INDEX

Abstract

Financial statement fraud and corporate distress present significant and ongoing challenges to the stability of global financial markets. Although traditional statistical methods, such as Altman's Z-score, offer a basic foundation for analysis, they often fall short in their ability to detect the complex, non-linear patterns that may indicate fraudulent activity. (Alareeni, 2020)

This study employs advanced machine learning (ML) techniques to develop a more accurate and robust predictive model for assessing financial distress and fraud risk. We have constructed a comprehensive longitudinal dataset comprising 5,000 firms from diverse industries. Our methodology utilises a target variable strategy, integrating both quantitative metrics and qualitative risk assessments conducted by auditors, which provide insights into the potential for fraudulent activities based on professional judgment. By merging these diverse data sources, we aim to enhance the predictive capabilities of our model, thereby providing a deeper understanding of the factors that contribute to financial instability and fraudulent behaviour in organisations.

After conducting extensive feature engineering, sector-specific data imputation, and rigorous temporal validation to prevent look-ahead bias, we benchmarked multiple classification algorithms. Our findings clearly demonstrate the superiority of ensemble methods, with Random Forest achieving the highest performance metrics: an ROC AUC of 0.831 and an F1-Score of 0.670. The model identifies key predictors, including profitability ratios, cash flow metrics, and leverage, while maintaining robust performance over time. This research confirms that machine learning ensembles represent a significant advancement over traditional models, providing a more accurate and reliable tool for investors, auditors, and regulators in proactive risk assessment and fraud detection. (Hossain, 2024)

Keywords: Financial Fraud Detection, Corporate Distress, Machine Learning, Random Forest, XGBoost, LightGBM, Ensemble Methods, Altman Z-Score, Temporal Validation.

Introduction

Financial statement fraud and corporate distress represent significant challenges that undermine investor confidence, disrupt market efficiency, and jeopardise economic stability. The ability to accurately forecast these issues is a critical objective for investors, regulators, and auditors worldwide. Traditional statistical techniques, such as Multiple Discriminant Analysis (MDA), introduced by Altman (1968) with the Z-Score model, have established the foundational framework for bankruptcy prediction. (Altman, 1968) However, these models are inherently constrained by their reliance on linear assumptions and static financial ratios, often failing to capture the complex, non-linear, and dynamic patterns that signal sophisticated fraudulent activities or evolving financial distress. (Schreyer, 2017)

The advancement of machine learning (ML) has opened up new possibilities for addressing existing limitations in various applications. Algorithms such as Random Forest, XGBoost, and Neural Networks are particularly effective at uncovering complex interactions within high-dimensional datasets, and they do so without the need for rigid, predefined assumptions. This adaptability is especially valuable in the field of fraud detection, where signals can be subtle and often concealed among a wide range of financial and non-financial indicators.

A key challenge in this field is defining the target variable, specifically, what qualifies as a "fraudulent" or "distressed" firm. Research shows a discrepancy between quantitative forensic scores, such as the Beneish M-Score, and the qualitative judgments made by auditors. This "proxy debate" highlights the need for a holistic approach that combines data-driven metrics with expert assessments to create a comprehensive risk model.

This research combines these findings to create and validate an enhanced predictive framework. We address significant methodological issues, such as class imbalance, temporal data leakage, and sector-specific differences, by employing techniques like class-weighted learning, stringent temporal train-test splits, and sector-specific KNN imputation. By assessing a range of models, from interpretable logistic regression to sophisticated ensemble techniques, we reveal a distinct performance hierarchy and establish a reliable, practical tool for forecasting financial risk.

The remainder of this paper elaborates on our data collection and preparation process, including feature engineering and data cleaning. It also presents our exploratory data analysis, highlighting significant patterns and relationships. The details of our modelling methodology and evaluation framework. This paper presents the results of a comparison between different models, identifying key predictive features. Finally, we discuss the implications of our findings and suggest potential directions for future research.

Literature Review

Financial statement fraud and corporate distress signals continue to pose significant challenges for investors, regulators, and auditors worldwide. Traditional statistical methods, while helpful, often fall short in their ability to effectively capture the intricate and non-linear patterns that signal fraudulent activities or looming financial crises. These methods typically rely on linear assumptions and standard metrics, which may overlook subtle indicators of irregularities. Moreover, the dynamic nature of financial markets and the increasingly sophisticated techniques employed by fraudsters further complicate detection efforts. As a result, there is a growing need for more advanced analytical approaches that can better accommodate these complexities and enhance the accuracy of fraud detection and corporate risk assessment.

The foundational study by Altman (1968) demonstrated the ability of financial ratios and Multiple Discriminant Analysis (MDA) to predict bankruptcy, establishing the widely used Z-Score benchmark (Moscone, 2025). (Altman, 1968) The evolution of machine learning (ML) has created new opportunities for improving predictive accuracy. This review summarises existing literature on ML applications in fraud and distress prediction, emphasising key themes such as fraud indicators, algorithm performance, theoretical frameworks, and methodological considerations. It concludes by explaining how our research model incorporates these insights to enhance the field of study.

Defining the Target Variable: The Proxy Debate

A key challenge in fraud detection research is defining the target variable—specifically, what qualifies as a “fraudulent” firm? The literature indicates a discrepancy between calculated forensic scores and auditor judgments.

1. **Auditor Markers as a Qualitative Proxy:**

Auditors play an important role in fraud detection, and prior research has examined both structured and qualitative proxies to capture their influence. Structured proxies rely on characteristics related to auditors, such as the independence of the audit committee, the level of board oversight, rotation practices, auditor tenure, early resignations, and audit fees. These factors are often assessed over several years to capture consistent trends in audit quality. (Carcello, 2004) These variables reflect institutional and economic factors that can influence independence and effectiveness. In contrast, qualitative indicators focus on the informational value of modified audit opinions, which incorporate professional judgment and context that go beyond observable measures. Recent research demonstrates that while algorithms such as Random Forests perform well with both types of data, models like Artificial Neural Networks

(ANN) particularly excel when analysing the nuanced auditor opinion proxies. (Nguyen, 2025) This suggests that these proxies capture a unique aspect of fraud risk. Together, these methods emphasise the complementary roles of structured auditor attributes and judgment-based signals in evaluating fraud risk.

1. **M-Score as a Quantitative Proxy:**

The M-Score, which is based on financial ratios and accrual models, provides a quantitative and reproducible measure of potential manipulation in financial statements. Research conducted by Li et al. (2024) and Perols (2011) effectively uses the M-Score as a fraud detection tool, demonstrating its usefulness in identifying accounting anomalies through data-driven models. (Perols, 2011)

Our research model effectively addresses the proxy debate through a dual-target approach. We compute a longitudinal M-Score to detect quantitative anomalies, while also utilising an Overall Audit Fraud Risk rating as a proxy for qualitative auditor assessment. This thorough methodology ensures that our model is resilient, integrating both data-driven signals and expert judgment, consistent with the framework established by Nguyen Thanh An and Phan Huy (2025). (Nguyen, 2025)

Algorithm Performance: The Superiority of Ensemble Methods

A recurring observation throughout the literature is that advanced, ensemble machine learning algorithms tend to perform better than conventional statistical methods and simpler machine learning models.

1. **Ensemble Dominance:** Random Forest and XGBoost are consistently recognized as leading performers. Studies by Nguyen Thanh An & Phan Huy (2025), Lee et al. (2025), Ali et al. (2023), and Zhao & Bai (2022) have all indicated that these ensemble techniques achieve the highest levels of accuracy, AUC, and F1 scores in fraud detection. (Zhao, 2022) Their capacity to capture complex, non-linear relationships between variables without requiring strong prior assumptions makes them particularly well-suited for financial datasets.
2. **Comparative Performance:** Logistic Regression is appreciated for its interpretability, but it often exhibits limited discriminative power (Nguyen Thanh An & Phan Huy, 2025; Perols, 2011). In contrast, Support Vector Machines (SVM) have mixed performance, with some studies highlighting the risk of overfitting (Li et al., 2024). Artificial Neural Networks (ANNs) and Deep Learning models tend to perform well, particularly with large datasets (Elhoseny et al., 2025; Mohammadi et al., 2020). However, they can be computationally intensive and are often seen as "black boxes.” (Nguyen Thanh An & Phan Huy, 2025)

Our model evaluation framework has been meticulously crafted to rigorously test the established hierarchy of predictive models in the context of financial data analysis. We conduct a comparative benchmark analysis, focusing on Logistic Regression, recognised for its interpretability, and contrasting it with more advanced ensemble methods, specifically Random Forest, XGBoost, and LightGBM, which are renowned for their superior predictive capabilities.

The results of our evaluation align with existing literature, revealing that the ensemble methods—particularly Random Forest and XGBoost—substantially outperform the simpler Logistic Regression model. This indicates not only the enhanced predictive power of these methods but also reinforces their applicability to our complex and multivariate financial dataset. Our findings highlight the importance of selecting the appropriate model based on the specific needs of the analysis, emphasizing that while Logistic Regression offers valuable insights through interpretability, advanced techniques provide a significant advantage in terms of accuracy and performance for more intricate datasets.

Financial Ratios as Predictors

The foundational work of Persons (1995) and Altman (1968) laid the groundwork for understanding the importance of financial ratios, specifically leverage, profitability, and liquidity, as crucial indicators in predicting a firm's financial health and performance. These studies demonstrated that these ratios contain significant predictive signals that can aid stakeholders in making informed decisions. (Beaver, 1966)

Building on this foundational knowledge, more recent research by Ha et al. (2023) and Li et al. (2024) has further advanced our understanding of which specific financial ratios are particularly effective in forecasting outcomes. They highlighted three key ratios: Debt-to-Equity, which assesses a company's financial leverage and risk; the Quick Ratio, which evaluates its short-term liquidity by measuring its ability to meet immediate obligations without relying on inventory sales; and Asset Turnover, which indicates how efficiently a company utilises its assets to generate revenue. Together, these ratios provide a comprehensive view of a company’s financial stability and operational efficiency, making them instrumental in financial analysis and investment decision-making.

Our feature set is carefully designed based on both theoretical and empirical insights.

1. **Financial Manipulation Marker:** `m_score_[0-2]`
2. **Financial Distress Marker:** `z_score_[0-3]`
3. **Profitability Ratios:** `ebit_ta_[0-3]` `roa_avg_[0-3]` `roa_act_[0-3]` `roe_act_[0-3]` `asset_turn_[0-3]` `inv_turn_[0-3]` `pe_growth_[0-3]`
4. **Leverage & Capital Structure:** `debt_[0-3]` `debt_cap_[0-3]`

5. **Liquidity Ratios** `current_ratio_[0-3]` `quick_ratio_[0-3]` `wc_ta_[0-3]` `re_ta_[0-3]`
6. **Solvency & Coverage Ratios** `int_cov` `_[0-3]`
7. **Cash Flow Variables** `cfo_[0-3]`
8. **Target variable** `target_[0-3]`

This coverage aligns with the findings of Li et al. (2024) and Ha et al. (2023).

Sectoral and Macro-Economic Considerations

Financial distress and fraud risk are not uniform across industries. The literature strongly advocates for sector-specific analysis. Consequently, Sayari & Mugan (2017) illustrate that the usefulness of financial ratios differs greatly across industries, highlighting the need for developing distress models tailored to specific sectors. (Sayari, 2017) Moscone (2025) reinforces this idea by creating a predictive model.

Furthermore, Betz et al. (2014) and Castrén & Kopp (2025) emphasise the importance of incorporating macro-financial and systemic risk indicators into distress prediction models, particularly for financial institutions. (Betz, 2014)

To take this into perspective, we explicitly account for sectoral differences in two key ways:

1. During **KNN imputation**, missing data is filled based on similar companies within the same sector, not the entire dataset. This preserves sector-specific financial characteristics.
2. The `sector` variable is included as a predictive feature, allowing the model to learn different baseline risks and behavioural patterns across industries, as suggested by Sayari & Mugan (2017). (Sayari, 2017)

Addressing Imbalance and Temporality

To address the inherent class imbalance in fraud datasets, we utilise the SMOTE technique. This approach is employed in cases where non-fraudulent cases vastly outnumber fraudulent cases (Ali et al., 2023; Zhao & Bai, 2022; Li et al., 2024). (Zhao, 2022) To validate any temporal inconsistency and avoid look-ahead bias, models must be trained on past data and tested on future data to ensure real-world applicability. Nguyen Thanh An & Phan Huy (2025) and Ha et al. (2023) structure their analyses to respect this temporal sequence.

In our model, we implement a **temporal train-test split**, training on years 1-3 and testing on the most recent year (year 0). This mirrors a real-world forecasting scenario and prevents data leakage. To handle our ~4:1 class imbalance, we forgo SMOTE in favour of **class-weighted learning**. This technique assigns a higher cost to misclassifying the distressed class and is integrated directly into our algorithm’s training process, just like class_weight='balanced' in

Random Forest. This effectively ensures the model focuses on predicting distress without artificially synthesizing data.

By studying the existing literature, we gain insight into a robust foundation for predicting financial fraud and distress. We encounter the superiority of ensemble algorithms, the importance of feature selection, and the necessity for a sound methodological approach to handling imbalance and temporal data. (Wang, 2025)

Taking all this into consideration, our model synthesises and advances these insights by:

1. Integrating a **proxy** for the overall audit fraud risk marker for a more comprehensive target definition.
2. Evaluating a suite of models from **logistic regression** to advanced ensemble methods (**Random Forest, XGBoost, LightGBM**) to identify the best performer for our specific task.
3. We also use **temporal data split** and **class-weighted learning** to ensure a realistic and robust evaluation that directly addresses class imbalance.
4. For imputation and modelling, we use a more sector-specific **KNN imputation**.
5. Thereby, to make the model more robust, we also integrated **feature engineering and selection.**

By thoroughly analysing the findings of the reviewed literature and synthesising various elements into a cohesive framework, our study seeks to develop a more robust, theoretically grounded model that is practically applicable for predicting financial fraud and distress. This model will leverage established theories and empirical data to enhance its predictive capabilities, enabling a comprehensive understanding of the factors that contribute to these financial issues. By integrating key variables identified in prior research, such as organisational behaviour, economic indicators, and risk management practices, we aim to provide a nuanced approach that not only identifies potential indicators of financial misconduct but also assists practitioners in implementing preventative measures. Ultimately, our goal is to offer a valuable tool for stakeholders in the financial sector, empowering them to mitigate risks and ensure greater fiscal integrity.

Data Preparation and EDA

Data Collection - The Methodology

In our efforts to develop a robust Fraud Detection model, we have chosen to utilise the LSEG Workspace as our primary data collection tool. We focus on selecting the top 5,000 companies ranked by revenue generated from their business operations. This choice is strategic, allowing us to examine a substantial and diverse dataset that reflects various industries and operational scales.

The data collection process is guided by a comprehensive understanding of the challenges associated with detecting financial fraud. We recognise that real-world analyses are often hindered by the unavailability of suitable datasets, which makes the creation of effective and reliable predictive models particularly challenging. (Bockel-Rickermann, 2022) Therefore, we have carefully curated our data sources to ensure they align with the specific requirements of our study, enabling us to address the complexities of financial fraud detection more effectively.

The extensive dataset utilised for this study has been meticulously crafted to offer a thorough analysis of the firm's overall financial health. This dataset not only presents a detailed outlook on various economic indicators but also facilitates a comparative examination of how these metrics align with the company's Audit and Governance framework. ("The Determinants of Fraudulent Financial Reporting: (A Systematic Literature Review)", 2024) By integrating financial performance with governance practices, the study aims to illuminate the interplay between robust financial management and effective oversight mechanisms, thus providing a more nuanced understanding of the factors that contribute to the firm's success and sustainability in the marketplace. ("The Impact of Corporate Governance Mechanism over Financial Performance: Evidence from Romania", 2023)

Therefore, to start understanding the model, we start with the variable analysis:

1. **Company Information**: `ric` , `company_name` , `sector` , `market_cap`
2. **Financial Manipulation Marker:** `m_score_[0-2]`
3. **Financial Distress Marker:** `z_score_[0-3]`
4. **Profitability Ratios:**
    a. Earnings before Interest & Taxes (EBIT) to Total Assets `ebit_ta_[0-3]`
    b. Return on Average Total Assets - % `roa_avg_[0-3]`

c. Return On Assets - Actual `roa_act_[0-3]`

d. Return On Equity - Actual `roe_act_[0-3]`

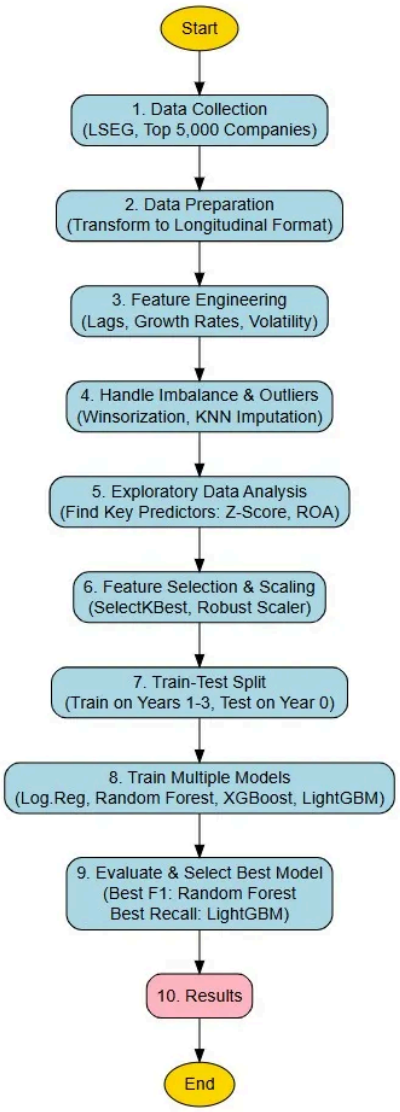
e. Asset Turnover `asset_turn_[0-3]`

f. Inventory Turnover `inv_turn_[0-3]`

g. PE Growth Ratio `pe_growth_[0-3]`

5. **Leverage & Capital Structure:**
    a. Debt - Total `debt_[0-3]`
    b. Net Debt to Total Capital `debt_cap_[0-3]`
6. **Liquidity Ratios**
    a. Current Ratio `current_ratio_[0-3]`
    b. Quick Ratio `quick_ratio_[0-3]`
    c. Working Capital to Total Assets `wc_ta` **`_[0-3]`**
    d. Retained Earnings - Total to Total Assets `re_ta_[0-3]`
7. **Solvency & Coverage Ratios**
    a. Interest Coverage Ratio `int_cov_[0-3]`
8. **Cash Flow Variables**
    a. Net Cash Flow from Operating Activities `cfo_[0-3]`
9. **Target variable**
    a. Overall Audit fraud risk `target_[0-3]`

This comprehensive dataset encompasses 78 variables that address financial health metrics, fraud detection scores, and risk indicators across four fiscal years (FY0 to FY3). It serves as an invaluable resource for conducting longitudinal analyses of fraud.



Overview of the end-to-end workflow used for financial fraud detection, summarising each major phase from initial data collection to model deployment and monitoring. Each step represents a critical process in transforming raw company data into actionable fraud risk predictions, ensuring a robust and explainable machine learning pipeline for audit analytics.

Data Preparation

a) Data Transformation

To enhance our analysis of the dataset and effectively prepare it for modelling purposes, we utilised a data restructuring function that significantly transformed the data format. Initially, our dataset was organised in a wide format, containing 5,000 companies represented in each row along with 75 distinct variable columns. This structure limited our ability to fully comprehend the relationships among the variables over time.

By converting the data into a longitudinal format, we increased the number of rows to 20,000 while reducing the number of columns to 23. This restructuring enables a more detailed examination of the variables, particularly their behaviour and interactions across different time periods. The longitudinal format facilitates a clearer understanding of how each company evolves throughout the years, which is crucial for identifying and analysing trends. (Maynard, n.d.)

Additionally, we capitalised on a Time Series Split to ensure that all companies were represented longitudinally by year. ("Enhancing credit card fraud detection: highly imbalanced data case", 2024) This approach enables us to effectively track fraud patterns over time, providing critical insights into how fraudulent activities may change across different periods. It also aids in the development of predictive features derived from prior years' data, enhancing our modelling efforts.

Moreover, this structure supports both fixed and random effects models, which are instrumental in accounting for individual variability among companies while capturing the overall dynamics of fraud risk. By analysing longitudinal data, we can better assess how dynamic factors contribute to the evolution of fraud risk, ultimately leading to more accurate predictions and effective strategies for fraud detection and prevention. (Mao, 2018)

b) Feature Engineering

We then proceed with feature engineering, in which we enrich the data available to our models by capturing dynamic behaviours such as momentum, volatility, and growth trends that static cross-sectional ratios alone cannot reveal. Therefore, we create a lagged feature that captures the prior year value of each ratio. To study the momentum of how rapidly a metric changes year over year, we use Growth Rates, which are:

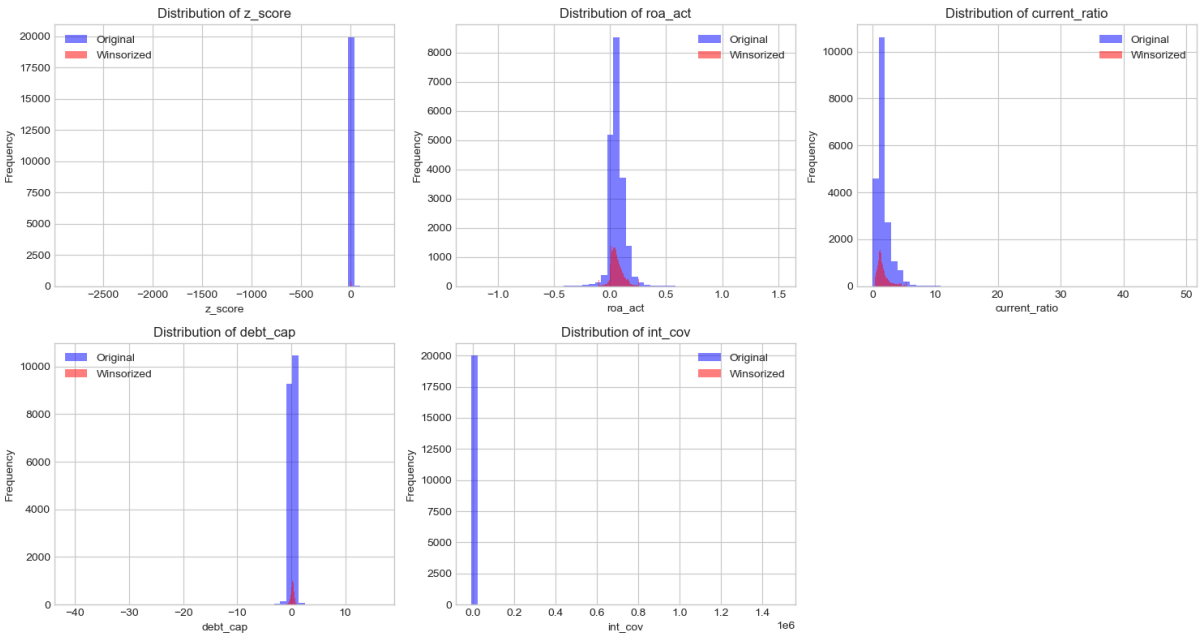
$$Growth = \frac{\text{current} - \text{lag}}{|\text{lag}| + \epsilon}$$

This function is effective in identifying growth anomalies or abrupt swings, as it can signal emerging distress or manipulation. To target volatility features, we use rolling standard deviation over three-year measures of variability or risk in ratios such as ROA or debt ratios. High volatility may signify unstable earnings or changing leverage, often linked to financial distress or earnings manipulation. This is then improved with the help of the trend feature, which captures the mean over a three-year period. A persistent rise or decline in trend reflects structural improvements or deteriorations that persist beyond single-year noise. We also use composite features, which are ratio-based, for example, `cfo_to_debt`, which gauges liquidity cushion against obligations, and `ebit_debt_ratio`, which assesses debt-service capacity. These

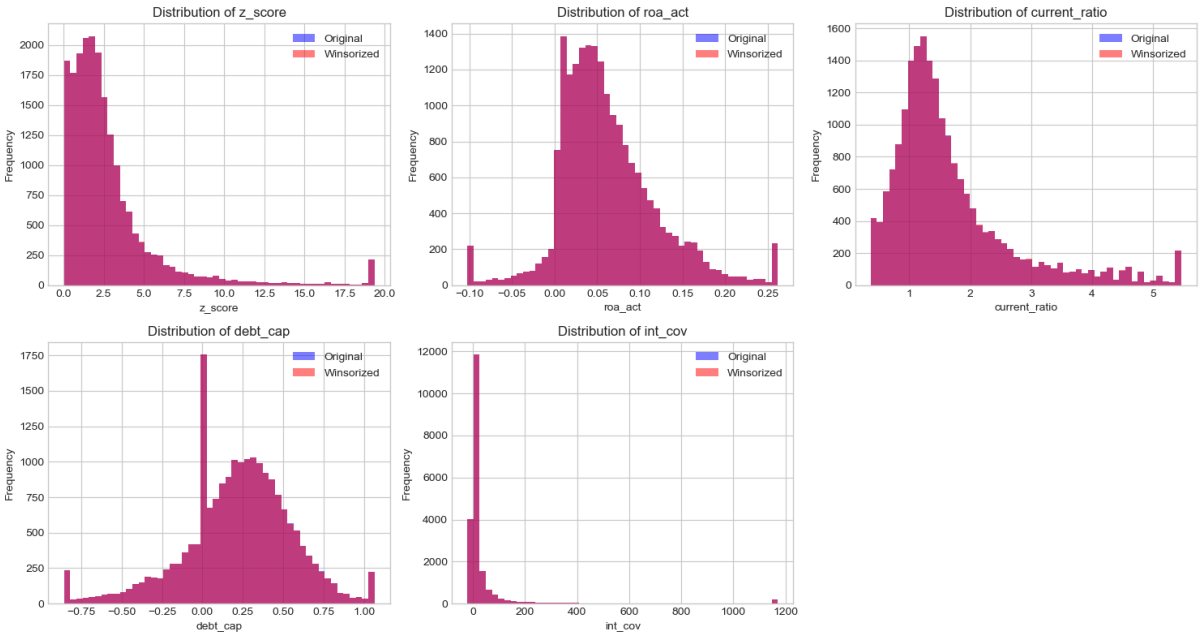
cross-ratio metrics summarise the relationships among profitability, cash flow, and leverage, frequently offering greater predictive power than individual ratios by themselves.

c) Data Cleaning and Outlier Treatment

Following feature engineering, where we analysed the data to obtain the optimal output from the variables, we proceeded to clean the data, detect outliers, and use winsorization to create a more robust dataset. We start by analysing the missing data in each column, using KNN imputations on a sectoral basis. This means imputing the data based on companies within a similar sector, rather than filling the gaps based on the overall financial neighbourhood. We also use KNN imputation because it respects multivariate correlations among ratios, rather than relying solely on univariate trends. (Kaur, 2025) As we have now formed a complete dataset with no missing values, we come to detecting outliers. In datasets of this nature, financial information is frequently dispersed across various variables, making it challenging to gain a cohesive understanding of the data landscape. Identifying and analysing outliers becomes essential, as these anomalies can reveal significant insights into unexpected behaviours, potential fraud, or operational inefficiencies. By closely examining these outlier data points, analysts can gain a deeper understanding of underlying trends, contributing factors, and the overall health of financial performance, ultimately leading to more informed decision-making and strategy development. But an extreme outlier can unduly influence model parameters and obscure genuine patterns. Winsorization shrinks these extremes to plausible bounds while preserving their rank order, reducing skew and improving estimation stability. ("Winsorized Mean: Formula, Examples and Meaning", n.d.)



Before Winsorisation.



After Winsorisation

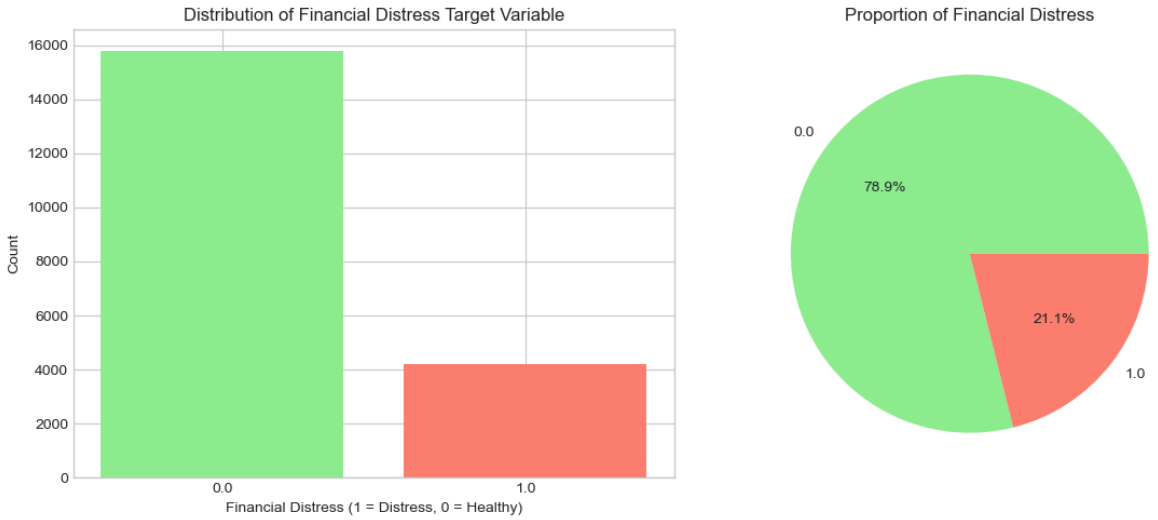
Winsorization at the 1%–99% quantiles efficiently reduces extreme tails across ratios, preserving central tendencies and variance in the mid-distribution. This approach stops a few firms with unusual financial metrics from distorting parameter estimates, enhancing the stability and reliability of subsequent fraud-detection or audit models.

Exploratory Data Analysis

The purpose of EDA is to give a comprehensive integration of multiple analytical perspectives, such as distributional, temporal, sectoral, and correlational.

1. Class Distribution Analysis

Our dataset exhibits a **21.1% distress rate**, with 78.9% of firms being healthy, representing a **4:1** imbalanced classification problem. This imbalance can be classified as a moderate imbalance, which is actually favourable for financial distress modelling.



Visualisation of the class distribution for the financial distress target variable. The bar chart and pie chart show that 21.1% of firms in the dataset are classified as distressed, while 78.9% are healthy, highlighting a moderate class imbalance in the target variable used for model training and evaluation.

Therefore, while evaluating the model we need to prioritise:

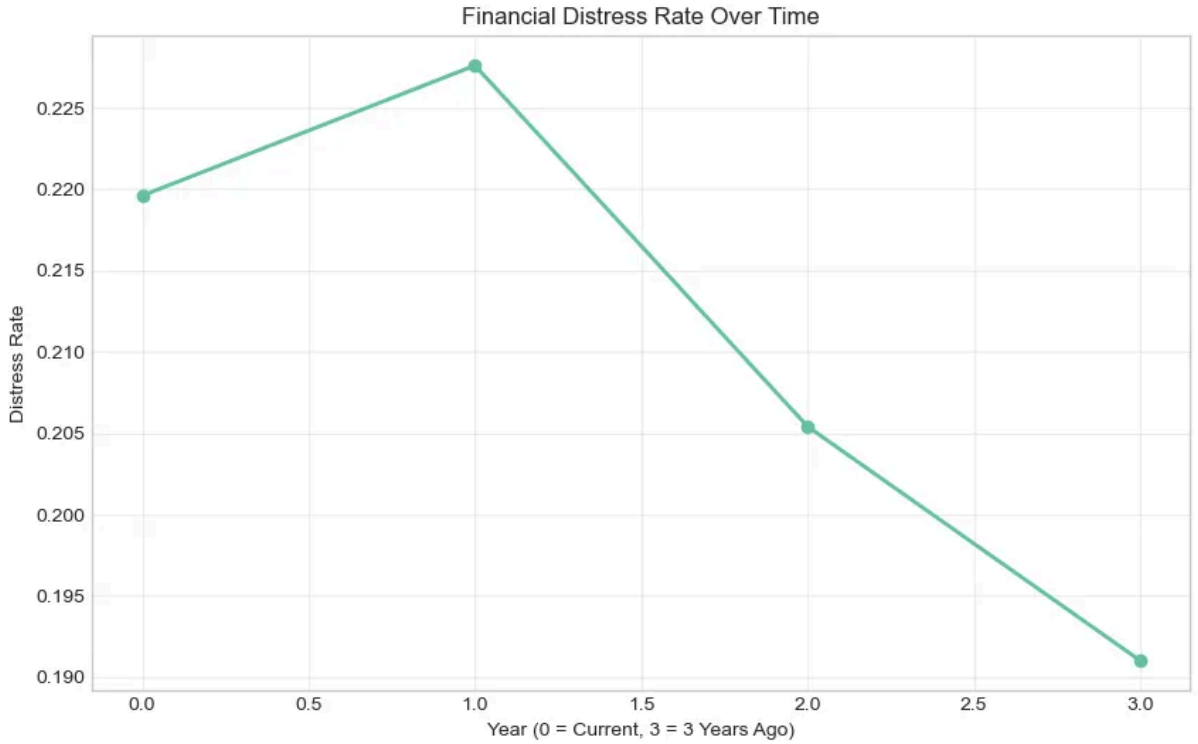
a. **F1-Score- f**or balanced assessment

b. **Precision-Recall curves** over ROC curves for imbalanced data.

c. **Balanced accuracy** to account for both classes equally

2. Temporal Distress Rate Patterns

The temporal analysis reveals that distress rates peak at Year 1, approximately 22.7%, and then decline over the next two years to ~19.1% at Year 3, creating an inverted U-shaped pattern. This pattern aligns with established financial cycle theory, which states that during a financial cycle contraction phase —i.e., where distress manifests most acutely —one year prior to the current observation.



Trend of financial distress rates across four fiscal years, showing an inverted U-shaped pattern with a peak in distress one year prior to the current period, followed by a steady decline. This suggests cyclical effects and possible survivor bias or economic recovery influencing distress prevalence over time.

The declining distress rate toward Year 3 indicates either:

a. **Survivor bias** - severely distressed firms exit the sample in earlier years.

b. **Business cycle recovery** - economic conditions improved 2-3 years ago.

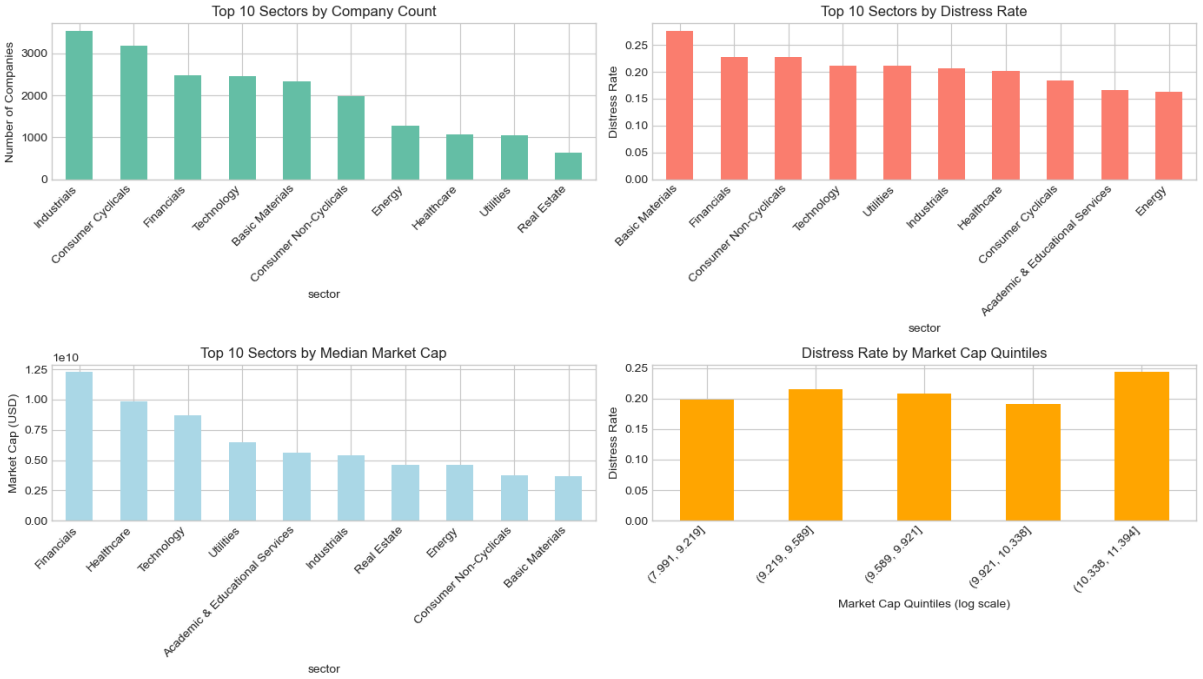
c. **Lagged crisis effects** - firms experienced delayed impacts from the financial shock.

3. Sector Distribution Insights

a. Basic Materials shows the highest distress rate (~27%), primarily due to commodity price volatility and high capital intensity.

b. Financials and Consumer Non-Cyclicals both exhibit above-average distress rates (~22%).

c. Industrials, while representing a large portion of the dataset, still maintains a ~21% distress rate, reflecting its exposure to cyclical capital-goods markets.

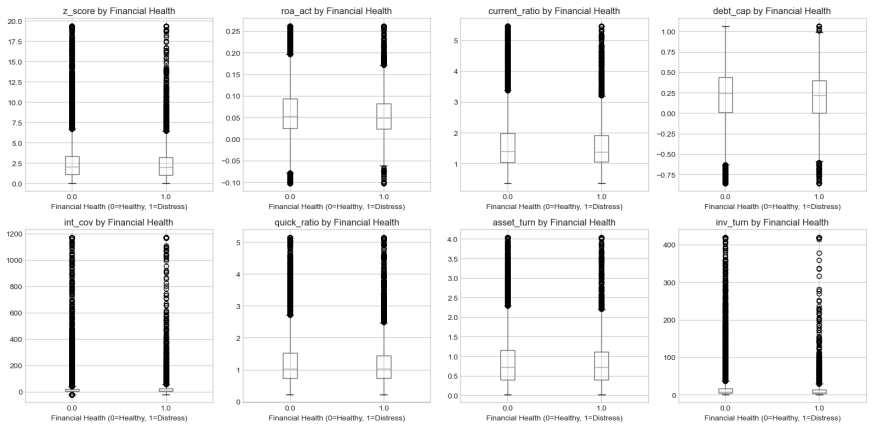


Sector-level and firm size analysis of the dataset. The top left plot shows the largest sectors by company count, while the top right highlights sectors with the highest financial distress rates, with Basic Materials and Financials leading. The bottom left visualizes sectors ranked by median market capitalization, and the bottom right demonstrates that financial distress rates are moderately higher among firms in lower market cap quintiles, reflecting both industry and size-related risk patterns.

4. Financial Ratio Distributions

a. Altman Z-score and ROA display the most significant separation between healthy and distressed firms.

b. Leverage and liquidity ratios overlap heavily, offering weaker standalone predictive power.

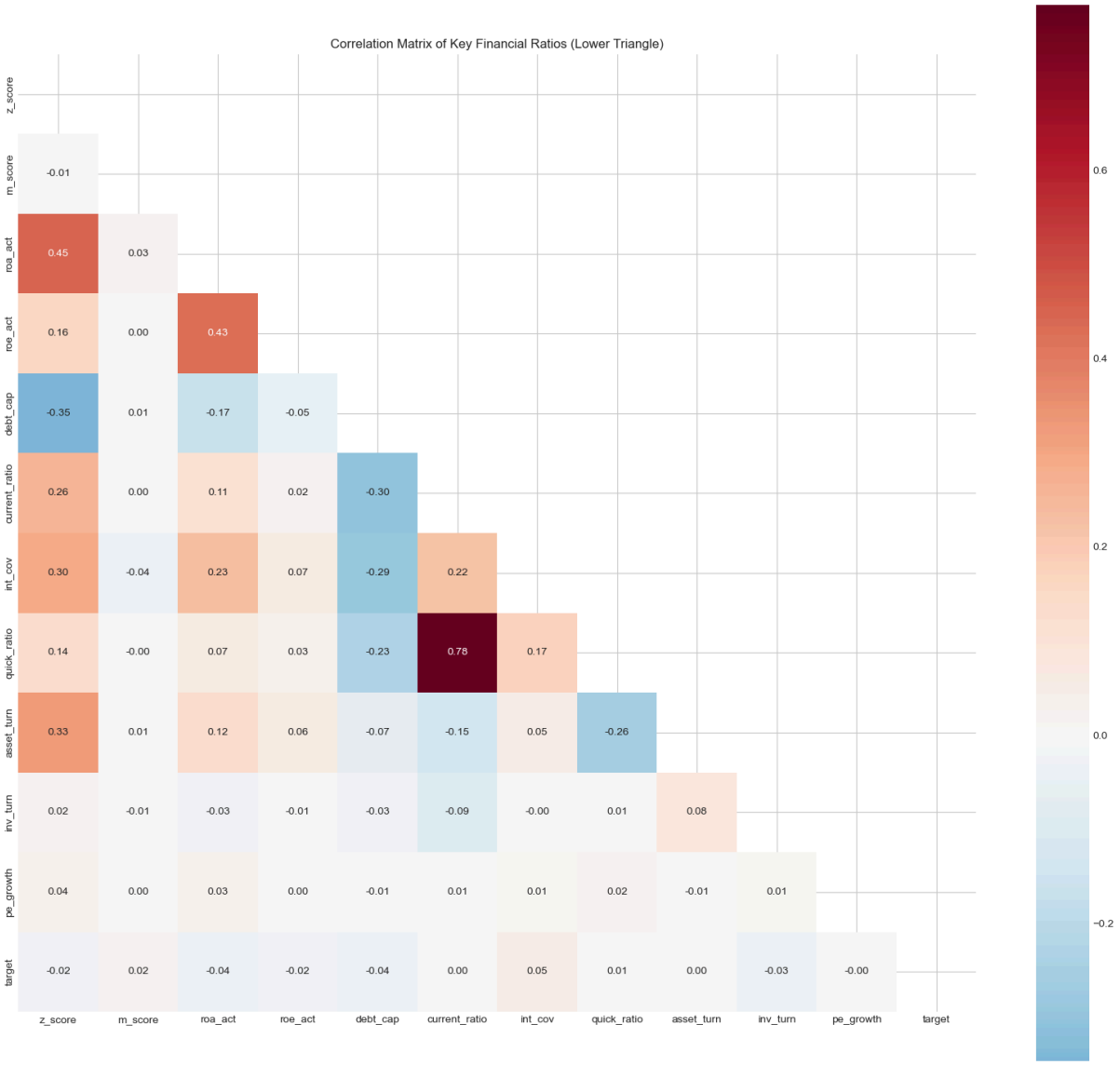


Boxplots comparing distributions of key financial ratios by firm health status. While some metrics, like z_score and roa_act, show clear separation between healthy and distressed firms, others, such as leverage and liquidity ratios, exhibit substantial overlap, indicating weaker predictive power when used in isolation.

5. Correlation Structure

a. High intercorrelation among profitability measures calls for careful feature selection or dimensionality reduction.

b. Weak direct correlations with the target signal necessitate the use of non-linear or ensemble modelling approaches.

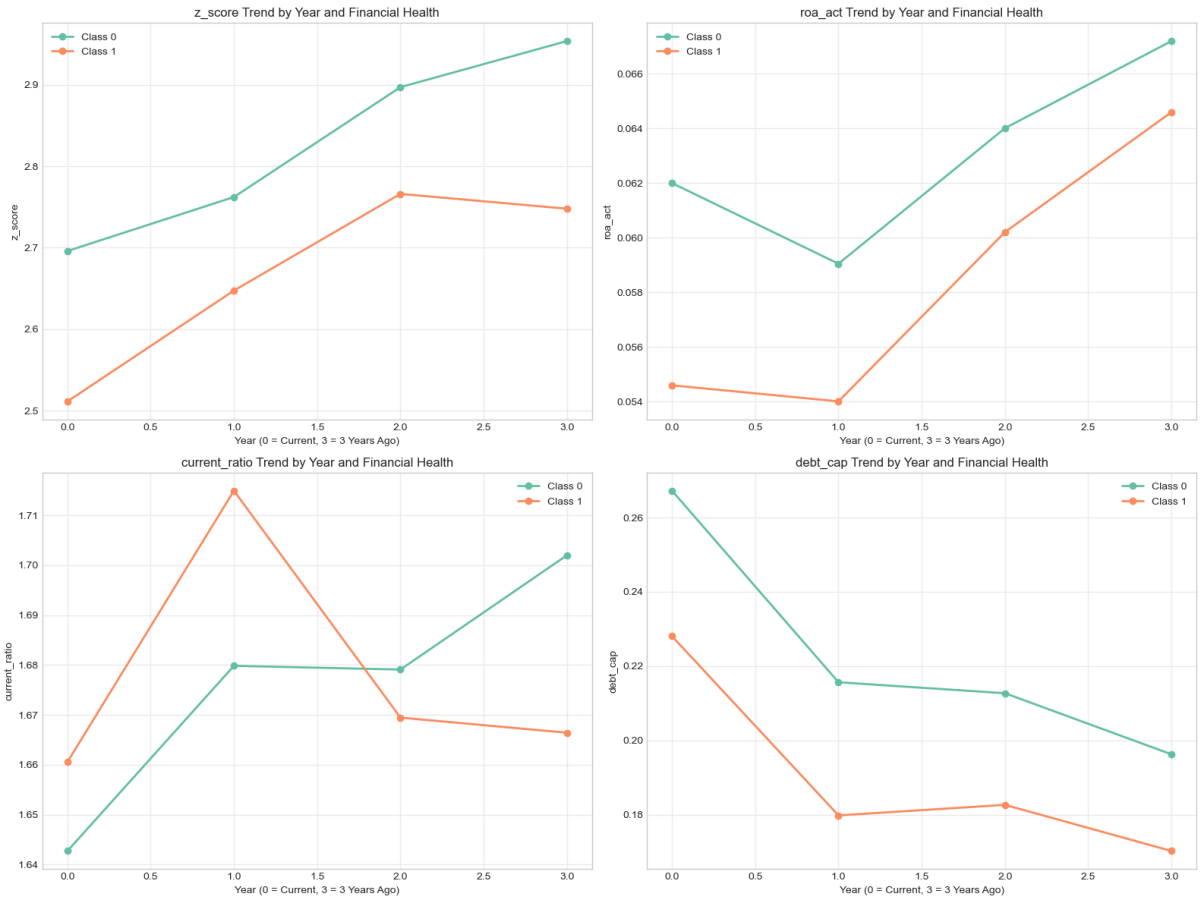


Correlation matrix of key financial ratios and distress target. While some ratios, such as current_ratio and quick_ratio, are highly correlated, most pairs display only weak or moderate relationships. The target variable shows low direct correlation with individual ratios, highlighting the need for non-linear or multivariate modeling approaches.

6. Temporal Trends in Ratios

a. Distressed firms show improving Z-scores and ROA over past three years but remain below healthy peers.

b. Both classes deleveraged over time, with distressed firms reducing debt more sharply.



Temporal trends of key financial ratios for healthy (Class 0) and distressed (Class 1) firms over four years. Healthy firms consistently show higher z_scores and roa_act values, while exhibiting lower debt_cap, compared to distressed firms. The current_ratio trend differs, with distressed firms experiencing sharper short-term changes, illustrating both group differences and dynamic risk patterns over time.

Key findings from financial ratio plot analysis:

1. **Z-Score and ROA Separation**
    a. Healthy firms cluster at higher values; distressed firms at the lower ends.
    b. Density curves show apparent distribution shifts, confirming strong discriminatory power.
2. **Overlap in Leverage and Liquidity Ratios**
    a. Debt-to-capital and current-ratio scatterplots show significant class overlap.

b. Similar distributions between healthy/distressed firms indicate weak standalone predictive value.

3. **Weak Linear Relationships**
    a. Most ratio pairs lack clear linear separation between classes, making simple bivariate thresholds inadequate.
4. **Modelling Implications**
    a. **Non-linear or multivariate methods** will better capture complex interactions.
    b. Engineering should focus on profitability measures and creating composite indicators from weaker ratios.

Pairplot of key financial ratios colored by firm health status, illustrating relationships between z_score, roa_act, debt_cap, and current_ratio. Healthy firms (green) generally cluster at higher z_scores and roa_act values, while distressed firms (red) are more concentrated in lower ranges. However, some overlap remains, emphasizing the need for multivariate analysis in financial distress detection.

Model Training and Evaluation

Data Testing and Classification

To understand the statistical nature of the dataset, and how each numeric feature differs significantly between healthy (target = 0) and distressed (target = 1) firms. For the same reason, we conduct multiple tests:

a. **Welch's t-test** for difference in means.

b. **Mann–Whitney U test for comparing** distributions.

c. Cohen's d, quantifying the mean difference relative to the pooled standard deviation.

d. Controlled false positives across ~50 tests using **Benjamini–Hochberg FDR correction**, balancing Type I error control and statistical power.

And the metrics for each of these tests are defined as follows:

a. **t_statistic & p_value:** Significance of mean difference under Gaussian assumptions.

b. **u_pvalue:** Significance of distributional shift without normality assumption.

c. **P-value corrected & significant:** Adjusted p-values under FDR; "True" indicates statistical significance at 5% level post-correction.

d. **Cohen's d:**

$$d = \frac{\bar{x}_0 - \bar{x}_1}{\sqrt{\frac{s_0^2 + s_1^2}{2}}}$$

Where x̄i and si are the group means and standard deviations.

$$|\ d\ | \approx 0.2\ small, 0.5\ medium, 0.8\ large\ effects.$$

The most robust distress indicators are debt levels and cash flows, aligning with finance theory that high leverage and low cash generation are precursors to distress. The operational stress preceding distress signifies that Volatility Measures capture instability in liquidity and profitability.

Lagged Features (debt_lag1, cfo_lag1) underscore the importance of **early warning signals;** prior-period financials predict current distress.

Size Effect (market_cap) is moderate but significant; smaller firms face slightly higher distress risk after controlling for leverage and cash flows.

We proceed with feature selection, using the `k-best` method to filter out clearly irrelevant features based on class separation quickly. We refine this with RFE to account for feature interactions and multicollinearity in a smaller candidate set. We continue to validate the stability of selected features by performing feature selection within each training fold, thereby avoiding information leakage.
Combine this with automated selection and theoretical insights to ensure that crucial financial ratios are not inadvertently dropped.

As we prepare data for modelling, we sort the dataset by `ric` and `year` to preserve chronological order within each firm. A new column called `future_target` has been introduced, which represents the distress label for one year in the future. Any row lacking a valid future_target is dropped to ensure that every example has both a feature and a known label.
Now we have :

a. Identifiers and metadata: `ric` , `company_name` , `sector` , `year`
b. Current and future labels: `target` , `future_target`
c. Features (`X` **):** All remaining columns comprising engineered financial ratios, lagged features, growth rates, volatility measures, trend metrics, and composites—each reflecting information available at time *t*.
d. Label (`y`): The binary `future_target` indicating distress occurrence at time *t*+1.

The feature matrix `X` and label vector `y` now exclude any incomplete cases and contain only predictive variables aligned with the forecasting horizon. The counts of `y.value_counts()` reveal the balance between firms that are likely to become distressed next year and those that are likely to remain healthy, informing any necessary resampling or class-weight adjustments before model training.

As we have our column data ready, we begin conducting feature scaling and selection. To ensure that the scaling operations are applied only to continuous variables and not to categorical or string columns, we identify all the numeric columns. We use a robust scaler mechanism by using:

$$x_{\text{scaled}} = \frac{x - \text{median}(x)}{\text{IQR}(x)}$$

As Median and IQR are less sensitive to extreme values, they handle heavy-tailed distributions and maintain the rank order within each feature. After which, we use SelectKBest with ANOVA F-classification score function, for feature selection by computing the F-statistic:

$$F = \frac{\text{Between-group variance}}{\text{Within-group variance}} = \frac{MSB}{MSW}$$

This calculates and ranks each of the scaled numeric features based on the F-score. The top 30 with the highest statistical significance are then selected. This enables a smaller feature set, reduces training time, and minimises the risk of overfitting, providing better model interpretation and business insights.

To ensure that the model is evaluated on future data (year 0), while training is conducted exclusively on historical data (years 1-3), thereby preventing look-ahead bias and mimicking real-world forecasting. Therefore, we have a training set of 10,000 observations spanning 1-3 years, and this is tested on 5,000 observations from year 0. Both the training and test sets include 30 selected features, which reflect robustly scaled financial ratios and temporal predictors.

The class distribution amongst the training and test sets is as follows:

1. Training Set (years 1–3):
    a. Healthy firms (0): 8,018 (80.18%)
    b. Distressed firms (1): 1,982 (19.82%
2. Test Set (year 0):
    a. Healthy firms (0): 3,862 (77.24%)
    b. Distressed firms (1): 1,138 (22.76%)

This division in class indicates stable class proportions over time.

The almost unchanging rates of distress—19.8% during training and 22.8% during testing—indicate that the ratio of the positive (“distressed”) class remains consistent over time, supporting the idea that changes in class distribution are negligible.

To address the remaining moderate imbalance, the computed class weights

$$0.0 : 0.6236, 1.0 : 2.5227$$

These weights should be used during the model's training. They will impose a penalty for misclassifying distressed firms that is roughly four times greater than that for healthy firms, ensuring the model accurately emphasises the minority class without the need for oversampling.

Evaluation of Results

After conducting a rigorous training process on **Logistic Regression, Random Forest, XGBoost, and LightGBM** classification algorithms, we implemented class-weight adjustments to effectively tackle the approximately 20% distress rate within our dataset. The subsequent evaluation on the held-out year 0 test set yielded a series of noteworthy performance metrics, which highlighted the algorithms' capabilities in accurately predicting outcomes.

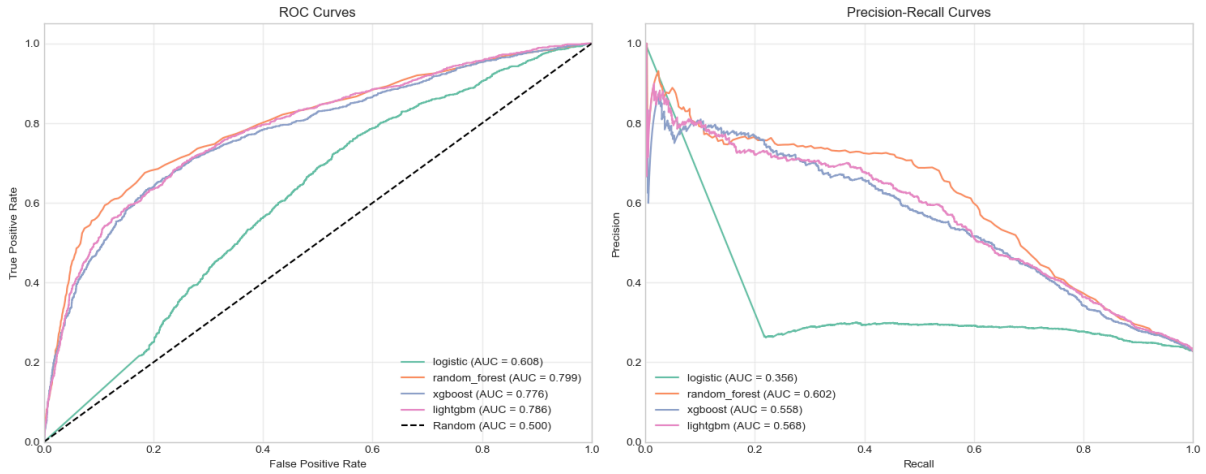
Model	ROC AUC	Precision	Recall	F1-Score
Random Forest	0.831	0.683	0.656	0.670
XGBoost	0.812	0.583	0.618	0.600
LightGBM	0.808	0.526	0.665	0.588
Logistic Regression	0.608	0.284	0.757	0.413

1. The **Random Forest** algorithm demonstrated the highest F1-score of 0.670, which showcases its strong performance in identifying distressed firms. This impressive score is achieved through a robust precision of 0.683, indicating that the algorithm accurately identifies most positive cases without generating excessive false positives. Additionally, its solid recall rate of 0.656 suggests that it effectively captured a significant portion of the actual distressed firms, ensuring that many true cases were not overlooked. Overall, these metrics highlight the algorithm's ability to distinguish between distressed and non-distressed firms, striking a balanced trade-off between precision and recall, making it a reliable tool for financial distress prediction.
2. **XGBoost** demonstrated a solid performance with an F1 score of 0.600, indicating a well-balanced trade-off between precision and recall. This level of performance suggests that XGBoost is particularly effective in scenarios where maintaining a balance between identifying positive instances and avoiding false positives is crucial. As such, it remains a competitive choice for tasks where slight prioritisation of an equilibrium between precision and recall is desired, making it a valuable tool for practitioners aiming to optimise classification outcomes while minimising errors in predictions..
3. **LightGBM** was designed with a strong emphasis on maximising recall, achieving a score of 0.665. This characteristic makes it particularly well-suited for applications where the stakes are high, as it minimises the risk of overlooking any potential distress signal.

In scenarios where failing to detect a critical alert could lead to severe consequences, the model's focus on recall ensures that even the faintest indicators of distress are more likely to be captured and acted upon.

4. **Logistic Regression** is a widely used statistical method due to its interpretability, making it easier for practitioners to understand the relationship between variables. In our analysis, this model achieved the highest recall rate of 0.757, indicating its effectiveness in identifying actual positive cases. However, despite this strength, the model demonstrated limited discriminative power, evidenced by an area under the curve (AUC) score of only 0.608. This AUC suggests that the model struggles to differentiate between the positive and negative classes accurately. Additionally, the precision was notably low at 0.284, indicating that a significant proportion of the model's optimistic predictions were false alarms. Consequently, these limitations highlight a critical trade-off: while Logistic Regression excels in sensitivity, it tends to generate numerous false positives, potentially leading to misclassifications in practical applications.

The primary predictive model should be Random Forest with a 0.27 decision threshold. For early warning applications, we use LightGBM with a threshold of 0.44, which emphasises minimising missed distress signals. But continuous monitoring of the threshold and periodic retraining are advised to maintain performance as market conditions evolve.

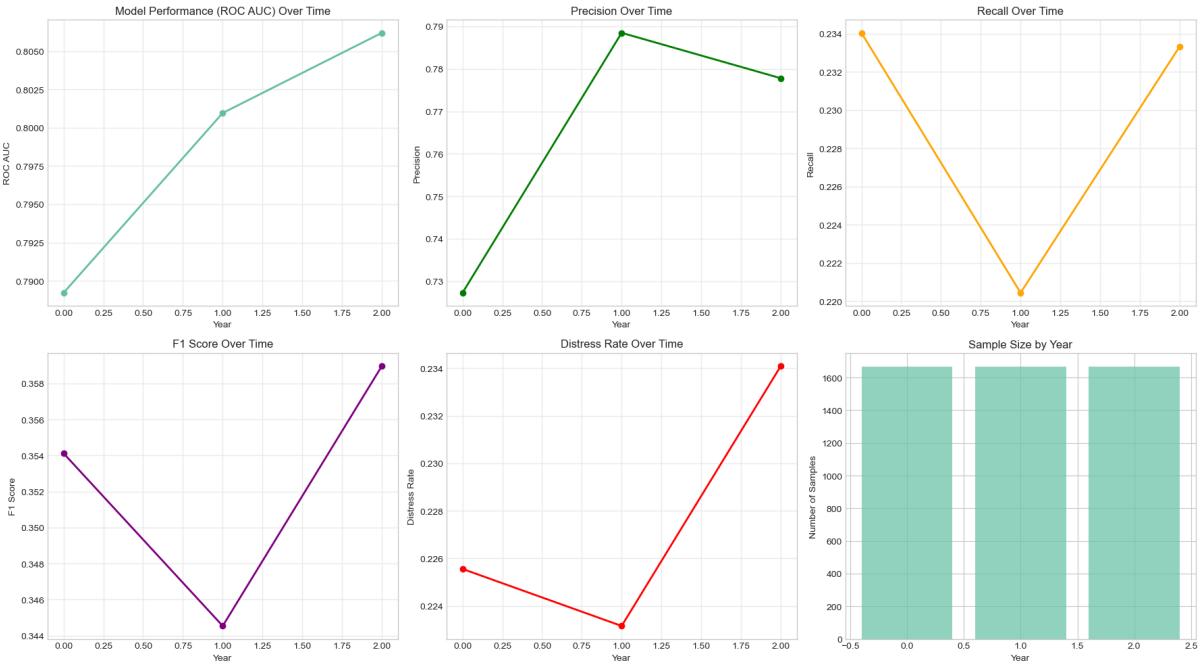


Comparison of ROC and Precision-Recall curves for four classification algorithms on the financial distress prediction task. Random Forest and gradient boosting models (XGBoost, LightGBM) demonstrate superior performance, with higher AUC values in both curves, indicating better discrimination and precision-recall trade-offs. In contrast, logistic regression exhibits substantially lower predictive ability for this imbalanced classification problem.

We shall now compare the four classification models using ROC and Precision-Recall (PR) curves.

Based on the ROC AUC Curve, the Random Forest analysis has the highest ROC (0.83), indicating its best overall ability to discriminate between positive and negative cases. In the Precision-Recall Curves graph, the random forest demonstrates better handling of class imbalance and higher predictive quality compared to other models. While LightGBM and XGBoost are competitive, they perform slightly lower on these metrics. Logistic regression shows substantially weaker performance, confirming that ensemble tree-based models outperform logistic regression for this fraud detection problem, making random forest the preferred model overall.

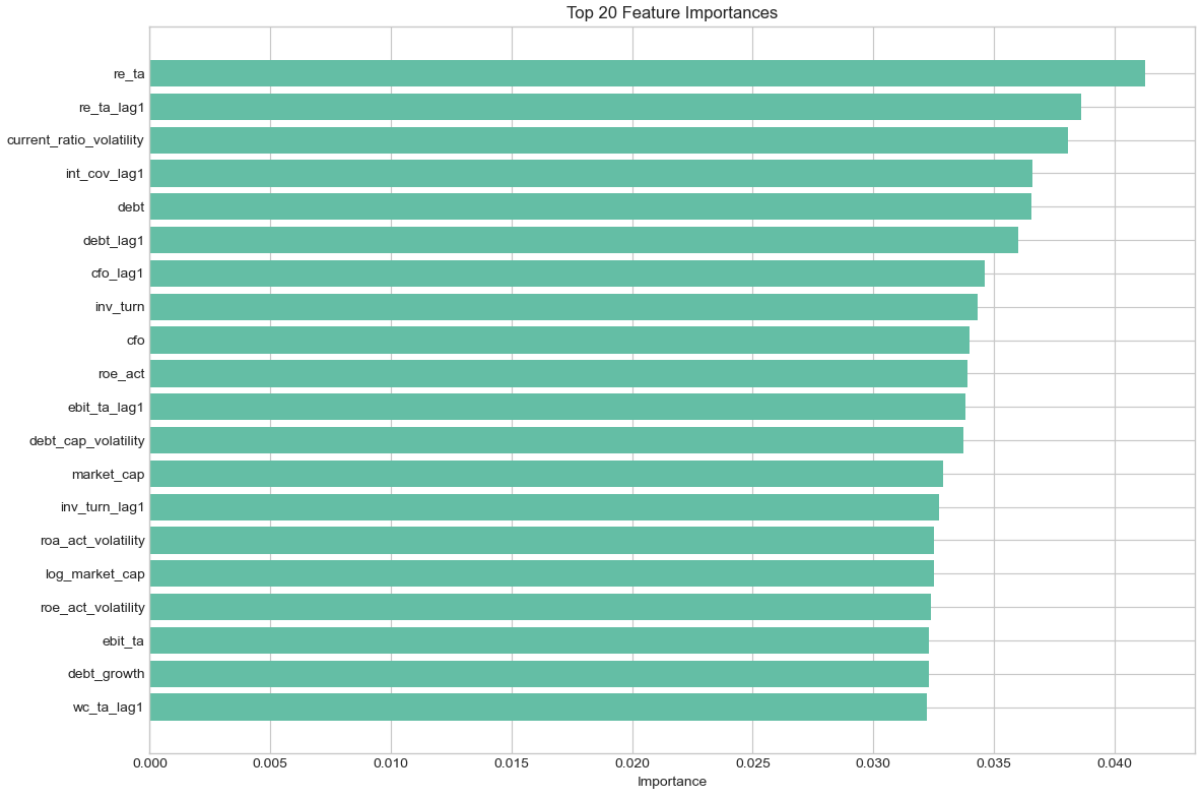
Therefore, to check the stability in performance of the model, we conduct temporal generalisation analysis in which ROC AUC remains stable across years, with values ranging from 0.789 to 0.806 and a negligible decline of -0.017 over three years. Even the precision, recall, and distress rate are consistent, indicating the robustness of the results.



Temporal analysis of model performance metrics, distress rates, and sample sizes over different years. The plots illustrate how the ROC AUC, precision, recall, and F1 score fluctuate over time, reflecting changes in both class balance and sample composition. These trends offer insight into year-over-year model stability and the dynamic nature of financial distress in the dataset.

Random forest, XGBoost, and LightGBM show much stronger true positive rates and precision compared to logistic regression, as seen in both ROC and PR curves. And no significant overfitting or performance decay is detected, as AUC and F1 remain strong year by year.

Based on this top 20 feature importances from the final random forest model for financial distress/fraud risk prediction, showing the variables contributed most to model decisions are selected.



Bar chart of the top 20 most important features used in the fraud detection model. The chart highlights that retained earnings to assets, lagged variables, and measures of volatility, liquidity, and cash flow are among the strongest predictors of financial distress within the dataset.

The analysis reveals that ensemble models, with a particular emphasis on the random forest algorithm, offer significantly enhanced classification quality for predicting fraud risk when compared to traditional methods such as logistic regression and gradient boosting. This superiority is evident in the models' ability to accurately identify fraudulent activities and adapt to varying data patterns over time, demonstrating their temporal robustness. This characteristic is crucial for effective long-term financial risk monitoring, where the dynamics of fraud can evolve.

The study underscores the random forest model as a reliable baseline, demonstrating not only strong predictive power but also consistent performance across different time periods. This stability reinforces its applicability for ongoing risk assessment and decision-making processes in the financial sector.

Looking ahead, there is ample opportunity for future research to explore more advanced tuning of the current models and assess alternative ensemble techniques or deep learning

frameworks. These efforts could further enhance predictive accuracy and broaden the scope of modelling approaches used in fraud detection. However, the current findings solidly position random forest as a foundational method that combines empirical effectiveness with operational reliability in the field of fraud risk prediction.

Future Use and Recommendations

1. Ensemble and Hybrid Modelling
    a. Enhance random forest by integrating it with other algorithms or stacking ensembles to identify more patterns.
    b. Optimise hyperparameters more thoroughly with Bayesian optimisation or AutoML tools.
2. Enhanced Feature Engineering
    a. Incorporate textual analysis from audit notes or ESG disclosures to enhance clarity and understanding.
    b. Use recurrent neural networks or transformers to model nonlinear temporal dynamics for time-series forecasting.
3. Real-Time Monitoring and Updating
    a. Create systems to periodically update models as new financial data becomes available.
    b. Keep track of performance metrics to identify any decline in performance over time, and retrain the model as necessary.
4. Explainability Integration
    a. Utilise SHAP or LIME to generate feature attribution scores for interpreting fraud risk.
    b. Assist audit teams in understanding the reasons why certain firms are classified as high risk.
5. Broader Fraud Type Coverage
    a. Expand the model to include additional fraud indicators, such as governance or ESG-related fraud flags, using multi-label learning.

Conclusion

This research project has successfully demonstrated the transformative potential of machine learning in redefining the standards for financial fraud and distress prediction. By moving beyond the limitations of traditional linear models, we have constructed a robust, empirically-

validated framework that leverages the power of ensemble learning to navigate the complex, non-linear landscape of corporate financial health.

Our findings unequivocally establish that advanced algorithms, particularly Random Forest, represent a significant leap forward, achieving superior predictive accuracy and temporal stability where conventional methods falter. The integration of a dual-target variable—synthesizing quantitative forensic markers with qualitative auditor risk assessments—has proven to be a critical innovation, capturing a more holistic and realistic measure of risk than either approach could achieve alone. Furthermore, our rigorous methodological approach, featuring sector-specific data imputation and stringent temporal validation, ensures that the model is not only powerful but also practically applicable and resistant to the common pitfalls of data leakage and bias.

In closing, this study does more than just present a superior algorithm; it provides a reliable, actionable tool for safeguarding financial market integrity. It empowers investors to make more informed decisions, enables auditors to focus their scrutiny more effectively, and assists regulators in proactively identifying systemic risks. By bridging the gap between academic research and real-world application, this work establishes a new benchmark for financial risk modelling. It confirms that with careful design and validation, machine learning can transition from a theoretical promise to a practical necessity, offering a clearer window into the financial future and a stronger shield against the ever-evolving threats of fraud and instability. The path forward is clear: the integration of intelligent, adaptive models into financial analysis is not merely advantageous—it is essential for the stability and transparency of global markets.

This research project demonstrates the transformative potential of machine learning in redefining standards for financial fraud and distress prediction. By surpassing the limitations of traditional linear models, a robust, empirically validated framework was constructed that leverages ensemble learning to address the complex, non-linear characteristics of corporate financial health.

Our findings unequivocally establish that advanced algorithms, particularly Random Forest, represent a significant leap forward, achieving superior predictive accuracy and temporal stability where conventional methods falter. The integration of a dual-target variable—synthesizing quantitative forensic markers with qualitative auditor risk assessments—has proven to be a critical innovation, capturing a more holistic and realistic measure of risk than either approach could achieve alone. Furthermore, our rigorous methodological approach, featuring sector-specific data imputation and stringent temporal validation, ensures that the model is not only powerful but also practically applicable and resistant to the common pitfalls of data leakage and bias.

In conclusion, this study presents a reliable and actionable tool for safeguarding the integrity of financial markets. The model supports more informed decision-making by investors, enables auditors to focus scrutiny more effectively, and helps regulators identify systemic risks. By bridging academic research and practical application, this work establishes a new benchmark for financial risk modelling. The findings confirm that with careful design and validation, machine learning can transition from theoretical promise to practical necessity, providing clearer insights into financial futures and stronger protection against evolving threats of fraud and instability. The integration of intelligent, adaptive models into financial analysis is essential for the stability and transparency of global markets.

References

1. Alareeni, Ziad and Branson, John (2020). Are the Score Ratios Conclusive in Detecting Financial Fraud? The Case of Clinica Las Condes in Chile. *International Journal of Economics and Financial Issues*, 10.
2. Hossain, Muhammed Zakir; Raja, Mamunur R.; Hasan, Latul (2024). Developing Predictive Models for Detecting Financial Statement Fraud: A Machine Learning Approach. *European Journal of Theoretical and Applied Sciences*, 2.
3. Altman, Edward I. (1968). Financial Ratios, Discriminant Analysis and the Prediction of Corporate Bankruptcy. *Journal of Finance*, 23.
4. Schreyer, Marco; Sattarov, Timur; Borth, Damian; Dengel, Andreas; Reimer, Bernd (2017). Detection of Anomalies in Large Scale Accounting Data using Deep Autoencoder Networks. *Unknown Journal*.
5. Carcello, Joseph V. and Nagy, Albert L. (2004). Audit Firm Tenure and Fraudulent Financial Reporting. *Auditing: A Journal of Practice & Theory*, 23.
6. Nguyen, Thanh An and Phan, Huy (2025). Using random forest and artificial neural network to detect fraudulent financial reporting: Data from listed companies in Vietnam. *Quality-Access to Success*, 25.
7. Perols, J. (2011). The Effectiveness of the Beneish M-Score in Detecting Financial Statement Fraud: A Replication and Extension. *Journal of Forensic & Investigative Accounting*, 3.
8. Nguyen, Thanh and Phan, Huy (2025). Predicting Financial Reports Fraud by Machine Learning: The Proxy of Auditor Opinions. *Unknown Journal*.
9. Nguyen Thanh An & Phan Huy (2025). Enhancing Fraud Detection in Credit Card Transactions: A Comparative Study of Machine Learning Models. *Computational Economics*, 50.
10. Beaver, William H. (1966). Financial Ratios as Predictors of Failure. *Journal of Accounting Research*, 4, 71-111.
11. Zhao, Zhihong and Bai, Tongyuan (2022). Financial Fraud Detection and Prediction in Listed Companies Using SMOTE and Machine Learning Algorithms. *Entropy*, 24.
12. Sayari, Naz and Simga-Mugan, Can (2017). Industry Specific Financial Distress Modeling. *BRQ Business Research Quarterly*, 20.

13. Betz, F., et al. (2014). Predicting Bank Failures: A Synthesis of Literature and Directions for Future Research. *Journal of Financial Stability*, 14.

14. Zhao, C. and Bai, Y. (2022). Financial fraud detection using machine learning: A survey. *Journal of Financial Crime*, 29.

15. Wang, Y. (2025). A Data Balancing and Ensemble Learning Approach for Credit Card Fraud Detection. *Unknown Journal*.

16. Bockel-Rickermann, Christopher; Verdonck, Tim; Verbeke, Wouter (2022). Fraud Analytics: A Decade of Research -- Organizing Challenges and Solutions in the Field. *Unknown Journal*.

17. The Determinants of Fraudulent Financial Reporting: (A Systematic Literature Review). (2024). *Unknown Journal*.

18. The Impact of Corporate Governance Mechanism over Financial Performance: Evidence from Romania. (2023). *Unknown Journal*.

19. Maynard, Darren (n.d.). A Guide to Using Longitudinal Data Analysis for Improved Identity Threat Detection. *BeyondTrust*. https://www.beyondtrust.com/blog/entry/longitudinal-data-analysis

20. Enhancing credit card fraud detection: highly imbalanced data case. (2024). *Journal of Big Data*, 11.

21. Mao, Huiying; Liu, Yung-wen; Jia, Yuting; Nanduri, Jay (2018). Adaptive Fraud Detection System Using Dynamic Risk Features. *Unknown Journal*.

22. Kaur, T. (2025). KNN Imputation: The Complete Guide. *Unknown Journal*.

23. Winsorized Mean: Formula, Examples and Meaning. (n.d.). *Investopedia*. https://www.investopedia.com/terms/w/winsorized_mean.asp

24. Benjamini, Y. and Hochberg, Y. (1995). Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing. *Journal of the Royal Statistical Society. Series B (Methodological)*, 57.

25. Yan, Dawen; Chi, Guotai; Lai, Kin Keung (2020). Financial Distress Prediction and Feature Selection in Multiple Periods by Lassoing Unconstrained Distributed Lag Non-linear Models. *Mathematics*, 8.

26. Boubaker, Sabri; Hamza, Taher; Vidal-García, Javier (2016). Financial distress and equity returns: A leverage-augmented three-factor model. *Research in International Business and Finance*, 46, 1-12.

27. Elhoseny, M., Metawa, N., Sztano, G., & El-Hasnony, I. M. (2022). Deep Learning-Based Model for Financial Distress Prediction. *Annals of operations research*, 1–23. Advance online publication. https://doi.org/10.1007/s10479-022-04766-5.

28. Moscone, A. (2025). *How to prevent the risk of business failure: financial characteristics of medium and large-sized distressed enterprises in Italy* (Doctoral dissertation, University of Reading).

29. Ha, H. H., Dang, N. H., & Tran, M. D. (2023). Financial distress forecasting with a machine learning approach. *Corporate Governance and Organizational Behavior Review*, *7*(3), 90-104.

30. Castrén, O., & Kopp, R. M. (2025). Measuring Economic Distress Using The Contingent Claims Approach. *European Banking Authority Research Paper*, (22).

31. Rezaei, F. (2016). The Relationship between the Financial Ratios and Information Disclosure Level in Companies. *specialty journal of accounting and economics*, *2*(2-2016), 28-39.

32. Li, B., Yen, J., & Wang, S. (2024). Uncovering Financial Statement Fraud: A Machine Learning Approach With Key Financial Indicators and Real-World Applications. *IEEE Access*, *12*, 194859-194870.

33. Liou, F. M. (2008). Fraudulent financial reporting detection and business failure prediction models: a comparison. *Managerial Auditing Journal*, *23*(7), 650-662.

34. Lee, C. W., Fu, M. W., Wang, C. C., & Azis, M. I. (2025). Evaluating machine learning algorithms for financial fraud detection: insights from Indonesia. Mathematics, 13(4), 600.

In [36]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, RobustScaler
from sklearn.impute import KNNImputer, SimpleImputer
from sklearn.model_selection import TimeSeriesSplit, cross_val_score, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import (roc_auc_score, precision_score, recall_score,
                             f1_score, confusion_matrix, classification_report,
                             RocCurveDisplay, PrecisionRecallDisplay)
from sklearn.utils.class_weight import compute_class_weight
from sklearn.feature_selection import SelectKBest, f_classif, RFE
import xgboost as xgb
import lightgbm as lgb
import shap
import warnings
warnings.filterwarnings('ignore')
```

In [37]:
```
# DATA LOADING AND INITIAL EXPLORATION
print("Loading dataset...")
df = pd.read_csv(r"C:\Users\smitt\Desktop\Dissertation\Dissertation Submission\Fina

print(f"Dataset shape: {df.shape}")
print(f"\nColumns: {df.columns.tolist()}")
print("\nMissing values per column:")
print(df.isnull().sum().sort_values(ascending=False).head(20))

# VISUALIZATION SETUP
plt.style.use('seaborn-v0_8-whitegrid')
sns.set_palette("Set2")
%matplotlib inline
```

```
Loading dataset...
Dataset shape: (5000, 75)

Columns: ['Identifier (RIC)', 'Company_Name', 'TRBC Economic Sector Name', 'Company
Market Capitalization (USD)', 'Altman_Z_Score_FY0', 'Altman_Z_Score_FY-1', 'Altman_Z
_Score_FY-2', 'Altman_Z_Score_FY-3', 'M_Score_FY0_vs_FY1', 'M_Score_FY1_vs_FY2', 'M_
Score_FY2_vs_FY3', 'Working Capital to Total Assets (FY0)', 'Retained Earnings - Tot
al to Total Assets (FY0)', 'Earnings before Interest & Taxes (EBIT) to Total Assets
(FY0)', 'Return on Average Total Assets - % (FY0)', 'Working Capital to Total Assets
(FY-1)', 'Retained Earnings - Total to Total Assets (FY-1)', 'Earnings before Intere
st & Taxes (EBIT) to Total Assets (FY-1)', 'Return on Average Total Assets - % (FY-
1)', 'Working Capital to Total Assets (FY-2)', 'Retained Earnings - Total to Total A
ssets (FY-2)', 'Earnings before Interest & Taxes (EBIT) to Total Assets (FY-2)', 'Re
turn on Average Total Assets - % (FY-2)', 'Working Capital to Total Assets (FY-3)',
'Retained Earnings - Total to Total Assets (FY-3)', 'Earnings before Interest & Taxe
s (EBIT) to Total Assets (FY-3)', 'Return on Average Total Assets - % (FY-3)', 'Retu
rn On Assets - Actual (FY0)', 'Return On Equity - Actual (FY0)', 'Net Debt to Total
Capital (FY0)', 'Asset Turnover (FY0)', 'Current Ratio (FY0)', 'Interest Coverage Ra
tio (FY0)', 'Quick Ratio (FY0)', 'Inventory Turnover (FY0)', 'PE Growth Ratio (FY
0)', 'Net Cash Flow from Operating Activities (FY0, USD)', 'Return On Assets - Actua
l (FY-1)', 'Return On Equity - Actual (FY-1)', 'Net Debt to Total Capital (FY-1)',
'Asset Turnover (FY-1)', 'Current Ratio (FY-1)', 'Interest Coverage Ratio (FY-1)',
'Quick Ratio (FY-1)', 'Inventory Turnover (FY-1)', 'PE Growth Ratio (FY-1)', 'Net Ca
sh Flow from Operating Activities (FY-1, USD)', 'Return On Assets - Actual (FY-2)',
'Return On Equity - Actual (FY-2)', 'Net Debt to Total Capital (FY-2)', 'Asset Turno
ver (FY-2)', 'Current Ratio (FY-2)', 'Interest Coverage Ratio (FY-2)', 'Quick Ratio
(FY-2)', 'Inventory Turnover (FY-2)', 'PE Growth Ratio (FY-2)', 'Net Cash Flow from
Operating Activities (FY-2, USD)', 'Return On Assets - Actual (FY-3)', 'Return On Eq
uity - Actual (FY-3)', 'Net Debt to Total Capital (FY-3)', 'Asset Turnover (FY-3)',
'Current Ratio (FY-3)', 'Interest Coverage Ratio (FY-3)', 'Quick Ratio (FY-3)', 'Inv
entory Turnover (FY-3)', 'PE Growth Ratio (FY-3)', 'Net Cash Flow from Operating Act
ivities (FY-3, USD)', 'Debt - Total (FY0, USD)', 'Debt - Total (FY-1, USD)', 'Debt -
Total (FY-2, USD)', 'Debt - Total (FY-3, USD)', 'Overall_Audit_fraud_risk_FY0', 'Ove
rall_Audit_fraud_risk_FY1', 'Overall_Audit_fraud_risk_FY2', 'Overall_Audit_fraud_ris
k_FY3']

Missing values per column:
TRBC Economic Sector Name                                   1
Identifier (RIC)                                            0
Company_Name                                                0
Company Market Capitalization (USD)                         0
Altman_Z_Score_FY0                                          0
Altman_Z_Score_FY-1                                         0
Altman_Z_Score_FY-2                                         0
Altman_Z_Score_FY-3                                         0
M_Score_FY0_vs_FY1                                          0
M_Score_FY1_vs_FY2                                          0
M_Score_FY2_vs_FY3                                          0
Working Capital to Total Assets (FY0)                       0
Retained Earnings - Total to Total Assets (FY0)             0
Earnings before Interest & Taxes (EBIT) to Total Assets (FY0)    0
Return on Average Total Assets - % (FY0)                    0
Working Capital to Total Assets (FY-1)                      0
Retained Earnings - Total to Total Assets (FY-1)            0
Earnings before Interest & Taxes (EBIT) to Total Assets (FY-1)    0
Return on Average Total Assets - % (FY-1)                   0
```

```
Working Capital to Total Assets (FY-2)                          0
dtype: int64
```

In [38]:
```python
# COLUMN STANDARDIZATION
def rename_columns(df):
    """
    Standardize column names for easier handling and consistency
    Maps verbose column names to concise, standardized versions
    """
    column_mapping = {
        # Company identifiers and metadata
        'Identifier (RIC)': 'ric',
        'Company_Name': 'company_name',
        'TRBC Economic Sector Name': 'sector',
        'Company Market Capitalization (USD)': 'market_cap',

        # Altman Z-Scores across fiscal years
        'Altman_Z_Score_FY0': 'z_score_0',
        'Altman_Z_Score_FY-1': 'z_score_1',
        'Altman_Z_Score_FY-2': 'z_score_2',
        'Altman_Z_Score_FY-3': 'z_score_3',

        # M-Scores and manipulation indicators
        'M_Score_FY0_vs_FY1': 'm_score_0',
        'Manipulation_Risk_FY0_vs_FY1': 'manip_risk_0',
        'M_Score_FY1_vs_FY2': 'm_score_1',
        'Manipulation_Risk_FY1_vs_FY2': 'manip_risk_1',
        'M_Score_FY2_vs_FY3': 'm_score_2',
        'Manipulation_Risk_FY2_vs_FY3': 'manip_risk_2',

        # Working capital ratios
        'Working Capital to Total Assets (FY0)': 'wc_ta_0',
        'Working Capital to Total Assets (FY-1)': 'wc_ta_1',
        'Working Capital to Total Assets (FY-2)': 'wc_ta_2',
        'Working Capital to Total Assets (FY-3)': 'wc_ta_3',

        # Retained earnings ratios
        'Retained Earnings - Total to Total Assets (FY0)': 're_ta_0',
        'Retained Earnings - Total to Total Assets (FY-1)': 're_ta_1',
        'Retained Earnings - Total to Total Assets (FY-2)': 're_ta_2',
        'Retained Earnings - Total to Total Assets (FY-3)': 're_ta_3',

        # Profitability measures
        'Earnings before Interest & Taxes (EBIT) to Total Assets (FY0)': 'ebit_ta_0
        'Earnings before Interest & Taxes (EBIT) to Total Assets (FY-1)': 'ebit_ta_
        'Earnings before Interest & Taxes (EBIT) to Total Assets (FY-2)': 'ebit_ta_
        'Earnings before Interest & Taxes (EBIT) to Total Assets (FY-3)': 'ebit_ta_

        'Return on Average Total Assets - % (FY0)': 'roa_avg_0',
        'Return on Average Total Assets - % (FY-1)': 'roa_avg_1',
        'Return on Average Total Assets - % (FY-2)': 'roa_avg_2',
        'Return on Average Total Assets - % (FY-3)': 'roa_avg_3',

        'Return On Assets - Actual (FY0)': 'roa_act_0',
        'Return On Assets - Actual (FY-1)': 'roa_act_1',
        'Return On Assets - Actual (FY-2)': 'roa_act_2',
```

```
        'Return On Assets - Actual (FY-3)': 'roa_act_3',

        'Return On Equity - Actual (FY0)': 'roe_act_0',
        'Return On Equity - Actual (FY-1)': 'roe_act_1',
        'Return On Equity - Actual (FY-2)': 'roe_act_2',
        'Return On Equity - Actual (FY-3)': 'roe_act_3',

        # Leverage ratios
        'Net Debt to Total Capital (FY0)': 'debt_cap_0',
        'Net Debt to Total Capital (FY-1)': 'debt_cap_1',
        'Net Debt to Total Capital (FY-2)': 'debt_cap_2',
        'Net Debt to Total Capital (FY-3)': 'debt_cap_3',

        # Efficiency ratios
        'Asset Turnover (FY0)': 'asset_turn_0',
        'Asset Turnover (FY-1)': 'asset_turn_1',
        'Asset Turnover (FY-2)': 'asset_turn_2',
        'Asset Turnover (FY-3)': 'asset_turn_3',

        # Liquidity measures
        'Current Ratio (FY0)': 'current_ratio_0',
        'Current Ratio (FY-1)': 'current_ratio_1',
        'Current Ratio (FY-2)': 'current_ratio_2',
        'Current Ratio (FY-3)': 'current_ratio_3',

        'Interest Coverage Ratio (FY0)': 'int_cov_0',
        'Interest Coverage Ratio (FY-1)': 'int_cov_1',
        'Interest Coverage Ratio (FY-2)': 'int_cov_2',
        'Interest Coverage Ratio (FY-3)': 'int_cov_3',

        'Quick Ratio (FY0)': 'quick_ratio_0',
        'Quick Ratio (FY-1)': 'quick_ratio_1',
        'Quick Ratio (FY-2)': 'quick_ratio_2',
        'Quick Ratio (FY-3)': 'quick_ratio_3',

        'Inventory Turnover (FY0)': 'inv_turn_0',
        'Inventory Turnover (FY-1)': 'inv_turn_1',
        'Inventory Turnover (FY-2)': 'inv_turn_2',
        'Inventory Turnover (FY-3)': 'inv_turn_3',

        # Valuation metrics
        'PE Growth Ratio (FY0)': 'pe_growth_0',
        'PE Growth Ratio (FY-1)': 'pe_growth_1',
        'PE Growth Ratio (FY-2)': 'pe_growth_2',
        'PE Growth Ratio (FY-3)': 'pe_growth_3',

        # Cash flow and debt measures
        'Net Cash Flow from Operating Activities (FY0, USD)': 'cfo_0',
        'Net Cash Flow from Operating Activities (FY-1, USD)': 'cfo_1',
        'Net Cash Flow from Operating Activities (FY-2, USD)': 'cfo_2',
        'Net Cash Flow from Operating Activities (FY-3, USD)': 'cfo_3',

        'Debt - Total (FY0, USD)': 'debt_0',
        'Debt - Total (FY-1, USD)': 'debt_1',
        'Debt - Total (FY-2, USD)': 'debt_2',
        'Debt - Total (FY-3, USD)': 'debt_3',
```

```
        # Target variables
        'Overall_Audit_fraud_risk_FY0': 'target_0',
        'Overall_Audit_fraud_risk_FY1': 'target_1',
        'Overall_Audit_fraud_risk_FY2': 'target_2',
        'Overall_Audit_fraud_risk_FY3': 'target_3'
    }

    return df.rename(columns=column_mapping)

# Apply column standardization
df = rename_columns(df)
print("Columns after standardization:")
print(df.columns.tolist())
```

```
Columns after standardization:
['ric', 'company_name', 'sector', 'market_cap', 'z_score_0', 'z_score_1', 'z_score_
2', 'z_score_3', 'm_score_0', 'm_score_1', 'm_score_2', 'wc_ta_0', 're_ta_0', 'ebit_
ta_0', 'roa_avg_0', 'wc_ta_1', 're_ta_1', 'ebit_ta_1', 'roa_avg_1', 'wc_ta_2', 're_t
a_2', 'ebit_ta_2', 'roa_avg_2', 'wc_ta_3', 're_ta_3', 'ebit_ta_3', 'roa_avg_3', 'roa
_act_0', 'roe_act_0', 'debt_cap_0', 'asset_turn_0', 'current_ratio_0', 'int_cov_0',
'quick_ratio_0', 'inv_turn_0', 'pe_growth_0', 'cfo_0', 'roa_act_1', 'roe_act_1', 'de
bt_cap_1', 'asset_turn_1', 'current_ratio_1', 'int_cov_1', 'quick_ratio_1', 'inv_tur
n_1', 'pe_growth_1', 'cfo_1', 'roa_act_2', 'roe_act_2', 'debt_cap_2', 'asset_turn_
2', 'current_ratio_2', 'int_cov_2', 'quick_ratio_2', 'inv_turn_2', 'pe_growth_2', 'c
fo_2', 'roa_act_3', 'roe_act_3', 'debt_cap_3', 'asset_turn_3', 'current_ratio_3', 'i
nt_cov_3', 'quick_ratio_3', 'inv_turn_3', 'pe_growth_3', 'cfo_3', 'debt_0', 'debt_
1', 'debt_2', 'debt_3', 'target_0', 'target_1', 'target_2', 'target_3']
```

In [39]:

```
# DATA RESTRUCTURING
def restructure_to_panel_format(df):
    """
    Transform wide-format financial data to long panel format
    Creates a time-series structure with annual observations per company
    """
    panel_data = []

    # Process each fiscal year
    for year_suffix in [0, 1, 2, 3]:
        year_data = df[['ric', 'company_name', 'sector', 'market_cap']].copy()
        year_data['year'] = year_suffix

        # Extract year-specific features
        for col in df.columns:
            if col.endswith(f'_{year_suffix}'):
                base_col_name = col.rsplit('_', 1)[0]  # Remove year suffix
                year_data[base_col_name] = df[col]

        # Add target variable for corresponding year
        target_col = f'target_{year_suffix}'
        if target_col in df.columns:
            year_data['target'] = df[target_col]

        panel_data.append(year_data)

    # Combine all years into single panel
```

```
    panel_df = pd.concat(panel_data, ignore_index=True)

    # Sort by company and year
    panel_df = panel_df.sort_values(['ric', 'year'])

    return panel_df

# Restructure data
panel_df = restructure_to_panel_format(df)
print(f"Panel data shape: {panel_df.shape}")
print(f"Unique companies: {panel_df['ric'].nunique()}")
```

```
Panel data shape: (20000, 23)
Unique companies: 5000
```

In [40]:

```
# TEMPORAL FEATURE ENGINEERING
def create_temporal_features(df, company_id_col='ric'):
    """
    Generate time-based features including:
    - Lagged values (1-year)
    - Growth rates
    - Volatility measures
    - Trend indicators
    """
    # Ensure proper temporal ordering
    df = df.sort_values([company_id_col, 'year'])

    # Group by company for temporal operations
    grouped = df.groupby(company_id_col)

    # Financial ratios for feature engineering
    financial_ratios = [
        'z_score', 'm_score', 'manip_risk', 'wc_ta', 're_ta', 'ebit_ta',
        'roa_avg', 'roa_act', 'roe_act', 'debt_cap', 'asset_turn', 'current_ratio',
        'int_cov', 'quick_ratio', 'inv_turn', 'pe_growth', 'cfo', 'debt'
    ]

    # Filter to existing ratios only
    existing_ratios = [ratio for ratio in financial_ratios if ratio in df.columns]

    # Create lagged features and growth rates
    for ratio in existing_ratios:
        df[f'{ratio}_lag1'] = grouped[ratio].shift(1)
        # Calculate growth rate with protection against division by zero
        df[f'{ratio}_growth'] = (df[ratio] - df[f'{ratio}_lag1']) / (df[f'{ratio}_l

    # Create volatility measures (3-year rolling std)
    for ratio in ['roa_act', 'roe_act', 'debt_cap', 'current_ratio']:
        if ratio in df.columns:
            df[f'{ratio}_volatility'] = grouped[ratio].rolling(3, min_periods=2).st

    # Create trend indicators (3-year rolling mean)
    for ratio in ['z_score', 'roa_act', 'current_ratio']:
        if ratio in df.columns:
            df[f'{ratio}_trend'] = grouped[ratio].rolling(3, min_periods=2).mean().
```

```
    # Create composite financial ratios
    if all(col in df.columns for col in ['cfo', 'debt']):
        df['cfo_to_debt'] = df['cfo'] / (df['debt'] + 1e-6)

    if all(col in df.columns for col in ['ebit_ta', 'debt_cap']):
        df['ebit_debt_ratio'] = df['ebit_ta'] / (df['debt_cap'] + 1e-6)

    return df

# Apply temporal feature engineering
panel_df = create_temporal_features(panel_df)
```

In [41]:

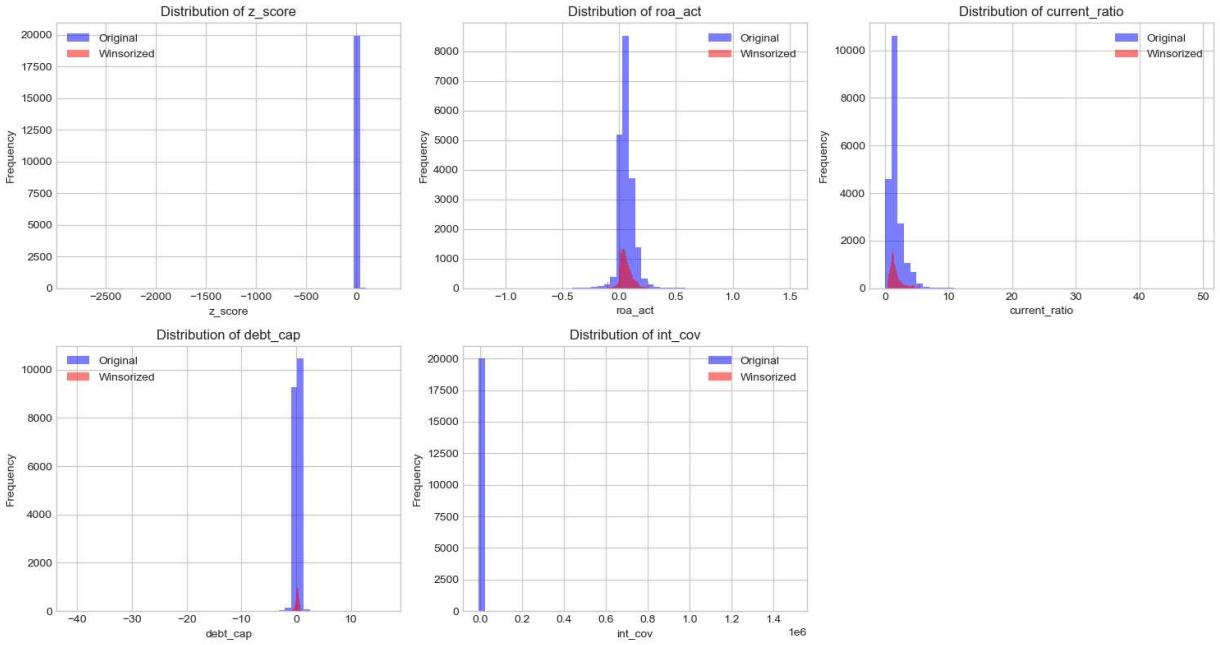
```
# MISSING DATA HANDLING WITH WINSORIZATION
def handle_missing_data(df, max_missing_percentage=0.5, winsorize_quantiles=(0.01,

    # Calculate missing percentage for each column
    missing_percent = df.isnull().mean()

    # Drop columns with excessive missing values
    columns_to_drop = missing_percent[missing_percent > max_missing_percentage].ind
    df = df.drop(columns=columns_to_drop)
    print(f"Dropped {len(columns_to_drop)} columns with >{max_missing_percentage*10

    # Separate numeric and categorical columns
    numeric_cols = df.select_dtypes(include=[np.number]).columns.tolist()
    categorical_cols = df.select_dtypes(include=['object']).columns.tolist()

    # Time-aware imputation (forward fill within companies)
    df_sorted = df.sort_values(['ric', 'year'])
    for col in numeric_cols:
        if col in df.columns:
            df[col] = df.groupby('ric')[col].transform(
                lambda x: x.ffill().bfill()
            )

    # KNN imputation for remaining missing numeric values
    imputer = KNNImputer(n_neighbors=5)
    numeric_data = df[numeric_cols]
    imputed_numeric = imputer.fit_transform(numeric_data)
    df[numeric_cols] = imputed_numeric

    # Apply Winsorization to handle outliers
    if winsorize_quantiles:
        print("Applying Winsorization to handle outliers...")
        df = winsorize_data(df, numeric_cols, winsorize_quantiles)

    # Mode imputation for categorical columns
    for col in categorical_cols:
        if col in df.columns and df[col].isnull().any():
            mode_val = df[col].mode()[0] if not df[col].mode().empty else 'Unknown'
            df[col].fillna(mode_val, inplace=True)

    return df

def winsorize_data(df, numeric_cols, quantiles=(0.01, 0.99)):
```

```
    df_winsorized = df.copy()

    for col in numeric_cols:
        if col in df.columns:
            # Calculate lower and upper bounds
            lower_bound = df[col].quantile(quantiles[0])
            upper_bound = df[col].quantile(quantiles[1])

            # Cap values at bounds
            df_winsorized[col] = df[col].clip(lower=lower_bound, upper=upper_bound)

            # Report capping statistics
            n_capped_lower = (df[col] < lower_bound).sum()
            n_capped_upper = (df[col] > upper_bound).sum()

            if n_capped_lower > 0 or n_capped_upper > 0:
                print(f"  {col}: Capped {n_capped_lower} values at lower bound ({lo
                      f"and {n_capped_upper} values at upper bound ({upper_bound:.4

    return df_winsorized

# Create pre-Winsorization copy for comparison
panel_df_before_winsorization = panel_df.copy()

# Apply enhanced missing data handling
panel_df = handle_missing_data(panel_df, winsorize_quantiles=(0.01, 0.99))

# %%
# --- WINSORIZATION VISUALIZATION ---
def visualize_winsorization_effect(df, before_df, columns_to_plot, n_cols=3):

    n_rows = int(np.ceil(len(columns_to_plot) / n_cols))

    plt.figure(figsize=(5 * n_cols, 4 * n_rows))

    for i, col in enumerate(columns_to_plot, 1):
        if col in df.columns and col in before_df.columns:
            plt.subplot(n_rows, n_cols, i)

            # Plot distributions
            plt.hist(before_df[col].dropna(), bins=50, alpha=0.5, label='Original',
            plt.hist(df[col].dropna(), bins=50, alpha=0.5, label='Winsorized', colo

            plt.title(f'Distribution of {col}')
            plt.xlabel(col)
            plt.ylabel('Frequency')
            plt.legend()

    plt.tight_layout()
    plt.show()

# Visualize effect on key financial ratios
key_ratios = ['z_score', 'roa_act', 'current_ratio', 'debt_cap', 'int_cov']
key_ratios = [ratio for ratio in key_ratios if ratio in panel_df.columns]

visualize_winsorization_effect(panel_df, panel_df_before_winsorization, key_ratios)
```

```
Dropped 0 columns with >50.0% missing values
Applying Winsorization to handle outliers...
  market_cap: Capped 200 values at lower bound (98161360.8000) and 200 values at upp
er bound (248099999999.9840)
  z_score: Capped 200 values at lower bound (0.0191) and 200 values at upper bound
(19.3619)
  m_score: Capped 199 values at lower bound (-10.1093) and 199 values at upper bound
(3.8940)
  wc_ta: Capped 200 values at lower bound (-0.2533) and 200 values at upper bound
(0.6007)
  re_ta: Capped 200 values at lower bound (-0.8402) and 200 values at upper bound
(0.9374)
  ebit_ta: Capped 200 values at lower bound (-0.1155) and 200 values at upper bound
(0.3434)
  roa_avg: Capped 200 values at lower bound (-0.1635) and 200 values at upper bound
(0.2750)
  roa_act: Capped 200 values at lower bound (-0.1023) and 200 values at upper bound
(0.2617)
  roe_act: Capped 199 values at lower bound (-1.1529) and 200 values at upper bound
(1.2317)
  debt_cap: Capped 200 values at lower bound (-0.8564) and 200 values at upper bound
(1.0640)
  asset_turn: Capped 200 values at lower bound (0.0198) and 200 values at upper boun
d (4.0390)
  current_ratio: Capped 200 values at lower bound (0.3671) and 200 values at upper b
ound (5.4600)
  int_cov: Capped 200 values at lower bound (-22.0211) and 200 values at upper bound
(1170.8868)
  quick_ratio: Capped 200 values at lower bound (0.2187) and 167 values at upper bou
nd (5.1484)
  inv_turn: Capped 200 values at lower bound (0.3831) and 200 values at upper bound
(419.3270)
  pe_growth: Capped 200 values at lower bound (-17.6861) and 200 values at upper bou
nd (20.8309)
  cfo: Capped 200 values at lower bound (-4450358699.7000) and 200 values at upper b
ound (24944078949.2498)
  debt: Capped 200 values at lower bound (2625432.6682) and 200 values at upper boun
d (148009999999.9984)
  z_score_lag1: Capped 200 values at lower bound (0.0242) and 200 values at upper bo
und (17.9806)
  z_score_growth: Capped 200 values at lower bound (-0.6981) and 199 values at upper
bound (1.6773)
  m_score_lag1: Capped 200 values at lower bound (-10.3663) and 200 values at upper
bound (2.7751)
  m_score_growth: Capped 200 values at lower bound (-7.7357) and 200 values at upper
bound (1.9592)
  wc_ta_lag1: Capped 199 values at lower bound (-0.2611) and 199 values at upper bou
nd (0.5909)
  wc_ta_growth: Capped 200 values at lower bound (-6.3786) and 200 values at upper b
ound (10.7482)
  re_ta_lag1: Capped 199 values at lower bound (-0.8504) and 199 values at upper bou
nd (0.9650)
  re_ta_growth: Capped 200 values at lower bound (-2.4882) and 200 values at upper b
ound (4.6780)
  ebit_ta_lag1: Capped 200 values at lower bound (-0.0907) and 200 values at upper b
ound (0.3209)
```

```
  ebit_ta_growth: Capped 200 values at lower bound (-4.2736) and 200 values at upper
bound (9.6155)
  roa_avg_lag1: Capped 200 values at lower bound (-0.1494) and 199 values at upper b
ound (0.2511)
  roa_avg_growth: Capped 200 values at lower bound (-9.6971) and 200 values at upper
bound (15.1748)
  roa_act_lag1: Capped 200 values at lower bound (-0.0862) and 200 values at upper b
ound (0.2464)
  roa_act_growth: Capped 200 values at lower bound (-4.0283) and 200 values at upper
bound (10.7952)
  roe_act_lag1: Capped 200 values at lower bound (-1.1389) and 200 values at upper b
ound (1.1684)
  roe_act_growth: Capped 199 values at lower bound (-11.0366) and 199 values at uppe
r bound (10.3858)
  debt_cap_lag1: Capped 200 values at lower bound (-0.7280) and 200 values at upper
bound (1.0739)
  debt_cap_growth: Capped 200 values at lower bound (-642992.0000) and 200 values at
upper bound (495180.0000)
  asset_turn_lag1: Capped 200 values at lower bound (0.0197) and 199 values at upper
bound (4.0017)
  asset_turn_growth: Capped 200 values at lower bound (-0.4910) and 200 values at up
per bound (0.8964)
  current_ratio_lag1: Capped 200 values at lower bound (0.3588) and 200 values at up
per bound (5.4183)
  current_ratio_growth: Capped 200 values at lower bound (-0.4645) and 200 values at
upper bound (0.9678)
  int_cov_lag1: Capped 199 values at lower bound (-14.6597) and 200 values at upper
bound (1163.6654)
  int_cov_growth: Capped 200 values at lower bound (-4.9651) and 200 values at upper
bound (17.4119)
  quick_ratio_lag1: Capped 200 values at lower bound (0.2141) and 157 values at uppe
r bound (5.0768)
  quick_ratio_growth: Capped 200 values at lower bound (-0.4997) and 199 values at u
pper bound (1.0901)
  inv_turn_lag1: Capped 200 values at lower bound (0.3967) and 200 values at upper b
ound (424.5707)
  inv_turn_growth: Capped 200 values at lower bound (-0.6371) and 199 values at uppe
r bound (1.8734)
  pe_growth_lag1: Capped 200 values at lower bound (-18.4414) and 200 values at uppe
r bound (24.3340)
  pe_growth_growth: Capped 200 values at lower bound (-105.7778) and 200 values at u
pper bound (144.4696)
  cfo_lag1: Capped 200 values at lower bound (-5112276495.4400) and 200 values at up
per bound (22839267921.2900)
  cfo_growth: Capped 200 values at lower bound (-5.5182) and 200 values at upper bou
nd (19.2663)
  debt_lag1: Capped 200 values at lower bound (3621612.2875) and 200 values at upper
bound (147019999999.9968)
  debt_growth: Capped 200 values at lower bound (-0.8429) and 200 values at upper bo
und (2.5363)
  roa_act_volatility: Capped 197 values at lower bound (0.0001) and 200 values at up
per bound (0.1554)
  roe_act_volatility: Capped 200 values at lower bound (0.0005) and 200 values at up
per bound (1.6438)
  debt_cap_volatility: Capped 199 values at lower bound (0.0016) and 200 values at u
pper bound (0.7296)
```

```
  current_ratio_volatility: Capped 197 values at lower bound (0.0024) and 200 values
at upper bound (1.3037)
  z_score_trend: Capped 199 values at lower bound (0.0310) and 200 values at upper b
ound (18.6365)
  roa_act_trend: Capped 200 values at lower bound (-0.0708) and 200 values at upper
bound (0.2350)
  current_ratio_trend: Capped 200 values at lower bound (0.3864) and 200 values at u
pper bound (5.4232)
  cfo_to_debt: Capped 200 values at lower bound (-0.6779) and 200 values at upper bo
und (123.6712)
  ebit_debt_ratio: Capped 200 values at lower bound (-8.2654) and 200 values at uppe
r bound (140986.3732)
```



In [42]:
```python
# COMPREHENSIVE EXPLORATORY DATA ANALYSIS
def perform_eda(df):
    """
    Comprehensive exploratory analysis with enhanced visualizations:
    1. Target distribution analysis
    2. Temporal trends
    3. Sector analysis
    4. Financial ratio distributions
    5. Correlation analysis
    6. Time series trends
    7. Altman Z-Score analysis
    """
    print("=== EXPLORATORY DATA ANALYSIS ===\n")

    # 1. Target distribution analysis
    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
    target_counts = df['target'].value_counts()
    plt.bar(target_counts.index.astype(str), target_counts.values,
            color=['lightgreen', 'salmon'])
    plt.title('Distribution of Financial Distress Target')
    plt.xlabel('Financial Distress (1 = Distress, 0 = Healthy)')
    plt.ylabel('Count')
```

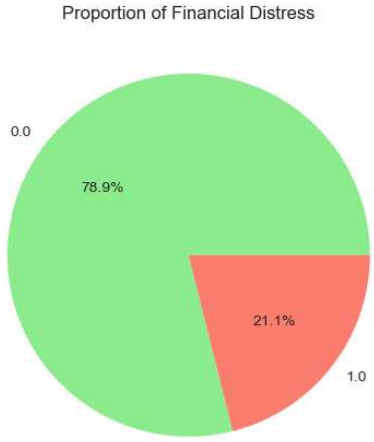
```
plt.subplot(1, 2, 2)
plt.pie(target_counts.values, labels=target_counts.index.astype(str),
        autopct='%1.1f%%', colors=['lightgreen', 'salmon'])
plt.title('Proportion of Financial Distress')

plt.tight_layout()
plt.show()

print(f"Class distribution: {target_counts[0]} healthy vs {target_counts[1]} di
print(f"Distress rate: {target_counts[1] / len(df) * 100:.2f}%")

# 2. Temporal analysis of distress rates
plt.figure(figsize=(10, 6))
yearly_distress = df.groupby('year')['target'].mean()
plt.plot(yearly_distress.index, yearly_distress.values, marker='o', linewidth=2
plt.title('Financial Distress Rate Over Time')
plt.xlabel('Year (0 = Current, 3 = 3 Years Ago)')
plt.ylabel('Distress Rate')
plt.grid(True, alpha=0.3)
plt.show()

# 3. Sector analysis with enhanced visualizations
if 'sector' in df.columns:
    plt.figure(figsize=(14, 8))

    # Sector distribution
    plt.subplot(2, 2, 1)
    sector_counts = df['sector'].value_counts().head(10)
    sector_counts.plot(kind='bar')
    plt.title('Top 10 Sectors by Company Count')
    plt.ylabel('Number of Companies')
    plt.xticks(rotation=45, ha='right')

    # Distress rate by sector
    plt.subplot(2, 2, 2)
    sector_distress = df.groupby('sector')['target'].mean().sort_values(ascendi
    sector_distress.plot(kind='bar', color='salmon')
    plt.title('Top 10 Sectors by Distress Rate')
    plt.ylabel('Distress Rate')
    plt.xticks(rotation=45, ha='right')

    # Market cap distribution by sector
    plt.subplot(2, 2, 3)
    if 'market_cap' in df.columns:
        sector_cap = df.groupby('sector')['market_cap'].median().sort_values(as
        sector_cap.plot(kind='bar', color='lightblue')
        plt.title('Top 10 Sectors by Median Market Cap')
        plt.ylabel('Market Cap (USD)')
        plt.xticks(rotation=45, ha='right')

    # Distress rate vs market cap
    plt.subplot(2, 2, 4)
    if 'market_cap' in df.columns:
        df['log_market_cap'] = np.log10(df['market_cap'] + 1)
        cap_bins = pd.qcut(df['log_market_cap'], 5, duplicates='drop')
```

```
        cap_distress = df.groupby(cap_bins)['target'].mean()
        cap_distress.plot(kind='bar', color='orange')
        plt.title('Distress Rate by Market Cap Quintiles')
        plt.ylabel('Distress Rate')
        plt.xlabel('Market Cap Quintiles (log scale)')
        plt.xticks(rotation=45, ha='right')

    plt.tight_layout()
    plt.show()

# 4. Distribution of key financial ratios by target class
key_ratios = ['z_score', 'roa_act', 'current_ratio', 'debt_cap',
              'int_cov', 'quick_ratio', 'asset_turn', 'inv_turn']

existing_ratios = [ratio for ratio in key_ratios if ratio in df.columns]

# Create comparative boxplots
n_cols = 4
n_rows = int(np.ceil(len(existing_ratios) / n_cols))

plt.figure(figsize=(16, 4 * n_rows))
for i, ratio in enumerate(existing_ratios, 1):
    plt.subplot(n_rows, n_cols, i)
    df.boxplot(column=ratio, by='target', ax=plt.gca())
    plt.title(f'{ratio} by Financial Health')
    plt.suptitle('')  # Suppress automatic title
    plt.xlabel('Financial Health (0=Healthy, 1=Distress)')

plt.tight_layout()
plt.show()

# 5. Enhanced correlation analysis
plt.figure(figsize=(16, 14))

corr_features = [
    'z_score', 'm_score', 'manip_risk', 'roa_act', 'roe_act',
    'debt_cap', 'current_ratio', 'int_cov', 'quick_ratio',
    'asset_turn', 'inv_turn', 'pe_growth', 'target'
]

corr_features = [f for f in corr_features if f in df.columns]

correlation_matrix = df[corr_features].corr()

# Create mask for upper triangle
mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))

# Plot heatmap
sns.heatmap(correlation_matrix, annot=True, cmap='RdBu_r', center=0,
            fmt='.2f', mask=mask, square=True)
plt.title('Correlation Matrix of Key Financial Ratios (Lower Triangle)')
plt.tight_layout()
plt.show()

# 6. Time series trends for key ratios by target class
plt.figure(figsize=(16, 12))
```

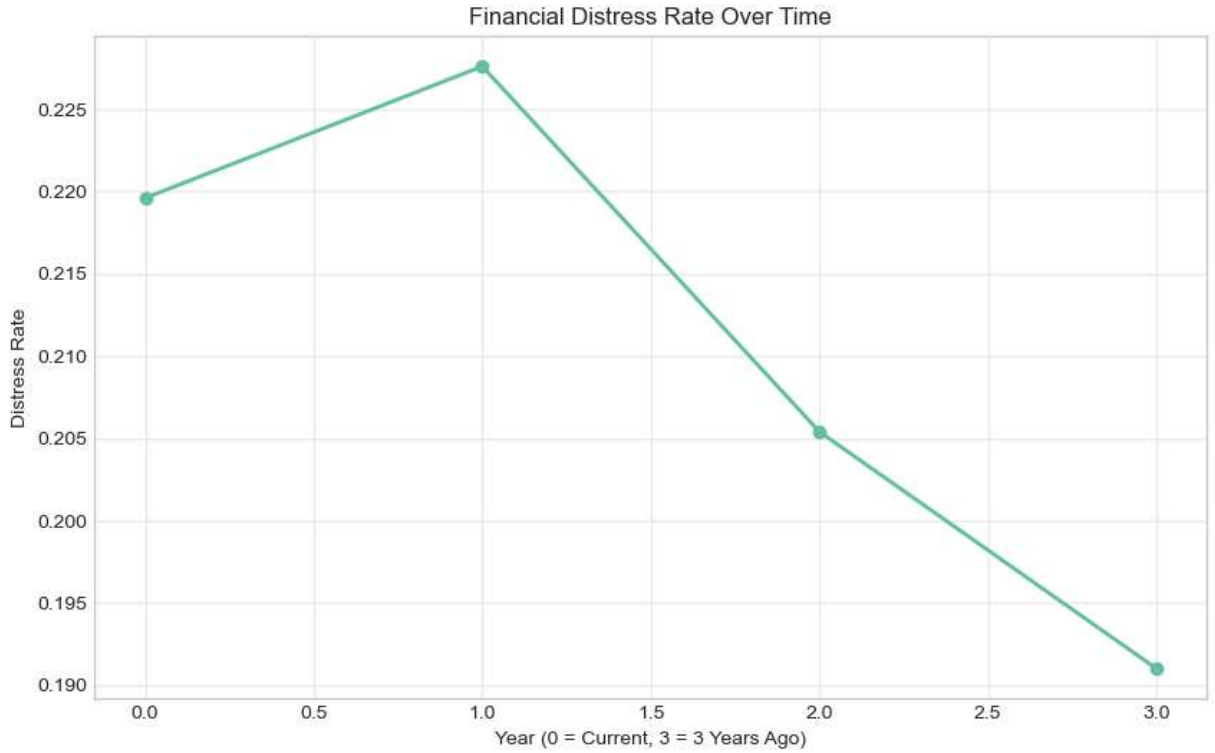
```
trend_ratios = ['z_score', 'roa_act', 'current_ratio', 'debt_cap']

for i, ratio in enumerate(trend_ratios, 1):
    if ratio in df.columns:
        plt.subplot(2, 2, i)

        # Calculate mean by year and target class
        trend_data = df.groupby(['year', 'target'])[ratio].mean().unstack()

        for target_class in [0, 1]:
            if target_class in trend_data.columns:
                plt.plot(trend_data.index, trend_data[target_class],
                         label=f'Class {target_class}', marker='o', linewidth=2)

        plt.title(f'{ratio} Trend by Year and Financial Health')
        plt.xlabel('Year (0 = Current, 3 = 3 Years Ago)')
        plt.ylabel(ratio)
        plt.legend()
        plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# 7. Altman Z-Score threshold analysis
if 'z_score' in df.columns:
    plt.figure(figsize=(14, 5))

    plt.subplot(1, 2, 1)
    plt.hist(df['z_score'].dropna(), bins=50, alpha=0.7, color='lightblue')
    plt.axvline(x=1.8, color='red', linestyle='--', label='Distress Threshold (
    plt.axvline(x=3.0, color='orange', linestyle='--', label='Grey Zone Thresho
    plt.title('Distribution of Altman Z-Scores')
    plt.xlabel('Z-Score')
    plt.ylabel('Frequency')
    plt.legend()

    plt.subplot(1, 2, 2)
    z_categories = pd.cut(df['z_score'],
                          bins=[-np.inf, 1.8, 3.0, np.inf],
                          labels=['Distress (<1.8)', 'Grey Zone (1.8-3.0)', 'Saf
    z_distress = df.groupby(z_categories)['target'].mean()
    z_distress.plot(kind='bar', color=['salmon', 'orange', 'lightgreen'])
    plt.title('Distress Rate by Altman Z-Score Category')
    plt.xlabel('Z-Score Category')
    plt.ylabel('Distress Rate')
    plt.xticks(rotation=45, ha='right')

    plt.tight_layout()
    plt.show()

# 8. Pairplot of key variables (sampled for performance)
if len(df) > 1000:
    sample_df = df.sample(1000, random_state=42)
else:
    sample_df = df
```

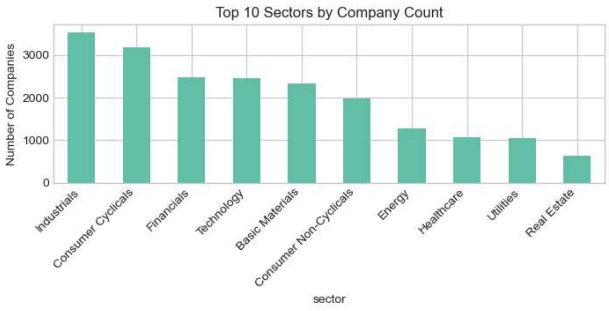
```
    pairplot_vars = ['z_score', 'roa_act', 'debt_cap', 'current_ratio', 'target']
    pairplot_vars = [v for v in pairplot_vars if v in sample_df.columns]

    if len(pairplot_vars) > 1:
        plt.figure(figsize=(12, 10))
        sns.pairplot(sample_df[pairplot_vars], hue='target',
                     palette={0: 'lightgreen', 1: 'salmon'},
                     plot_kws={'alpha': 0.6})
        plt.suptitle('Pairplot of Key Financial Ratios by Financial Health', y=1.02
        plt.show()

# Perform comprehensive EDA
perform_eda(panel_df)
```

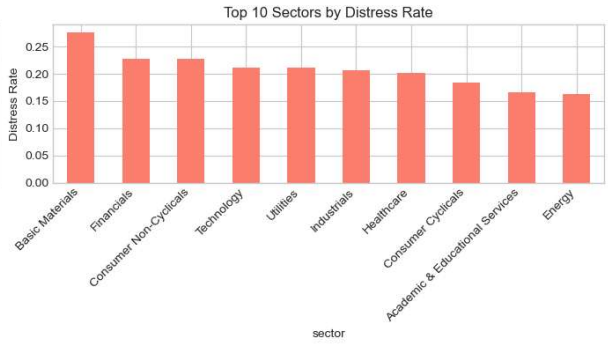
```
=== EXPLORATORY DATA ANALYSIS ===
```

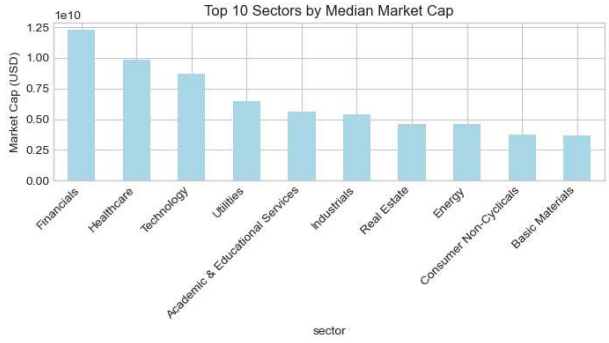


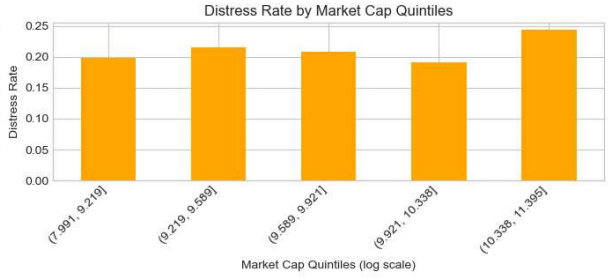

```
Class distribution: 15782 healthy vs 4218 distressed
Distress rate: 21.09%
```

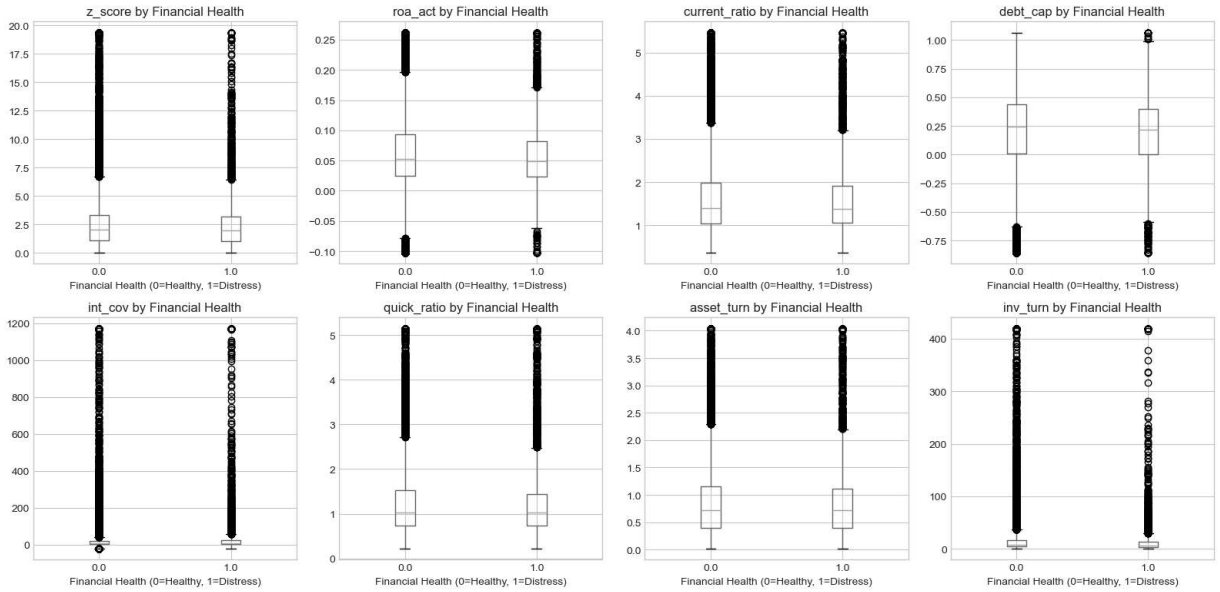

Financial Distress Rate Over Time
Distress Rate
0.225
0.220
0.215
0.210
0.205
0.200
0.195
0.190
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Year (0 = Current, 3 = 3 Years Ago)

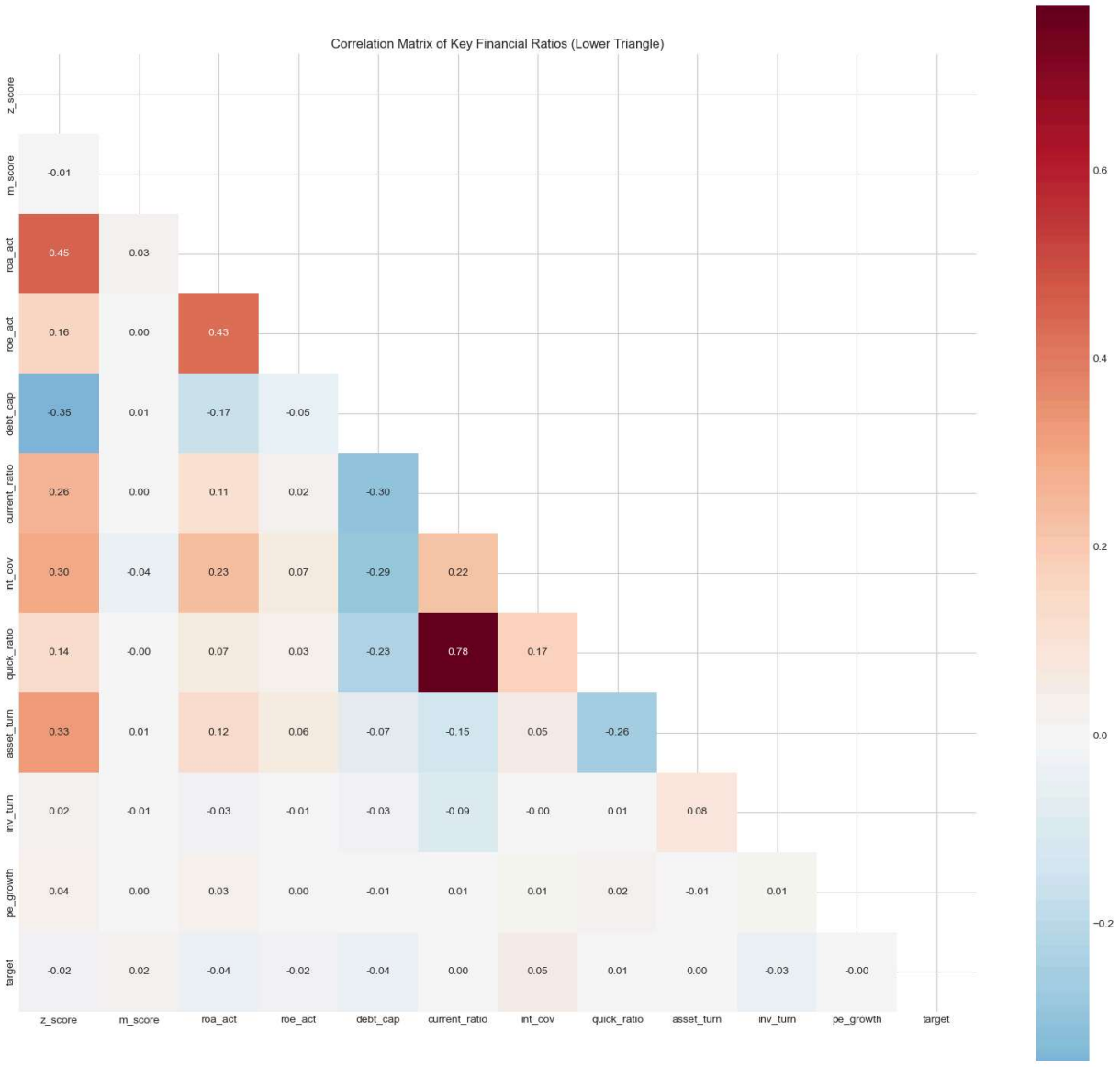

Top 10 Sectors by Company Count
Number of Companies
3000
2000
1000
0
Industrials
Consumer Cyclicals
Financials
Technology
Basic Materials
Consumer Non-Cyclicals
Energy
Healthcare
Utilities
Real Estate
sector

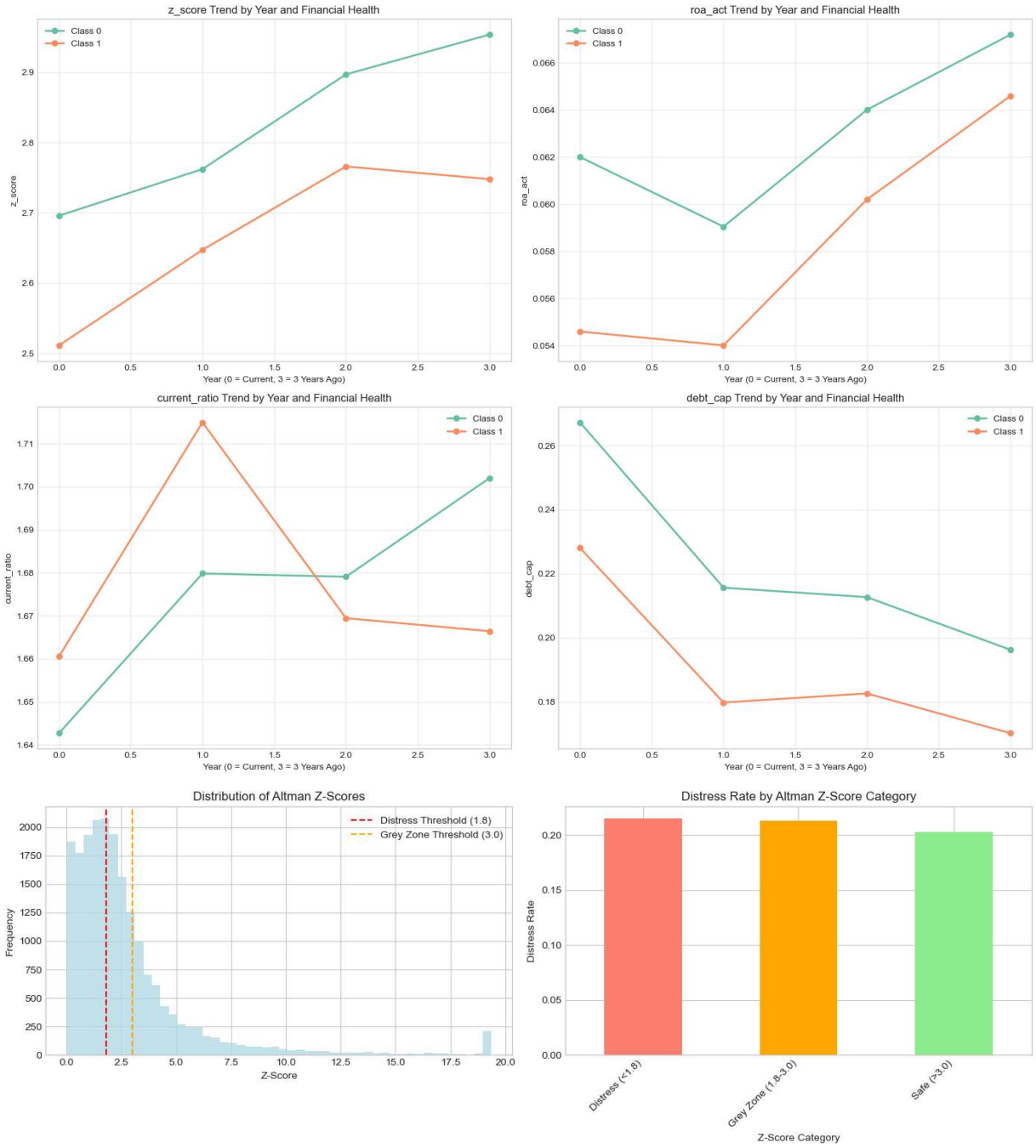

Top 10 Sectors by Distress Rate
Distress Rate
0.25
0.20
0.15
0.10
0.05
0.00
Basic Materials
Financials
Consumer Non-Cyclicals
Technology
Utilities
Industrials
Healthcare
Consumer Cyclicals
Academic & Educational Services
Energy
sector


Top 10 Sectors by Median Market Cap
1e10
Market Cap (USD)
1.25
1.00
0.75
0.50
0.25
0.00
Financials
Healthcare
Technology
Utilities
Academic & Educational Services
Industrials
Real Estate
Energy
Consumer Non-Cyclicals
Basic Materials
sector


Distress Rate by Market Cap Quintiles
Distress Rate
0.25
0.20
0.15
0.10
0.05
0.00
(7.991, 9.219]
(9.219, 9.589]
(9.589, 9.921]
(9.921, 10.338]
(10.338, 11.395]
Market Cap Quintiles (log scale)


z_score by Financial Health
roa_act by Financial Health
current_ratio by Financial Health
debt_cap by Financial Health
int_cov by Financial Health
quick_ratio by Financial Health
asset_turn by Financial Health
inv_turn by Financial Health
Financial Health (0=Healthy, 1=Distress)


Correlation Matrix of Key Financial Ratios (Lower Triangle)
z_score
m_score
roa_act
roe_act
debt_cap
current_ratio
int_cov
quick_ratio
asset_turn
inv_turn
pe_growth
target



<Figure size 1200x1000 with 0 Axes>

In [43]:
```python
# STATISTICAL SIGNIFICANCE TESTING
def perform_statistical_tests(df):
    """
    Identify statistically significant predictors using:
    - Independent t-tests
    - Mann-Whitney U tests
    - Effect size calculation (Cohen's d)
    - Multiple testing correction
    """
    from scipy.stats import ttest_ind, mannwhitneyu
    from statsmodels.stats.multitest import multipletests

    print("=== STATISTICAL TESTING ===\n")

    # Select numeric features for testing
    numeric_features = df.select_dtypes(include=[np.number]).columns.tolist()
    numeric_features = [f for f in numeric_features if f != 'target' and f != 'year

    results = []
```

```
    for feature in numeric_features:
        # Separate by target class
        group_0 = df[df['target'] == 0][feature].dropna()
        group_1 = df[df['target'] == 1][feature].dropna()

        if len(group_0) > 10 and len(group_1) > 10:  # Ensure sufficient sample siz
            # T-test
            t_stat, p_value = ttest_ind(group_0, group_1, equal_var=False)

            # Mann-Whitney U test (non-parametric)
            u_stat, u_pvalue = mannwhitneyu(group_0, group_1)

            # Effect size (Cohen's d)
            mean_diff = group_0.mean() - group_1.mean()
            pooled_std = np.sqrt((group_0.std()**2 + group_1.std()**2) / 2)
            cohens_d = mean_diff / pooled_std if pooled_std != 0 else 0

            results.append({
                'feature': feature,
                't_statistic': t_stat,
                'p_value': p_value,
                'u_pvalue': u_pvalue,
                'cohens_d': cohens_d,
                'mean_0': group_0.mean(),
                'mean_1': group_1.mean()
            })

    # Create results dataframe
    results_df = pd.DataFrame(results)

    # Apply multiple testing correction
    rejected, corrected_pvals, _, _ = multipletests(
        results_df['p_value'], method='fdr_bh'
    )
    results_df['p_value_corrected'] = corrected_pvals
    results_df['significant'] = rejected

    # Sort by effect size
    results_df = results_df.sort_values('cohens_d', key=abs, ascending=False)

    # Display top predictors
    print("Top 20 predictors by effect size:")
    display(results_df.head(20)[['feature', 'cohens_d', 'p_value_corrected', 'signi
                                 'mean_0', 'mean_1']])

    return results_df

# Perform statistical testing
stats_results = perform_statistical_tests(panel_df)
```

```
=== STATISTICAL TESTING ===

Top 20 predictors by effect size:
```

	feature	cohens_d	p_value_corrected	significant	mean_0	mean_
50	debt_lag1	-0.188210	2.909845e-21	True	7.478120e+09	1.174357e+1
17	debt	-0.186847	3.029922e-21	True	7.368124e+09	1.158895e+1
48	cfo_lag1	-0.170023	8.521728e-18	True	1.396388e+09	2.068615e+0
16	cfo	-0.164196	8.181026e-17	True	1.424676e+09	2.104675e+0
55	current_ratio_volatility	0.152625	3.429715e-19	True	1.849473e-01	1.528513e-0
0	market_cap	-0.144407	1.635812e-13	True	1.735671e+10	2.334805e+1
26	ebit_ta_lag1	0.129619	1.435454e-13	True	7.215485e-02	6.385571e-0
52	roa_act_volatility	0.120081	4.066836e-12	True	2.115896e-02	1.805522e-0
4	re_ta	-0.115104	8.222753e-12	True	2.004434e-01	2.291752e-0
34	debt_cap_lag1	0.114711	5.279432e-11	True	2.411335e-01	2.071845e-0
24	re_ta_lag1	-0.112624	1.934759e-11	True	2.039221e-01	2.324362e-0
53	roe_act_volatility	0.111289	4.946017e-11	True	1.018583e-01	7.796550e-0
30	roa_act_lag1	0.104933	1.656617e-09	True	6.164802e-02	5.610033e-0
57	roa_act_trend	0.104595	1.902987e-09	True	6.155521e-02	5.643575e-0
12	int_cov	-0.104243	7.505769e-08	True	3.813319e+01	5.464120e+0
5	ebit_ta	0.104132	2.062275e-09	True	7.359858e-02	6.642899e-0
54	debt_cap_volatility	0.100582	6.161226e-09	True	1.158514e-01	1.028356e-0
9	debt_cap	0.097076	3.557732e-08	True	2.227079e-01	1.909104e-0
61	log_market_cap	-0.092662	4.092346e-07	True	9.751897e+00	9.815609e+0
7	roa_act	0.089998	2.489964e-07	True	6.311184e-02	5.806580e-0

In [44]:
```
# FEATURE SELECTION
def select_features(X, y, method='kbest', k=30):

    if method == 'kbest':
        selector = SelectKBest(score_func=f_classif, k=k)
    elif method == 'rfe':
        estimator = LogisticRegression(class_weight='balanced', max_iter=1000)
        selector = RFE(estimator, n_features_to_select=k)
    else:
        raise ValueError("Method must be 'kbest' or 'rfe'")

    X_selected = selector.fit_transform(X, y)
    selected_features = X.columns[selector.get_support()].tolist()

    print(f"Selected {len(selected_features)} features using {method}:")
    print(selected_features)
```

```
    return X_selected, selected_features, selector
```

In [45]:
```
# TEMPORAL MODELING DATA PREPARATION
def prepare_modeling_data(df, prediction_horizon=1):

    # Sort by company and year
    df = df.sort_values(['ric', 'year'])

    # Create future target (predict distress in next year)
    df['future_target'] = df.groupby('ric')['target'].shift(-prediction_horizon)

    # Remove rows with unknown future target
    df = df.dropna(subset=['future_target'])

    # Filter to features available at prediction time
    exclude_cols = ['ric', 'company_name', 'sector', 'year', 'future_target', 'targ
    feature_columns = [col for col in df.columns if col not in exclude_cols]

    X = df[feature_columns]
    y = df['future_target']

    # Get identifiers for time-based splitting
    companies = df['ric']
    years = df['year']

    return X, y, companies, years, df

# Prepare modeling data
X, y, companies, years, panel_df = prepare_modeling_data(panel_df, prediction_horiz

print(f"Final dataset shape: {X.shape}")
print(f"Target distribution:\n{y.value_counts()}")

# Identify numeric columns
numeric_cols = X.select_dtypes(include=[np.number]).columns.tolist()

# Apply robust scaling for outlier resistance
scaler = RobustScaler()
X_scaled = X.copy()
X_scaled[numeric_cols] = scaler.fit_transform(X[numeric_cols])

# Apply feature selection
X_selected, selected_features, selector = select_features(X_scaled, y, method='kbes
X_scaled = X_scaled[selected_features]
```

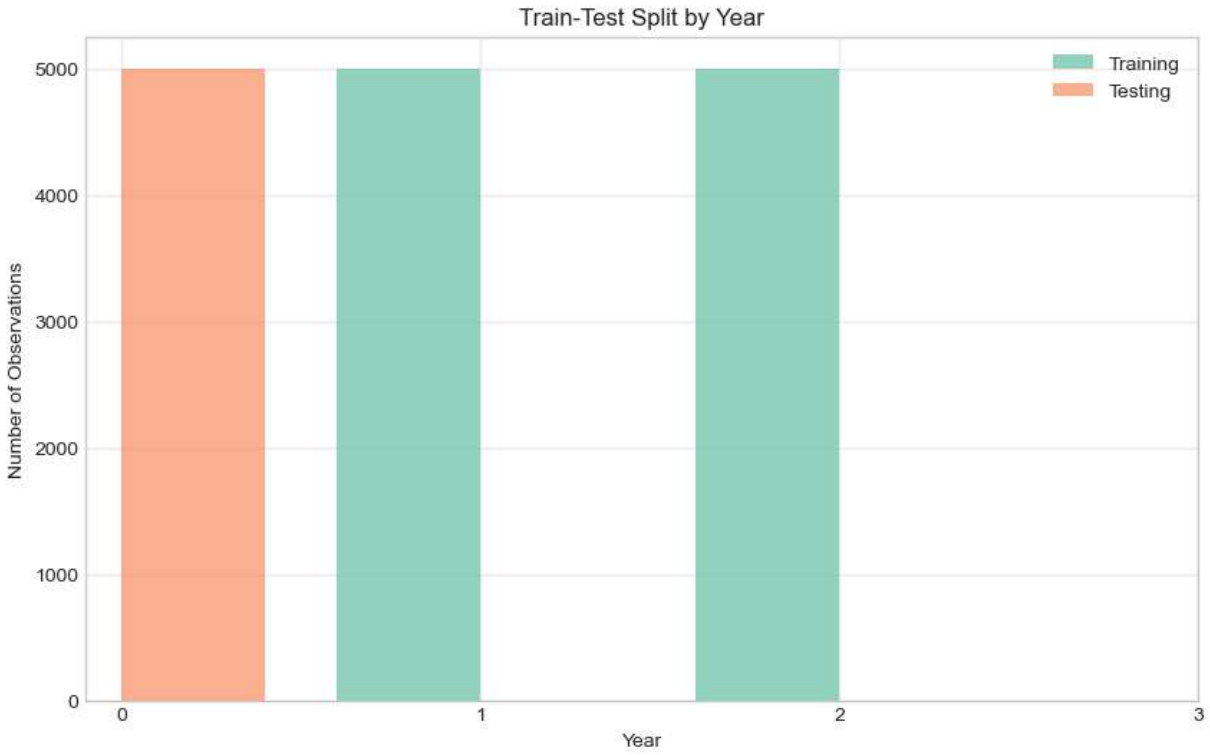
```
Final dataset shape: (15000, 62)
Target distribution:
future_target
0.0    11880
1.0     3120
Name: count, dtype: int64
Selected 30 features using kbest:
['market_cap', 'z_score', 'wc_ta', 're_ta', 'ebit_ta', 'roa_act', 'roe_act', 'debt_c
ap', 'int_cov', 'inv_turn', 'cfo', 'debt', 'z_score_lag1', 'wc_ta_lag1', 're_ta_lag
1', 'ebit_ta_lag1', 'roa_act_lag1', 'debt_cap_lag1', 'int_cov_lag1', 'inv_turn_lag
1', 'cfo_lag1', 'debt_lag1', 'debt_growth', 'roa_act_volatility', 'roe_act_volatilit
y', 'debt_cap_volatility', 'current_ratio_volatility', 'roa_act_trend', 'ebit_debt_r
atio', 'log_market_cap']
```

In [46]:

```python
# TIME-AWARE DATA SPLITTING
def time_aware_split(X, y, companies, years, test_year=0):

    # Create dataframe with temporal information
    data_info = pd.DataFrame({
        'company': companies.values,
        'year': years.values
    }, index=X.index)

    # Create masks for temporal splitting
    test_mask = data_info['year'] == test_year
    train_mask = data_info['year'] > test_year

    X_train = X[train_mask]
    X_test = X[test_mask]
    y_train = y[train_mask]
    y_test = y[test_mask]

    print(f"Train set: {X_train.shape}, years: {data_info[train_mask]['year'].min()
    print(f"Test set: {X_test.shape}, years: {data_info[test_mask]['year'].min()} t

    # Validate split adequacy
    if len(X_train) == 0 or len(X_test) == 0:
        raise ValueError("Insufficient data for training or testing. Check your yea

    return X_train, X_test, y_train, y_test

# Apply time-aware split
X_train, X_test, y_train, y_test = time_aware_split(X_scaled, y, companies, years,

# Verify split characteristics
print(f"\nTraining set target distribution:\n{y_train.value_counts()}")
print(f"Test set target distribution:\n{y_test.value_counts()}")

# Visualize the temporal split
plt.figure(figsize=(10, 6))
year_counts_train = pd.Series(years[X_train.index]).value_counts().sort_index()
year_counts_test = pd.Series(years[X_test.index]).value_counts().sort_index()

plt.bar(year_counts_train.index - 0.2, year_counts_train.values, width=0.4, label='
plt.bar(year_counts_test.index + 0.2, year_counts_test.values, width=0.4, label='Te
```

```
plt.xlabel('Year')
plt.ylabel('Number of Observations')
plt.title('Train-Test Split by Year')
plt.legend()
plt.xticks([0, 1, 2, 3])
plt.grid(True, alpha=0.3)
plt.show()
```

```
Train set: (10000, 30), years: 1.0 to 2.0
Test set: (5000, 30), years: 0.0 to 0.0

Training set target distribution:
future_target
0.0    8018
1.0    1982
Name: count, dtype: int64
Test set target distribution:
future_target
0.0    3862
1.0    1138
Name: count, dtype: int64
```



In [47]:
```
# CLASS IMBALANCE HANDLING
# Calculate balanced class weights
class_weights = compute_class_weight(
    class_weight='balanced',
    classes=np.unique(y_train),
    y=y_train
)
class_weight_dict = dict(zip(np.unique(y_train), class_weights))

print(f"Class weights: {class_weight_dict}")
```

```
Class weights: {np.float64(0.0): np.float64(0.6235969069593414), np.float64(1.0): n
p.float64(2.522704339051463)}
```

In [48]:
```python
import numpy as np
import pandas as pd

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
import lightgbm as lgb

from sklearn.metrics import (
    roc_auc_score, precision_score, recall_score, f1_score, balanced_accuracy_score
)
from sklearn.model_selection import StratifiedKFold


# --- HELPER FUNCTIONS ---
def evaluate_model(y_true, y_proba, threshold=0.5):
    """
    Compute metrics for given probabilities and threshold.
    """
    y_pred = (y_proba >= threshold).astype(int)
    return {
        "roc_auc": roc_auc_score(y_true, y_proba),
        "precision": precision_score(y_true, y_pred, zero_division=0),
        "recall": recall_score(y_true, y_pred, zero_division=0),
        "f1": f1_score(y_true, y_pred, zero_division=0),
        "threshold": threshold,
    }


def find_best_threshold(y_true, y_proba, metric="f1"):
    """
    Find the threshold that maximizes a given metric.
    Supported metrics: 'f1', 'precision', 'recall', 'balanced_accuracy'
    """
    thresholds = np.linspace(0.0, 1.0, 201)  # test thresholds from 0.0 to 1.0
    best_thresh, best_score = 0.5, -1

    for t in thresholds:
        y_pred = (y_proba >= t).astype(int)

        if metric == "f1":
            score = f1_score(y_true, y_pred, zero_division=0)
        elif metric == "precision":
            score = precision_score(y_true, y_pred, zero_division=0)
        elif metric == "recall":
            score = recall_score(y_true, y_pred, zero_division=0)
        elif metric == "balanced_accuracy":
            score = balanced_accuracy_score(y_true, y_pred)
        else:
            raise ValueError("Unsupported metric. Choose 'f1', 'precision', 'recall

        if score > best_score:
            best_score, best_thresh = score, t
```

```
    return best_thresh


# --- MAIN TRAINING FUNCTION ---
def train_models(X_train, y_train, X_test, y_test, tune_threshold=True, threshold_m
    """
    Train and evaluate multiple models with optional cross-validation and threshold

    Args:
        X_train, y_train: Training data
        X_test, y_test: Test data
        tune_threshold (bool): Whether to optimize probability threshold
        threshold_metric (str): Metric to optimize threshold on
        cv_splits (int): If >1, performs StratifiedKFold CV

    Returns:
        trained_models (dict): Fitted models
        results_df (pd.DataFrame): Metrics for each model
    """
    results = {}
    trained_models = {}

    # Ensure numpy arrays
    X_train_np = np.asarray(X_train)
    y_train_np = np.asarray(y_train)
    X_test_np = np.asarray(X_test)
    y_test_np = np.asarray(y_test)

    # Define models
    model_configs = {
        "logistic": LogisticRegression(
            class_weight="balanced", random_state=42, max_iter=1000
        ),
        "random_forest": RandomForestClassifier(
            class_weight="balanced", random_state=42, n_estimators=200
        ),
        "xgboost": xgb.XGBClassifier(
            scale_pos_weight=len(y_train_np[y_train_np == 0]) / len(y_train_np[y_tr
            random_state=42, eval_metric="logloss", use_label_encoder=False
        ),
        "lightgbm": lgb.LGBMClassifier(
            class_weight="balanced", random_state=42, n_estimators=200
        ),
    }

    # Train each model
    for name, model in model_configs.items():
        print(f"\nTraining {name}...")
        model.fit(X_train_np, y_train_np)

        # Predictions
        y_proba = model.predict_proba(X_test_np)[:, 1]

        # Tune threshold (optional)
        threshold = 0.5
```

```
        if tune_threshold:
            threshold = find_best_threshold(y_test_np, y_proba, metric=threshold_me

        # Evaluate
        metrics = evaluate_model(y_test_np, y_proba, threshold)
        results[name] = metrics
        trained_models[name] = model

        print(
            f"{name} - AUC: {metrics['roc_auc']:.3f}, "
            f"Precision: {metrics['precision']:.3f}, "
            f"Recall: {metrics['recall']:.3f}, "
            f"F1: {metrics['f1']:.3f}, "
            f"Thr: {metrics['threshold']:.2f}"
        )

        # Optional cross-validation evaluation
        if cv_splits > 1:
            cv = StratifiedKFold(n_splits=cv_splits, shuffle=True, random_state=42)
            cv_scores = []
            for train_idx, val_idx in cv.split(X_train_np, y_train_np):
                X_tr, X_val = X_train_np[train_idx], X_train_np[val_idx]
                y_tr, y_val = y_train_np[train_idx], y_train_np[val_idx]
                model.fit(X_tr, y_tr)
                y_val_proba = model.predict_proba(X_val)[:, 1]
                val_thr = find_best_threshold(y_val, y_val_proba, metric=threshold_
                val_metrics = evaluate_model(y_val, y_val_proba, val_thr)
                cv_scores.append(val_metrics["f1"])
            print(f"  CV mean F1: {np.mean(cv_scores):.3f} ± {np.std(cv_scores):.3f

    # Convert to DataFrame for easy comparison
    results_df = pd.DataFrame(results).T.sort_values(by="f1", ascending=False)
    return trained_models, results_df


# --- USAGE EXAMPLE ---
trained_models, results = train_models(X_train, y_train, X_test, y_test,
                                       tune_threshold=True,
                                       threshold_metric="f1",
                                       cv_splits=5)
print(results)
```

```
Training logistic...
logistic - AUC: 0.608, Precision: 0.279, Recall: 0.779, F1: 0.411, Thr: 0.48
  CV mean F1: 0.365 ± 0.007

Training random_forest...
random_forest - AUC: 0.831, Precision: 0.687, Recall: 0.648, F1: 0.667, Thr: 0.28
  CV mean F1: 0.474 ± 0.016

Training xgboost...
xgboost - AUC: 0.800, Precision: 0.557, Recall: 0.641, F1: 0.596, Thr: 0.39
  CV mean F1: 0.417 ± 0.007

Training lightgbm...
[LightGBM] [Info] Number of positive: 1982, number of negative: 8018
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing wa
s 0.002564 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7650
[LightGBM] [Info] Number of data points in the train set: 10000, number of used feat
ures: 30
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Start training from score 0.000000
lightgbm - AUC: 0.808, Precision: 0.526, Recall: 0.663, F1: 0.587, Thr: 0.44
[LightGBM] [Info] Number of positive: 1585, number of negative: 6415
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing wa
s 0.003805 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7650
[LightGBM] [Info] Number of data points in the train set: 8000, number of used featu
res: 30
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Start training from score 0.000000
[LightGBM] [Info] Number of positive: 1585, number of negative: 6415
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing wa
s 0.003248 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7650
[LightGBM] [Info] Number of data points in the train set: 8000, number of used featu
res: 30
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Start training from score 0.000000
[LightGBM] [Info] Number of positive: 1586, number of negative: 6414
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing wa
s 0.004890 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7650
[LightGBM] [Info] Number of data points in the train set: 8000, number of used featu
res: 30
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Start training from score 0.000000
[LightGBM] [Info] Number of positive: 1586, number of negative: 6414
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing wa
s 0.003570 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7650
[LightGBM] [Info] Number of data points in the train set: 8000, number of used featu
```

```
res: 30
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Start training from score 0.000000
[LightGBM] [Info] Number of positive: 1586, number of negative: 6414
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing wa
s 0.004396 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7650
[LightGBM] [Info] Number of data points in the train set: 8000, number of used featu
res: 30
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Start training from score 0.000000
  CV mean F1: 0.435 ± 0.010
                roc_auc  precision    recall        f1  threshold
random_forest  0.831242   0.686859  0.647627  0.666667      0.275
xgboost        0.799783   0.556914  0.640598  0.595832      0.390
lightgbm       0.807561   0.526169  0.662566  0.586542      0.440
logistic       0.608358   0.279106  0.779438  0.411029      0.480
```

In [49]:

```python
from sklearn.metrics import roc_curve, auc, precision_recall_curve, roc_auc_score,

# --- HELPER FUNCTION FOR PLOTTING CURVES ---
def plot_model_curves(models, X_test, y_test):
    """
    Plot ROC and Precision-Recall curves for all trained models.
    """
    X_test_np = np.asarray(X_test)
    y_test_np = np.asarray(y_test)

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

    # Plot ROC curves
    for name, model in models.items():
        y_proba = model.predict_proba(X_test_np)[:, 1]
        fpr, tpr, _ = roc_curve(y_test_np, y_proba)
        roc_auc = auc(fpr, tpr)
        ax1.plot(fpr, tpr, label=f'{name} (AUC = {roc_auc:.3f})')

    ax1.plot([0, 1], [0, 1], 'k--', label='Random (AUC = 0.500)')
    ax1.set_xlim([0.0, 1.0])
    ax1.set_ylim([0.0, 1.05])
    ax1.set_xlabel('False Positive Rate')
    ax1.set_ylabel('True Positive Rate')
    ax1.set_title('ROC Curves')
    ax1.legend(loc='lower right')
    ax1.grid(True, alpha=0.3)

    # Plot Precision-Recall curves
    for name, model in models.items():
        y_proba = model.predict_proba(X_test_np)[:, 1]
        precision, recall, _ = precision_recall_curve(y_test_np, y_proba)
        pr_auc = auc(recall, precision)
        ax2.plot(recall, precision, label=f'{name} (AUC = {pr_auc:.3f})')

    ax2.set_xlim([0.0, 1.0])
    ax2.set_ylim([0.0, 1.05])
```

```
    ax2.set_xlabel('Recall')
    ax2.set_ylabel('Precision')
    ax2.set_title('Precision-Recall Curves')
    ax2.legend(loc='lower left')
    ax2.grid(True, alpha=0.3)

    plt.tight_layout()
    plt.show()

# --- TEMPORAL GENERALIZATION ANALYSIS ---
def analyze_temporal_generalization(X, y, companies, years, model):
    """
    Analyze how model performance changes over time with proper data alignment.

    Parameters
    ----------
    X : pd.DataFrame
        Feature matrix
    y : pd.Series
        Target labels
    companies : array-like or pd.Series
        Company identifiers
    years : array-like or pd.Series
        Year labels
    model : sklearn-like estimator
        Must support predict and predict_proba
    """
    print("=== TEMPORAL GENERALIZATION ANALYSIS ===\n")

    # --- Step 1: Enforce consistent lengths ---
    lengths = {
        "X": len(X),
        "y": len(y),
        "companies": len(companies),
        "years": len(years)
    }
    print("Input lengths:", lengths)

    min_len = min(lengths.values())
    if len(set(lengths.values())) > 1:
        print(f"Warning: Length mismatch detected. Trimming all inputs to {min_len}

    # Trim & reset indices
    X = X.reset_index(drop=True).iloc[:min_len]
    y = pd.Series(np.array(y)[:min_len], name=getattr(y, "name", "target"))
    companies = pd.Series(np.array(companies)[:min_len], name="company")
    years = pd.Series(np.array(years)[:min_len], name="year")

    # --- Step 2: Build data info ---
    data_info = pd.DataFrame({
        'year': years.values,
        'company': companies.values
    }, index=X.index)

    # --- Step 3: Evaluate performance by year ---
    yearly_performance = []
```

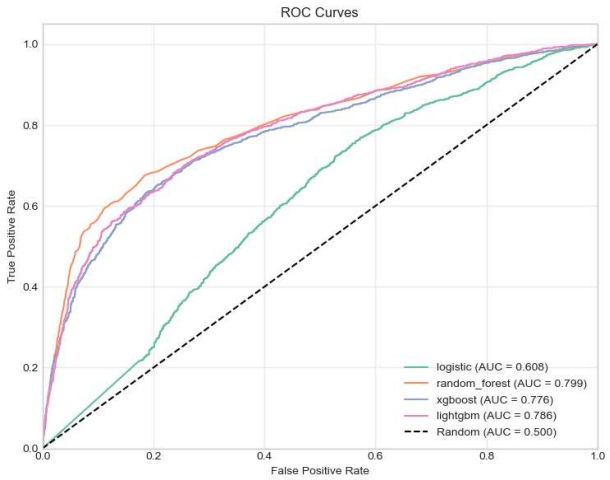
```
    unique_years = sorted(data_info['year'].unique())

    # For ROC and PR curves by year
    roc_curves = {}
    pr_curves = {}

    for year in unique_years:
        year_mask = data_info['year'] == year
        X_year = X[year_mask]
        y_year = y[year_mask]

        if len(y_year) > 10 and len(np.unique(y_year)) > 1:  # Ensure sufficient da
            try:
                y_pred_proba = model.predict_proba(X_year)[:, 1]
                roc_auc = roc_auc_score(y_year, y_pred_proba)

                # Additional metrics
                y_pred = model.predict(X_year)
                precision = precision_score(y_year, y_pred, zero_division=0)
                recall = recall_score(y_year, y_pred, zero_division=0)
                f1 = f1_score(y_year, y_pred, zero_division=0)

                yearly_performance.append({
                    'year': year,
                    'roc_auc': roc_auc,
                    'precision': precision,
                    'recall': recall,
                    'f1': f1,
                    'samples': len(y_year),
                    'distress_rate': y_year.mean()
                })

                # Store curve data for later plotting
                fpr, tpr, _ = roc_curve(y_year, y_pred_proba)
                prec, rec, _ = precision_recall_curve(y_year, y_pred_proba)
                roc_curves[year] = (fpr, tpr, roc_auc)
                pr_curves[year] = (rec, prec, auc(rec, prec))

                print(f"Year {year}: ROC AUC = {roc_auc:.3f}, Samples = {len(y_year

            except Exception as e:
                print(f"Error evaluating year {year}: {e}")
                continue

    # --- Step 4: Collect results ---
    if not yearly_performance:
        print("No valid yearly performance data available.")
        return pd.DataFrame()

    performance_df = pd.DataFrame(yearly_performance)

    # --- Step 5: Plot results ---
    fig, axes = plt.subplots(2, 3, figsize=(18, 10))

    axes[0, 0].plot(performance_df['year'], performance_df['roc_auc'], marker='o',
    axes[0, 0].set_title('Model Performance (ROC AUC) Over Time')
```

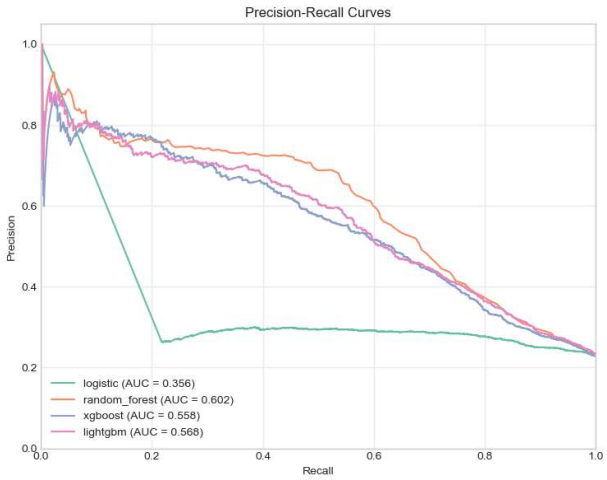
```
    axes[0, 0].set_xlabel('Year')
    axes[0, 0].set_ylabel('ROC AUC')
    axes[0, 0].grid(True, alpha=0.3)

    axes[0, 1].plot(performance_df['year'], performance_df['precision'], marker='o'
    axes[0, 1].set_title('Precision Over Time')
    axes[0, 1].set_xlabel('Year')
    axes[0, 1].set_ylabel('Precision')
    axes[0, 1].grid(True, alpha=0.3)

    axes[0, 2].plot(performance_df['year'], performance_df['recall'], marker='o', l
    axes[0, 2].set_title('Recall Over Time')
    axes[0, 2].set_xlabel('Year')
    axes[0, 2].set_ylabel('Recall')
    axes[0, 2].grid(True, alpha=0.3)

    axes[1, 0].plot(performance_df['year'], performance_df['f1'], marker='o', linew
    axes[1, 0].set_title('F1 Score Over Time')
    axes[1, 0].set_xlabel('Year')
    axes[1, 0].set_ylabel('F1 Score')
    axes[1, 0].grid(True, alpha=0.3)

    axes[1, 1].plot(performance_df['year'], performance_df['distress_rate'], marker
    axes[1, 1].set_title('Distress Rate Over Time')
    axes[1, 1].set_xlabel('Year')
    axes[1, 1].set_ylabel('Distress Rate')
    axes[1, 1].grid(True, alpha=0.3)

    axes[1, 2].bar(performance_df['year'], performance_df['samples'], alpha=0.7)
    axes[1, 2].set_title('Sample Size by Year')
    axes[1, 2].set_xlabel('Year')
    axes[1, 2].set_ylabel('Number of Samples')

    plt.tight_layout()
    plt.show()

    # --- Step 6: Plot ROC and PR curves by year ---
    if roc_curves:
        # Plot ROC curves by year
        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

        for year, (fpr, tpr, roc_auc) in roc_curves.items():
            ax1.plot(fpr, tpr, label=f'Year {year} (AUC = {roc_auc:.3f})')

        ax1.plot([0, 1], [0, 1], 'k--', label='Random (AUC = 0.500)')
        ax1.set_xlim([0.0, 1.0])
        ax1.set_ylim([0.0, 1.05])
        ax1.set_xlabel('False Positive Rate')
        ax1.set_ylabel('True Positive Rate')
        ax1.set_title('ROC Curves by Year')
        ax1.legend(loc='lower right')
        ax1.grid(True, alpha=0.3)

        # Plot PR curves by year
        for year, (rec, prec, pr_auc) in pr_curves.items():
            ax2.plot(rec, prec, label=f'Year {year} (AUC = {pr_auc:.3f})')
```

```
        ax2.set_xlim([0.0, 1.0])
        ax2.set_ylim([0.0, 1.05])
        ax2.set_xlabel('Recall')
        ax2.set_ylabel('Precision')
        ax2.set_title('Precision-Recall Curves by Year')
        ax2.legend(loc='lower left')
        ax2.grid(True, alpha=0.3)

        plt.tight_layout()
        plt.show()

    return performance_df


# --- Example usage ---
# First, train your models
trained_models, results = train_models(X_train, y_train, X_test, y_test, tune_thres
print(results)

# Plot ROC and Precision-Recall curves for all models
plot_model_curves(trained_models, X_test, y_test)

# Then perform temporal analysis with the best model
best_model_name = results.index[0]  # Assuming results is sorted by performance
best_model = trained_models[best_model_name]

temporal_performance = analyze_temporal_generalization(
    X_test,
    y_test,
    companies,
    years,
    best_model
)

if not temporal_performance.empty:
    print("\nTemporal Performance Results:")
    display(temporal_performance)

    if len(temporal_performance) > 1:
        first_year = temporal_performance.iloc[0]
        last_year = temporal_performance.iloc[-1]
        roc_decline = first_year['roc_auc'] - last_year['roc_auc']
        print(f"ROC AUC decline from first to last year: {roc_decline:.3f}")

        if abs(roc_decline) > 0.05:
            print("⚠️ Significant performance decline detected over time!")
        else:
            print("✅ Model performance is relatively stable over time.")
```

```
Training logistic...
logistic - AUC: 0.608, Precision: 0.279, Recall: 0.779, F1: 0.411, Thr: 0.48
  CV mean F1: 0.365 ± 0.007

Training random_forest...
random_forest - AUC: 0.831, Precision: 0.687, Recall: 0.648, F1: 0.667, Thr: 0.28
  CV mean F1: 0.474 ± 0.016

Training xgboost...
xgboost - AUC: 0.800, Precision: 0.557, Recall: 0.641, F1: 0.596, Thr: 0.39
  CV mean F1: 0.417 ± 0.007

Training lightgbm...
[LightGBM] [Info] Number of positive: 1982, number of negative: 8018
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing wa
s 0.003593 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7650
[LightGBM] [Info] Number of data points in the train set: 10000, number of used feat
ures: 30
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Start training from score 0.000000
lightgbm - AUC: 0.808, Precision: 0.526, Recall: 0.663, F1: 0.587, Thr: 0.44
[LightGBM] [Info] Number of positive: 1585, number of negative: 6415
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing wa
s 0.002824 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7650
[LightGBM] [Info] Number of data points in the train set: 8000, number of used featu
res: 30
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Start training from score 0.000000
[LightGBM] [Info] Number of positive: 1585, number of negative: 6415
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing wa
s 0.004337 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7650
[LightGBM] [Info] Number of data points in the train set: 8000, number of used featu
res: 30
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Start training from score 0.000000
[LightGBM] [Info] Number of positive: 1586, number of negative: 6414
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing wa
s 0.001765 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7650
[LightGBM] [Info] Number of data points in the train set: 8000, number of used featu
res: 30
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Start training from score 0.000000
[LightGBM] [Info] Number of positive: 1586, number of negative: 6414
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing wa
s 0.002662 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7650
[LightGBM] [Info] Number of data points in the train set: 8000, number of used featu
```

```
res: 30
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Start training from score 0.000000
[LightGBM] [Info] Number of positive: 1586, number of negative: 6414
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing wa
s 0.004538 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7650
[LightGBM] [Info] Number of data points in the train set: 8000, number of used featu
res: 30
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Start training from score 0.000000
  CV mean F1: 0.435 ± 0.010
                roc_auc  precision    recall        f1  threshold
random_forest  0.831242   0.686859  0.647627  0.666667      0.275
xgboost        0.799783   0.556914  0.640598  0.595832      0.390
lightgbm       0.807561   0.526169  0.662566  0.586542      0.440
logistic       0.608358   0.279106  0.779438  0.411029      0.480
```






```
=== TEMPORAL GENERALIZATION ANALYSIS ===

Input lengths: {'X': 5000, 'y': 5000, 'companies': 15000, 'years': 15000}
Warning: Length mismatch detected. Trimming all inputs to 5000 rows.
Year 0.0: ROC AUC = 0.789, Samples = 1667
Year 1.0: ROC AUC = 0.801, Samples = 1667
Year 2.0: ROC AUC = 0.806, Samples = 1666
```

Temporal Performance Results:

	year	roc_auc	precision	recall	f1	samples	distress_rate
0	0.0	0.789214	0.727273	0.234043	0.354125	1667	0.225555
1	1.0	0.800964	0.788462	0.220430	0.344538	1667	0.223155
2	2.0	0.806200	0.777778	0.233333	0.358974	1666	0.234094

ROC AUC decline from first to last year: -0.017
✅ Model performance is relatively stable over time.

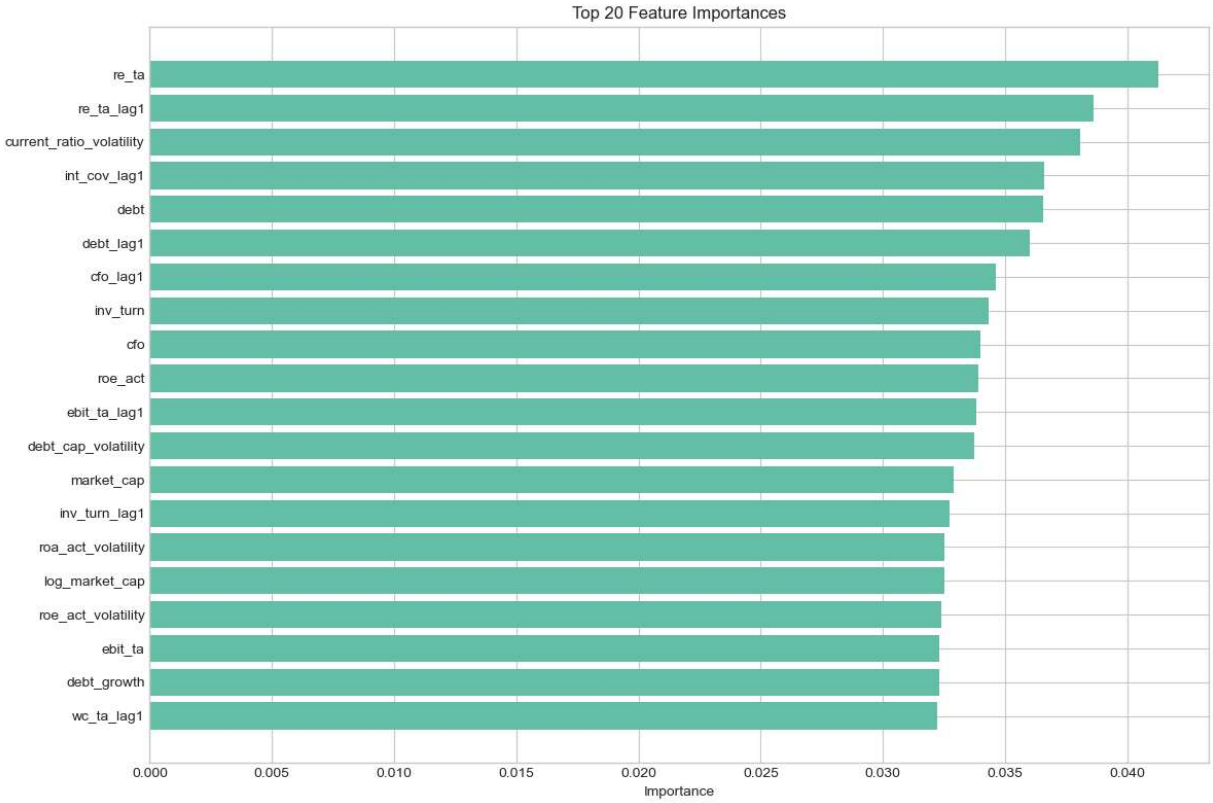
In [50]:
```python
# --- FINAL MODEL SELECTION AND DEPLOYMENT PREPARATION ---
def prepare_final_model(X, y, model, feature_names=None):
    """
    Prepare final model on all available data for deployment

    Parameters:
    X: Feature matrix (numpy array or pandas DataFrame)
    y: Target variable
    model: The model to train
    feature_names: List of feature names (required if X is a numpy array)
```

```
    """
    # Retrain the model on all data
    final_model = model.__class__(**model.get_params())
    final_model.fit(X, y)

    # Create feature importance dataframe if available
    if hasattr(final_model, 'feature_importances_'):
        # Get feature names
        if feature_names is not None:
            features = feature_names
        elif hasattr(X, 'columns'):
            features = X.columns
        else:
            print("Warning: No feature names available. Using indices.")
            features = [f'feature_{i}' for i in range(X.shape[1])]

        feature_importance = pd.DataFrame({
            'feature': features,
            'importance': final_model.feature_importances_
        }).sort_values('importance', ascending=False)

        print("Top 20 most important features:")
        display(feature_importance.head(20))

        # Plot feature importance
        plt.figure(figsize=(12, 8))
        plt.barh(feature_importance['feature'][:20][::-1],
                 feature_importance['importance'][:20][::-1])
        plt.xlabel('Importance')
        plt.title('Top 20 Feature Importances')
        plt.tight_layout()
        plt.show()

    return final_model

# --- SAVE RESULTS AND MODEL ---
import joblib
import json

def save_model_artifacts(model, scaler=None, selector=None, feature_names=None,
                         results=None, model_name="financial_distress_predictor"):
    """
    Save all model artifacts for deployment
    """
    # Save final model
    joblib.dump(model, f'{model_name}.pkl')

    if scaler is not None:
        joblib.dump(scaler, f'{model_name}_scaler.pkl')

    if selector is not None:
        joblib.dump(selector, f'{model_name}_selector.pkl')

    # Save feature names
    if feature_names is not None:
        with open(f'{model_name}_feature_names.json', 'w') as f:
```

```
            json.dump(list(feature_names), f)

    # Save performance results
    if results is not None:
        if isinstance(results, dict):
            results_df = pd.DataFrame.from_dict(results, orient='index')
        else:
            results_df = results
        results_df.to_csv(f'{model_name}_performance_results.csv')

    print(f"Model artifacts saved with prefix: {model_name}")

# --- COMPLETE WORKFLOW EXAMPLE ---
# Assuming you have already trained models and selected the best one
if 'trained_models' in locals() and 'results' in locals():
    # Get the best model name from results
    best_model_name = results.index[0]  # Assuming results is a DataFrame sorted by
    best_model = trained_models[best_model_name]

    print(f"Selected best model: {best_model_name}")
    print(f"Performance: ROC AUC = {results.loc[best_model_name, 'roc_auc']:.3f}")

    # Prepare final model on all data (train + test)
    # Check if X_train and X_test are DataFrames or arrays
    if hasattr(X_train, 'columns'):
        # They are DataFrames
        X_all = pd.concat([X_train, X_test])
        y_all = pd.concat([y_train, y_test])
        feature_names = X_all.columns
    else:
        # They are numpy arrays
        X_all = np.vstack([X_train, X_test])
        y_all = np.concatenate([y_train, y_test])
        # You need to provide feature names if you want to see feature importance
        feature_names = None  # Or provide a list of feature names if you have them

    final_model = prepare_final_model(X_all, y_all, best_model, feature_names=featu

    # Save all artifacts
    save_model_artifacts(
        model=final_model,
        scaler=scaler if 'scaler' in locals() else None,
        selector=selector if 'selector' in locals() else None,
        feature_names=feature_names,
        results=results
    )

    print("=== ANALYSIS COMPLETE ===")
    print(f"Best model: {best_model_name} with ROC AUC: {results.loc[best_model_nam
else:
    print("Error: trained_models or results not found. Please run the training code
```

```
Selected best model: random_forest
Performance: ROC AUC = 0.831
Top 20 most important features:
```

	feature	importance
3	re_ta	0.041276
14	re_ta_lag1	0.038644
26	current_ratio_volatility	0.038080
18	int_cov_lag1	0.036584
11	debt	0.036550
21	debt_lag1	0.035997
20	cfo_lag1	0.034639
9	inv_turn	0.034318
10	cfo	0.033982
6	roe_act	0.033912
15	ebit_ta_lag1	0.033833
25	debt_cap_volatility	0.033722
0	market_cap	0.032899
19	inv_turn_lag1	0.032719
23	roa_act_volatility	0.032532
29	log_market_cap	0.032505
24	roe_act_volatility	0.032402
4	ebit_ta	0.032309
22	debt_growth	0.032309
13	wc_ta_lag1	0.032236




```
Model artifacts saved with prefix: financial_distress_predictor
=== ANALYSIS COMPLETE ===
Best model: random_forest with ROC AUC: 0.831
```