

print(classification_report(y_test,DT_y_pred_smoteeen))

ConfusionMatrixDisplay.from_predictions(y_test,DT_y_pred_smoteeen, cmap='Blues')plt.grid(False)

```

	precision	recall	f1-score	support
0	0.98	0.93	0.95	1320
1	0.15	0.39	0.22	44
accuracy			0.91	1364
macro avg	0.57	0.66	0.59	1364
weighted avg	0.95	0.91	0.93	1364



```

auc = round(metrics.roc_auc_score(y_test,DT_y_pred_smoteeen), 2)
f1 = round(metrics.f1_score(y_test,DT_y_pred_smoteeen, average='weighted'), 2)
precision = round(metrics.precision_score(y_test,DT_y_pred_smoteeen), 2)
recall = round(metrics.recall_score(y_test,DT_y_pred_smoteeen), 2)
accuracy = round(metrics.accuracy_score(y_test,DT_y_pred_smoteeen), 2)
print("auc",auc)
print("f1",f1)

```

✓ Naive Bayes

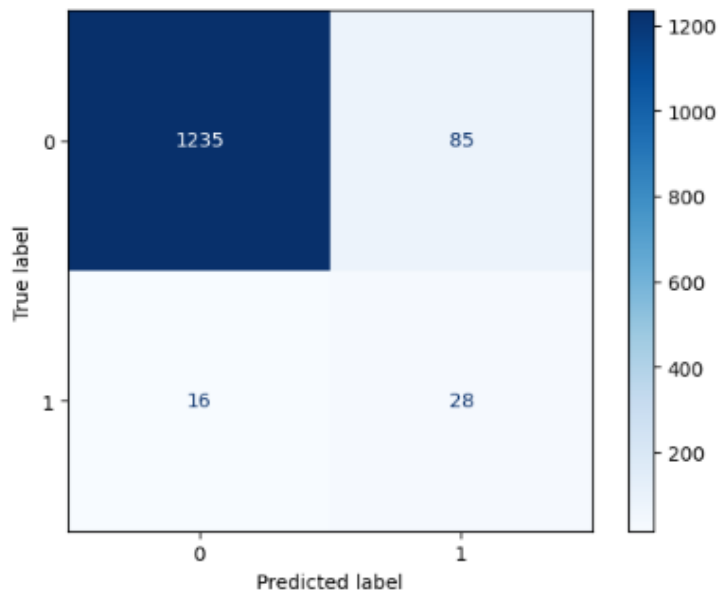
```
[ ] # Initialize the Gaussian Naive Bayes classifier
naive_bayes_classifier_smoteeen = GaussianNB()

# Train the model on the training set
naive_bayes_classifier_smoteeen.fit(X_train_pca_smoteeen, y_train_smoteeen)

# Make predictions on the testing set
Naive_y_pred_smoteeen = naive_bayes_classifier_smoteeen.predict(X_test_pca_smoteeen)

[ ] print(classification_report(y_test, Naive_y_pred_smoteeen))
ConfusionMatrixDisplay.from_predictions(y_test, Naive_y_pred_smoteeen, cmap='Blues')
plt.grid(False)
```

	precision	recall	f1-score	support
0	0.99	0.94	0.96	1320
1	0.25	0.64	0.36	44
accuracy			0.93	1364
macro avg	0.62	0.79	0.66	1364
weighted avg	0.96	0.93	0.94	1364



```
[ ] auc = round(metrics.roc_auc_score(y_test,Naive_y_pred_smoteeen), 2)
f1 = round(metrics.f1_score(y_test,Naive_y_pred_smoteeen, average='weighted'), 2)
precision = round(metrics.precision_score(y_test,Naive_y_pred_smoteeen), 2)
recall = round(metrics.recall_score(y_test,Naive_y_pred_smoteeen), 2)
accuracy = round(metrics.accuracy_score(y_test,Naive_y_pred_smoteeen), 2)
print("auc",auc)
print("f1",f1)
print("precision",precision)
print("recall",recall)
```

```
auc 0.79
f1 0.94
precision 0.25
recall 0.64
```

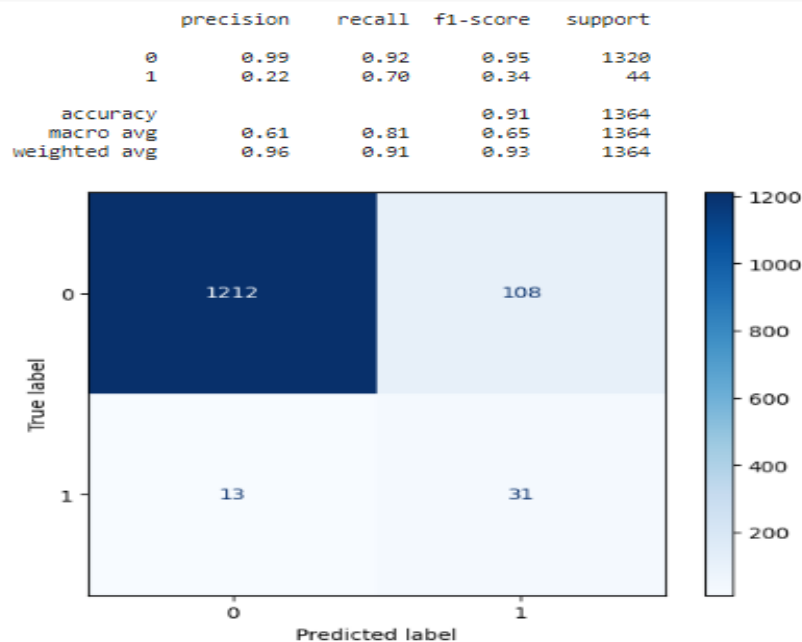
✓ KNN

```
✓ [1029] # Create a KNN classifier with k=3 (you can adjust the value of k)
      knn_classifier_smoteeen = KNeighborsClassifier(n_neighbors=3)

      # Train the KNN classifier on the training data
      knn_classifier_smoteeen.fit(X_train_pca_smoteeen, y_train_smoteeen)

      # Make predictions on the test set
      knn_y_pred_smoteeen = knn_classifier_smoteeen.predict(X_test_pca_smoteeen)

✓ [1030] print(classification_report(y_test, knn_y_pred_smoteeen))
      ConfusionMatrixDisplay.from_predictions(y_test, knn_y_pred_smoteeen, cmap='Blues')
      plt.grid(False)
```



```

[1031] auc = round(metrics.roc_auc_score(y_test,knn_y_pred_smoteeen), 2)
      f1 = round(metrics.f1_score(y_test, knn_y_pred_smoteeen, average='weighted'), 2)
      precision = round(metrics.precision_score(y_test,knn_y_pred_smoteeen), 2)
      recall = round(metrics.recall_score(y_test,knn_y_pred_smoteeen), 2)
      accuracy = round(metrics.accuracy_score(y_test,knn_y_pred_smoteeen), 2)
      print("auc",auc)
      print("f1",f1)
      print("precision",precision)
      print("recall",recall)
      print("accuracy",accuracy)

auc 0.81
f1 0.93
precision 0.22
recall 0.7
accuracy 0.91

```

✓ XGboost

```

[1032] # Initialize the XGBoost classifier
      xgb_classifier_smoteeen = xgb.XGBClassifier(random_state=42)

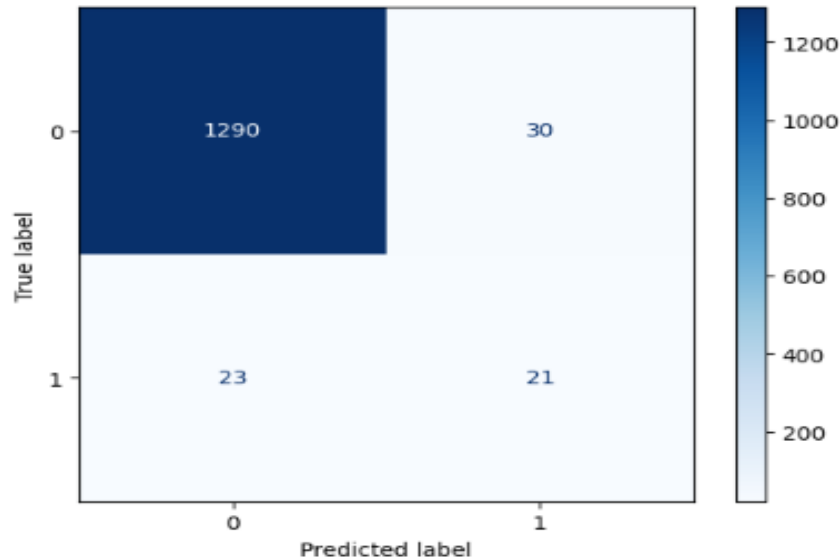
      # Train the model on the training set
      xgb_classifier_smoteeen.fit(X_train_pca_smoteeen, y_train_smoteeen)

      # Make predictions on the testing set
      XG_y_pred_smoteeen = xgb_classifier_smoteeen.predict(X_test_pca_smoteeen)

```

```
[1033] print(classification_report(y_test, XG_y_pred_smoteeen))
ConfusionMatrixDisplay.from_predictions(y_test,XG_y_pred_smoteeen, cmap='Blues')
plt.grid(False)
```

	precision	recall	f1-score	support
0	0.98	0.98	0.98	1320
1	0.41	0.48	0.44	44
accuracy			0.96	1364
macro avg	0.70	0.73	0.71	1364
weighted avg	0.96	0.96	0.96	1364



```
[1034] auc = round(metrics.roc_auc_score(y_test,XG_y_pred_smoteeen), 2)
f1 = round(metrics.f1_score(y_test, XG_y_pred_smoteeen, average='weighted'), 2)
precision = round(metrics.precision_score(y_test,XG_y_pred_smoteeen), 2)
recall = round(metrics.recall_score(y_test,XG_y_pred_smoteeen), 2)
accuracy = round(metrics.accuracy_score(y_test,XG_y_pred_smoteeen), 2)
print("auc",auc)
print("f1",f1)
print("precision",precision)
print("recall",recall)
print("accuracy",accuracy)
```

```
auc 0.73
f1 0.96
precision 0.41
recall 0.48
accuracy 0.96
```


▼ comparison

```
[1035: from tabulate import tabulate
      from sklearn import metrics

models = [rf_smoteen_model, svm_classifier_smoteen, adaboost_classifier_smoteen, decision_tree_classifier_smoteen, naive_bayes_classifier_smoteen, logreg_model_smoteen, knn_classifier_smoteen, xgb_classifier_smoteen, qda_classifier_smoteen]
model_names = ['random forest', 'SVM', 'adaboost', 'Decision Tree', 'naive bayes', 'Logistic Regression', 'KNN', 'XGBoost', 'qda']

results = []

for model, model_name in zip(models, model_names):
    #auc = metrics.roc_auc_score(y_test, model.predict(X_test_pca_smoteen))
    f1 = metrics.f1_score(y_test, model.predict(X_test_pca_smoteen), average='weighted')
    precision = metrics.precision_score(y_test, model.predict(X_test_pca_smoteen))
    recall = metrics.recall_score(y_test, model.predict(X_test_pca_smoteen))
    accuracy = metrics.accuracy_score(y_test, model.predict(X_test_pca_smoteen))

    results.append([model_name, round(auc, 2), round(f1, 2), round(precision, 2), round(recall, 2), round(accuracy, 2)])

# Display the results in a table
headers = ['Model', 'AUC', 'F1', 'Precision', 'Recall', 'Accuracy']
table = tabulate(results, headers, tablefmt='grid')

print(table)
```

Model	AUC	F1	Precision	Recall	Accuracy
random forest	0.73	0.96	0.45	0.39	0.96
SVM	0.73	0.95	0.32	0.55	0.95
adaboost	0.73	0.94	0.22	0.59	0.92
Decision Tree	0.73	0.93	0.15	0.39	0.91
naive bayes	0.73	0.94	0.25	0.64	0.93
Logistic Regression	0.73	0.94	0.24	0.75	0.92
KNN	0.73	0.93	0.22	0.7	0.91
XGBoost	0.73	0.96	0.41	0.48	0.96
qda	0.73	0.94	0.24	0.77	0.92

▼ HYPER PARMETER TUNING

```
[1059]: import numpy as np
        from sklearn.model_selection import train_test_split, RandomizedSearchCV
        from sklearn.linear_model import LogisticRegression
        from sklearn.metrics import f1_score
        from scipy.stats import uniform, loguniform

        # Assuming X and y are your data and labels

        # Split the data into train and test sets
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

        # Define the hyperparameter search space
        param_dist = {
            'C': loguniform(1e-5, 1e5), # Regularization parameter
            'penalty': ['l1', 'l2'],
            'solver': ['liblinear', 'saga'],
        }

        # Instantiate Logistic Regression model
        logreg = LogisticRegression()

        # Create RandomizedSearchCV object
        random_search = RandomizedSearchCV(logreg, param_distributions=param_dist, scoring='accuracy', n_iter=10, cv=5, n_jobs=-1, random_state=42)

        # Perform the Randomized Search
        random_search.fit(X_train, y_train)

        # Get the best hyperparameters
        best_params = random_search.best_params_

        print("Best Hyperparameters:", best_params)

        # Predict on the test set with the best model
        y_test_pred = random_search.predict(X_test)

        # Calculate F1 score on the test set
        f1_test = f1_score(y_test, y_test_pred)
        print("F1 Score on Test Set:", f1_test)

Best Hyperparameters: {'C': 0.05564180225431373, 'penalty': 'l1', 'solver': 'liblinear'}
F1 Score on Test Set: 0.8
/usr/local/lib/python3.10/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
```

```
▶ from sklearn.linear_model import LogisticRegression
   # Create a logistic regression model
   logreg_model = LogisticRegression(max_iter=1000, random_state=42, C=0.05564180225431373, penalty='l1', solver='liblinear')

   # Train the model
   logreg_model.fit(X_train_pca, y_train)

   # Make predictions
   log_y_pred = logreg_model.predict(X_test_pca)
```

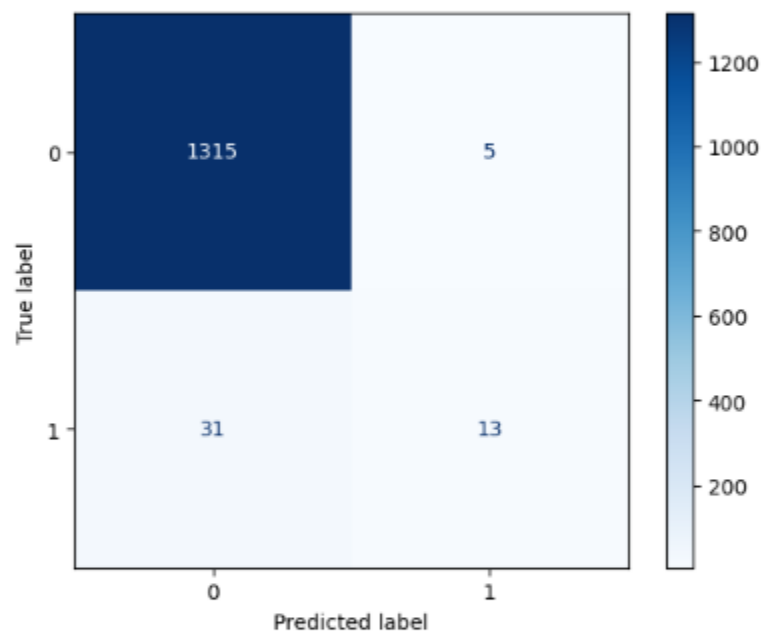
```
[1136] print(classification_report(y_test, log_y_pred))
ConfusionMatrixDisplay.from_predictions(y_test, log_y_pred, cmap='Blues')
plt.grid(False)
```

```

precision    recall  f1-score   support

0           0.98        1.00        0.99        1320
1           0.72        0.30        0.42         44

accuracy          0.97        1364
macro avg          0.85        0.65        0.70        1364
weighted avg          0.97        0.97        0.97        1364
```



```
[1137] auc = round(metrics.roc_auc_score(y_test,log_y_pred), 2)
f1 = round(metrics.f1_score(y_test, log_y_pred, average='weighted'), 2)
precision = round(metrics.precision_score(y_test, log_y_pred), 2)
recall = round(metrics.recall_score(y_test,log_y_pred), 2)
accuracy = round(metrics.accuracy_score(y_test,log_y_pred), 2)
print("auc",auc)
print("f1",f1)
print("precision",precision)
print("recall",recall)
print("accuracy",accuracy)
```

```

auc 0.65
f1 0.97
precision 0.72
recall 0.3
```