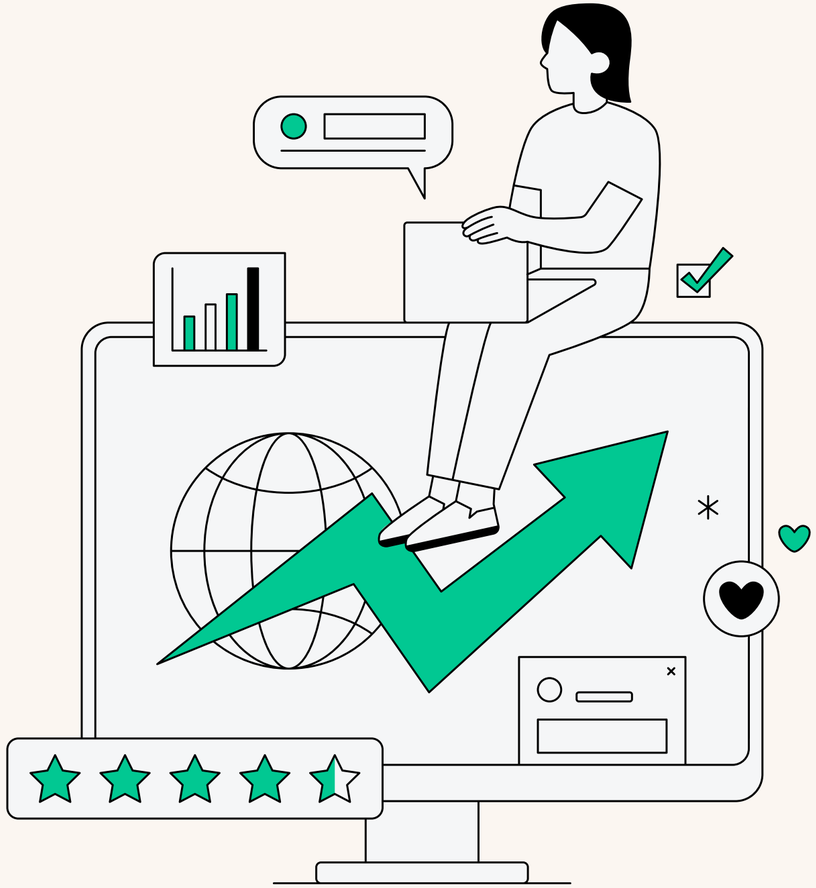
Sc1015 Mini-project

Bankruptcy Prediction

Axalya Raffa Aurelia Jazuli
Owen Lim Qi Xun
Yap Zhi Ling



Motivation

- Predicting bankruptcy is crucial for assessing creditworthiness and managing financial risk

- Accurate predictions enable banks and creditors to adjust lending terms and reduce default risks.

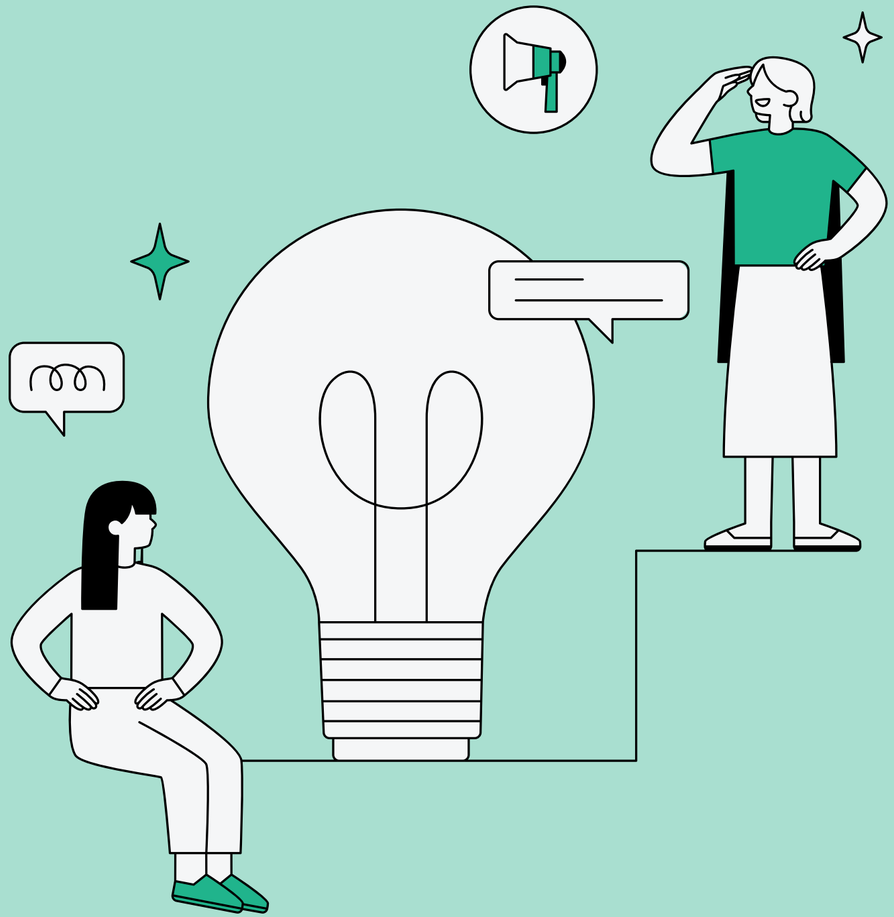
Problem Definition

Which variable best predicts bankruptcy?

Our Dataset


FEDESORIANO · UPDATED 4 YEARS AGO
757
Company Bankruptcy Prediction
Bankruptcy data from the Taiwan Economic Journal for the years 1999–2009
Data Card
Code (230)
Discussion (15)
Suggestions (0)
About Dataset

Basic EDA & Data Preparation



1. Vital Statistics
2. Correlation
3. Selection of Features
4. Extract the Selected Features

1. Vital Statistics

- The dataset contains 6,819 data points and 96 variables
- The target variable, ‘Bankrupt?’, is categorical with two distinct values: 0 and 1
- With 95 features available, we aim to reduce the number of features for more focused analysis

```
# Check the vital statistics of the dataset
print("Data type : ", type(data))
print("Data dims : ", data.shape)
```

```
Data type :  <class 'pandas.core.frame.DataFrame'>
Data dims :  (6819, 96)
```

`Bankrupt?` may be a categorical variable so check below

[11]:

```
# Count the number of unique values
num_unique = data['Bankrupt?'].nunique()
print("Number of unique values:", num_unique)

# Check unique values in the 'Bankrupt?' column
unique_values = data['Bankrupt?'].unique()
print("Unique values in 'Bankrupt?' column:", unique_values)
```

```
Number of unique values: 2
Unique values in 'Bankrupt?' column: [1 0]
```

2. Correlations

- We analyze the correlations between the 95 features and the target variable, ‘Bankrupt?’
- The top 10 most correlated features are printed

```python
# Compute absolute correlations with 'Bankrupt?'
correlations = data.corr()['Bankrupt?'].abs().sort_values(ascending=False)

# Print all correlation values
print("Correlations of all variables with 'Bankrupt?':\n", correlations)
```

```python
# Observe the top 10 correlated features (excluding 'Bankrupt?')
topFeatures = correlations.index[1:11]
topCorrelations = correlations[1:11]  # Get their correlation values

# Print the top 10 correlated features with values
print("\nTop 10 correlated features with 'Bankrupt?':")
for feature, value in zip(topFeatures, topCorrelations):
    print(f"{feature}: {value:.4f}")
```

Top 10 correlated features with 'Bankrupt?':

- Net Income to Total Assets: 0.3155
- ROA(A) before interest and % after tax: 0.2829
- ROA(B) before interest and depreciation after tax: 0.2731
- ROA(C) before interest and depreciation before interest: 0.2608
- Net worth/Assets: 0.2502
- Debt ratio %: 0.2502
- Persistent EPS in the Last Four Seasons: 0.2196
- Retained Earnings to Total Assets: 0.2178
- Net profit before tax/Paid-in capital: 0.2079
- Per Share Net profit before tax (Yuan ¥): 0.2014

Selected Features

1. Net Income to Total Assets
2. ROA(A) before interest and % after tax
3. Debt ratio %
4. Net worth/ Assets
5. Persistent EPS in the Last Four Season
6. Retained Earnings to Total Assets

3. Selection of 6 Features

1.Correlation Strength

- The six selected variables are among the top correlated features with ‘Bankrupt?’
- Their correlations range from 0.2178 to 0.3155, indicating a moderate positive linear correlation with bankruptcy.

2. Economic & Financial Interpretability

3. Selection of 6 Features

2. Economic & Financial Interpretability

- Net Income to Total Assets: Measures overall profitability relative to assets
- ROA (A) before interest and after tax: Shows how efficiently assets generate profit
- Debt Ratio (%): Indicates leverage and financial risk
- Net Worth to Assets: Reflects financial stability
- Persistent EPS over the Last Four Seasons: Earnings per share over time signal long-term profitability trends
- Retained Earnings to Total Assets: Represents accumulated profits

4. Extract the Selected Features

- We extracted ‘Bankrupt?’ and the six selected features
- The columns were renamed for easier manipulation in future steps

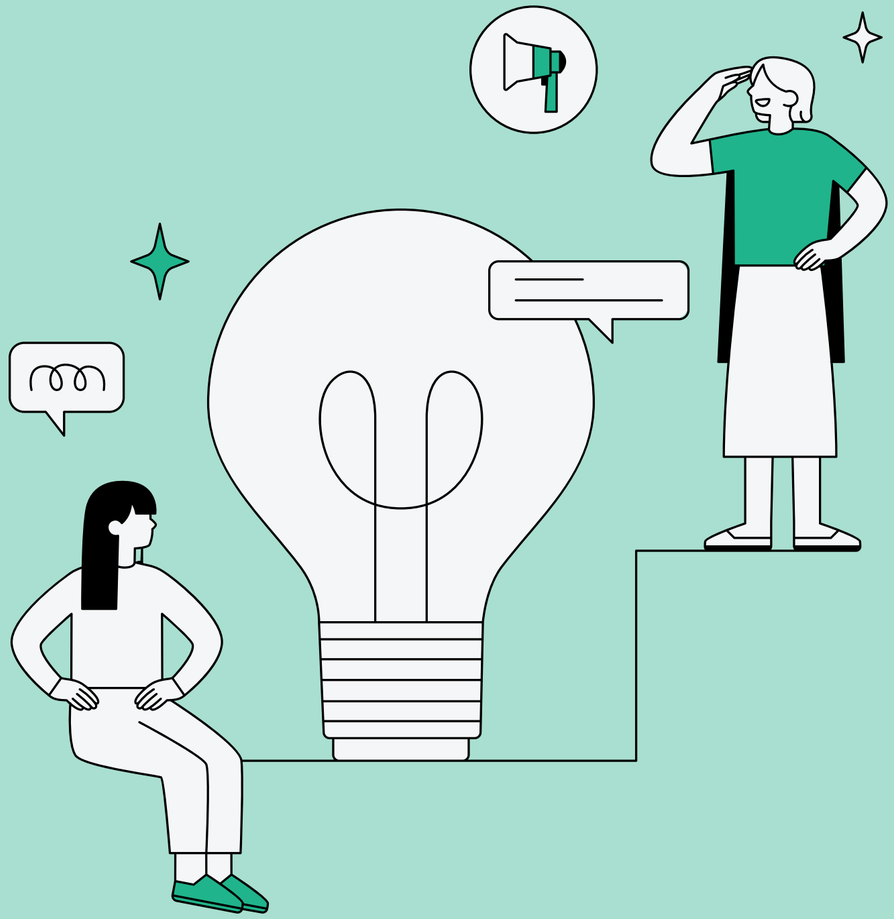
```
# Strip leading/ trailing spaces from column names
data.columns = data.columns.str.strip()

# List of the original column names
columns_to_extract = ['Bankrupt?', 'Net Income to Total Assets',
                      'ROA(A) before interest and % after tax',
                      'Debt ratio %', 'Net worth/Assets',
                      'Persistent EPS in the Last Four Seasons',
                      'Retained Earnings to Total Assets']

# Extract the selected columns
df = data[columns_to_extract]

# Rename the columns to simpler names
df = df.rename(columns={
    'Bankrupt?': 'Bankrupt?',
    'Net Income to Total Assets': 'NetIncome_to_Assets',
    'ROA(A) before interest and % after tax': 'ROA_A',
    'Debt ratio %': 'DebtRatio',
    'Net worth/Assets': 'NetWorth_to_Assets',
    'Persistent EPS in the Last Four Seasons': 'EPS',
    'Retained Earnings to Total Assets': 'Earnings_to_Assets'
})
```

EDA on Selected Features



1. Summary Statistics
2. Distribution of ‘Bankrupt?”
3. Box Plots
4. Outliers
5. Export Cleaned Dataset

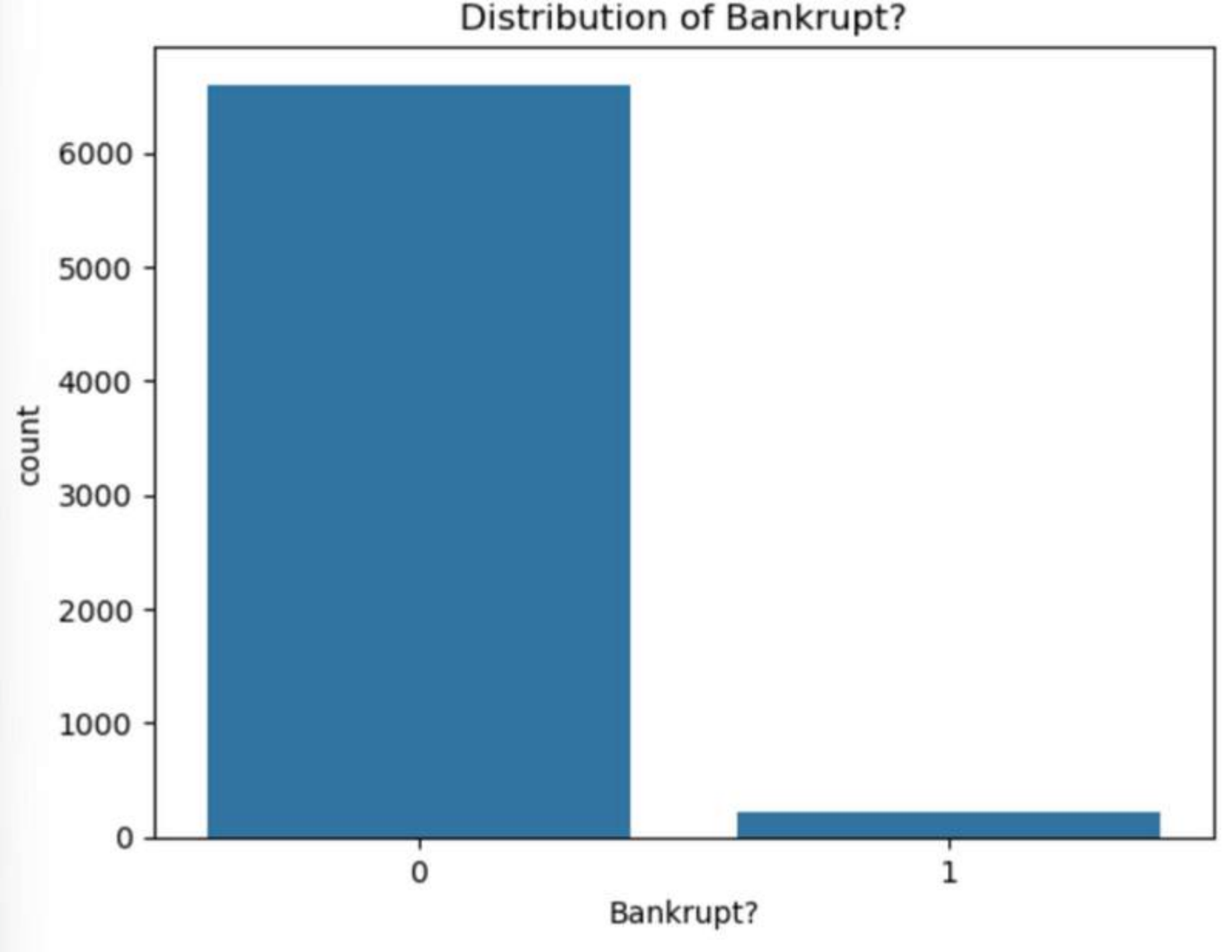
1.Summary Statistics

```
print("Summary Statistics:")
print(df.describe())
```

```
Summary Statistics:
         Bankrupt?  NetIncome_to_Assets        ROA_A    DebtRatio  \
count  6819.000000          6819.000000  6819.000000  6819.000000
mean      0.032263             0.807760     0.558625     0.113177
std       0.176710             0.040332     0.065620     0.053920
min       0.000000             0.000000     0.000000     0.000000
25%       0.000000             0.796750     0.535543     0.072891
50%       0.000000             0.810619     0.559802     0.111407
75%       0.000000             0.826455     0.589157     0.148804
max       1.000000             1.000000     1.000000     1.000000

       NetWorth_to_Assets          EPS  Earnings_to_Assets
count         6819.000000  6819.000000         6819.000000
mean             0.886823     0.228813            0.934733
std              0.053920     0.033263            0.025564
min              0.000000     0.000000            0.000000
25%              0.851196     0.214711            0.931097
50%              0.888593     0.224544            0.937672
75%              0.927109     0.238820            0.944811
max              1.000000     1.000000            1.000000
```

2. Distribution of ‘Bankrupt?’

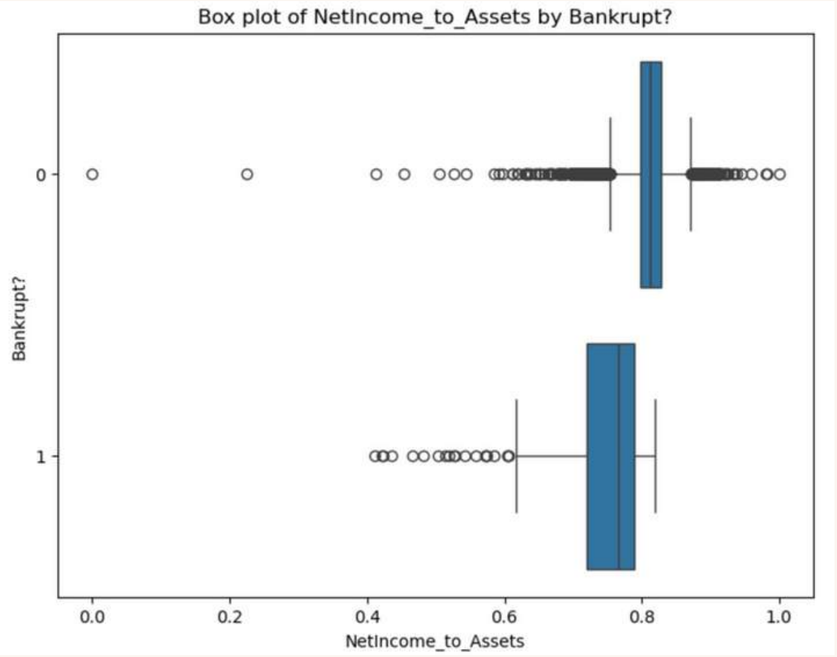


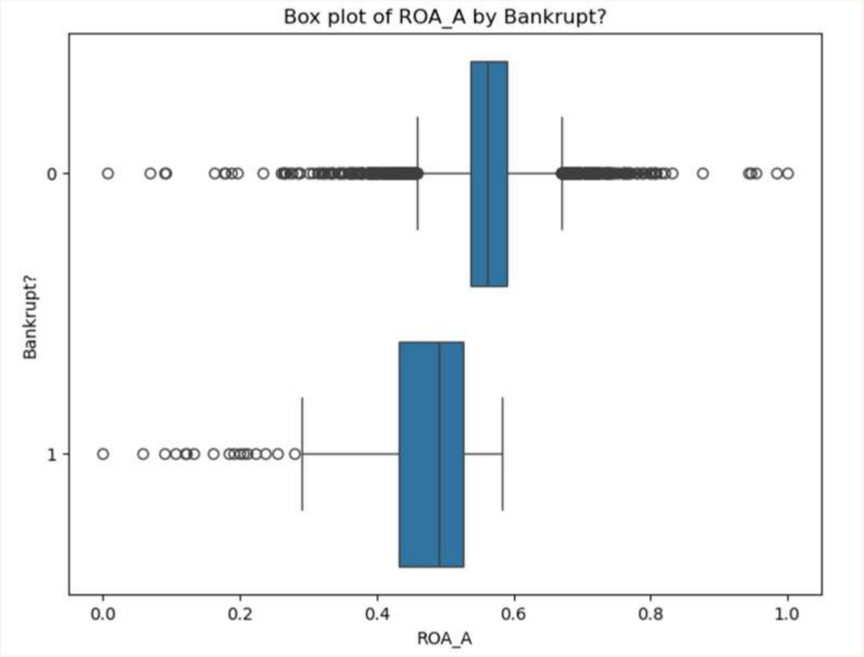
Imbalance in ‘Bankrupt?’ Variable

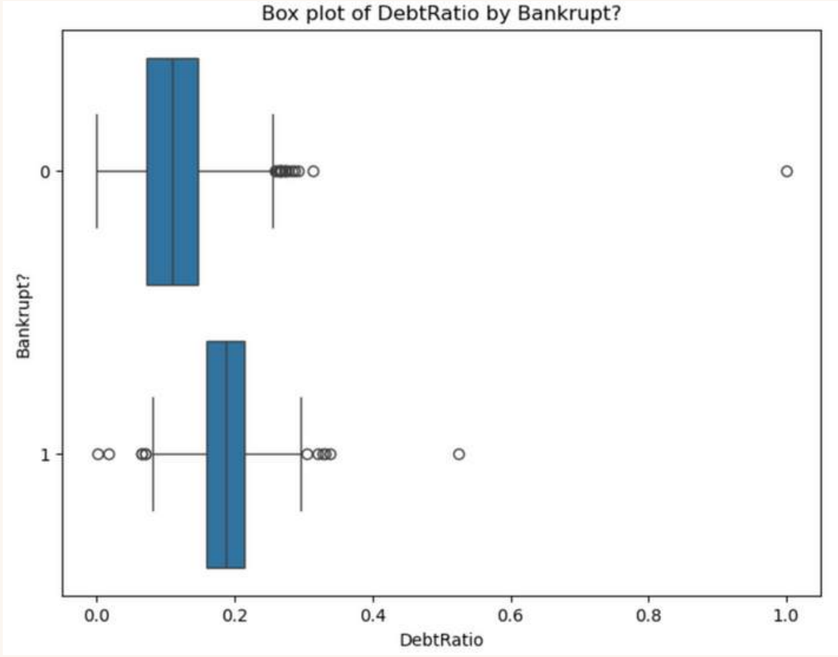
- The dataset shows a significant class imbalance, with a much larger number of non-bankrupt cases compared to bankrupt cases
- This imbalance could impact the performance of the machine learning models
- Therefore, it is crucial to address this issue moving forward

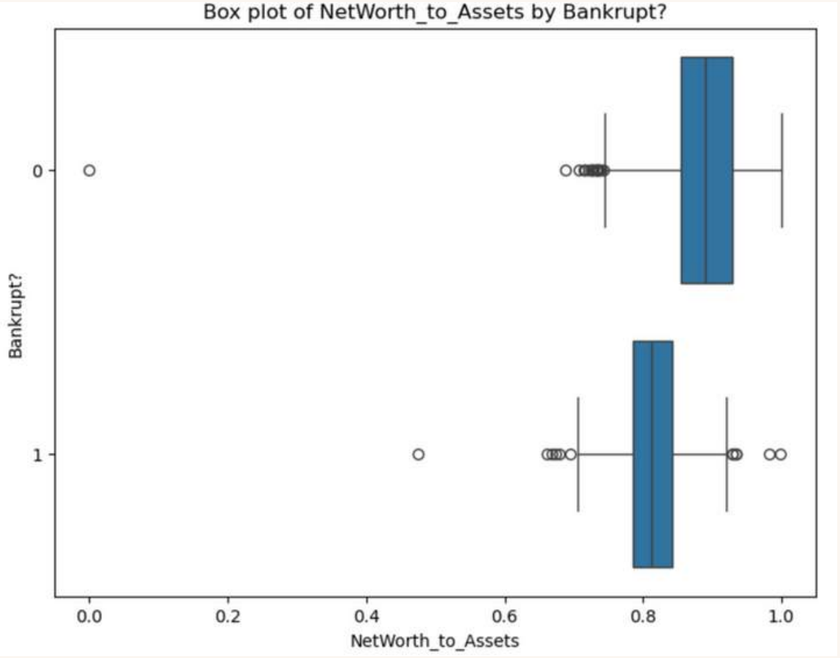
```
Distribution of Bankrupt:
Bankrupt?
0    6599
1     220
Name: count, dtype: int64
```

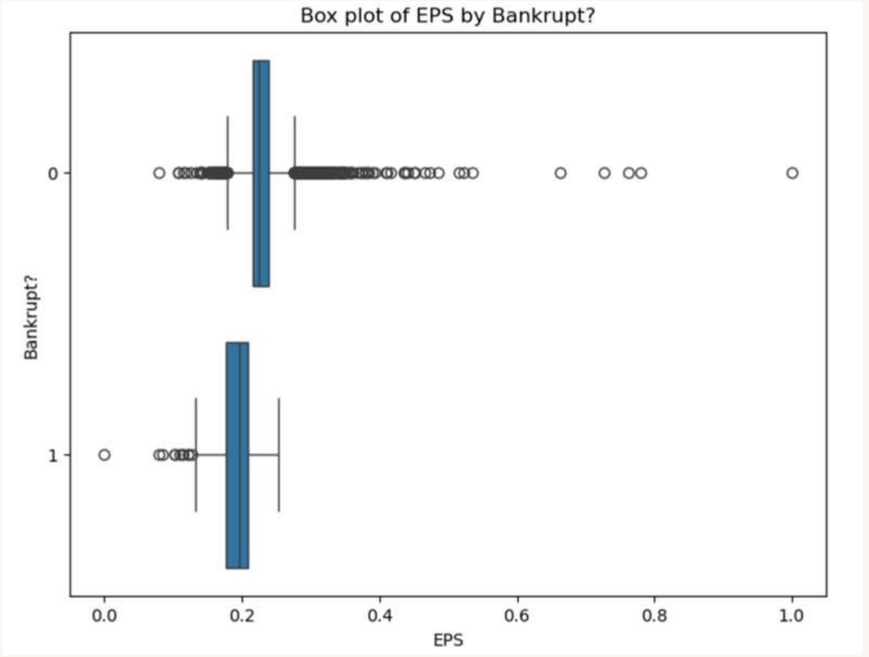
3. Box Plots

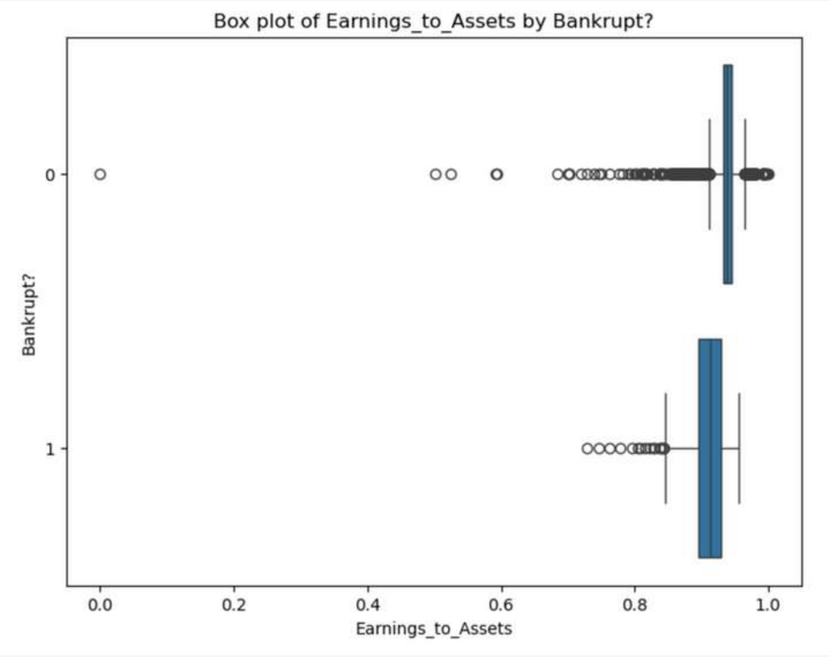

Box plot of NetIncome_to_Assets by Bankrupt?
Bankrupt?
0
1
0.0
0.2
0.4
0.6
0.8
1.0
NetIncome_to_Assets


Box plot of ROA_A by Bankrupt?
Bankrupt?
0
1
0.0
0.2
0.4
0.6
0.8
1.0
ROA_A


Box plot of DebtRatio by Bankrupt?
Bankrupt?
0
1
0.0
0.2
0.4
0.6
0.8
1.0
DebtRatio


Box plot of NetWorth_to_Assets by Bankrupt?
Bankrupt?
0
1
0.0
0.2
0.4
0.6
0.8
1.0
NetWorth_to_Assets


Box plot of EPS by Bankrupt?
Bankrupt?
0
1
0.0
0.2
0.4
0.6
0.8
1.0
EPS


Box plot of Earnings_to_Assets by Bankrupt?
Bankrupt?
0
1
0.0
0.2
0.4
0.6
0.8
1.0
Earnings_to_Assets

4. Outliers

```python
# Define function to detect and print number of outliers using IQR
def detectOutliers(df, features):
    outlier_indices = set()

    print("\nOutlier count per feature:")
    for feature in features:
        Q1 = df[feature].quantile(0.25)
        Q3 = df[feature].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Get outlier indices for this feature
        feature_outliers = df[(df[feature] < lower_bound) | (df[feature] > upper_bound)].index
        outlier_indices.update(feature_outliers)

        # Print number of outliers detected for this feature
        print(f" {feature}: {len(feature_outliers)}")

    return list(outlier_indices)
```

```python
# Detect and print outliers per feature
outlierIndices = detectOutliers(df, numericalFeatures)
```

Outlier count per feature:

- NetIncome_to_Assets: 561
- ROA_A: 561
- DebtRatio: 30
- NetWorth_to_Assets: 30
- EPS: 508
- Earnings_to_Assets: 633

Removing Outliers

- Outliers can distort the relationships in the data, making it harder for the model to identify meaningful patterns

- Removing them helps the model learn more accurate relationships, ultimately improving its performance

```
# Remove outliers from dataset
dfCleaned = df.drop(index=outlierIndices).reset_index(drop=True)

# Show shape before and after removing outliers
print(f"Original dataset shape: {df.shape}")
print(f"Dataset shape after removing outliers: {dfCleaned.shape}")
```

```
Original dataset shape: (6819, 7)
Dataset shape after removing outliers: (5731, 7)
```

5. Export Cleaned Dataset

```
dfCleaned.to_csv('finalData.csv', index=False)
print("\nCleaned dataset exported as 'finalData.csv'")
```

```
Cleaned dataset exported as 'finalData.csv'
```

Machine Learning Models

Metrics

- **Class 1 Precision**: How many flagged companies actually went bankrupt? (Avoid false alarms)
- **Class 1 Recall**: How many real bankruptcies did we catch? (Avoid missing risks)
- **AUC-PR**: Overall balance between the two (higher = better).

Logistic Regression

- Predicts the probability that an instance belongs to a particular class
- Use a threshold to make a binary prediction

Class weights

- Assign different weights for different classes
- Modify cost function

Synthetic Minority Over-sampling (SMOTE)

- Addresses class imbalance by creating synthetic samples of the minority class

Synthetic Minority Over-sampling Technique & Edited Nearest Neighbors (SMOTE-ENN)

- SMOTE creates synthetic examples of the minority class
- ENN looks at K-Nearest Neighbours and eliminates noisy examples

Youden's J Statistic

- Captures both sensitivity and specificity:

$$J = \text{Sensitivity} + \text{Specificity} - 1$$

- Equally weighs sensitivity and specificity, avoiding bias toward the majority class

Evaluations

Models	Class 1 Precision	Class 1 Recall	AUC-PR
Baseline	0.000	0.000	0.113
with class weights	0.080	0.810	0.119
with SMOTE	0.080	0.740	0.120
with SMOTE-ENN	0.070	0.840	0.119
with Youden's J statistic	0.060	0.900	0.113
with SMOTE & class weights & Youden's J stats	0.060	0.900	0.120

XGBoost

How does it work?

XGBoost

Works by combining multiple weak leaners, typically decision trees, to create a strong predictive model. It improves predictions iteratively by minimising errors from previous trees.

scale_pos_weight

- Improve the model by adjusting the class weights using `scale_pos_weight` parameter in XGBoost

- This adjusts the model to place more weight on the minority class (bankrupt cases), pushing it to focus more on correctly classifying bankruptcies

4 Variations

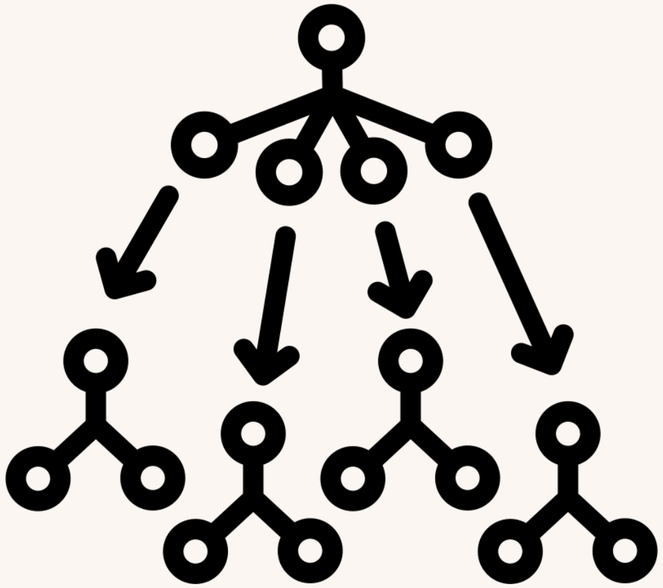
1. scale_pos_weight
2. scale_pos_weight + SMOTE
3. scale_pos_weight + SMOTE-ENN
4. scale_pos_weight + SMOTE-ENN + Youden's J

Evaluation of XGBoost

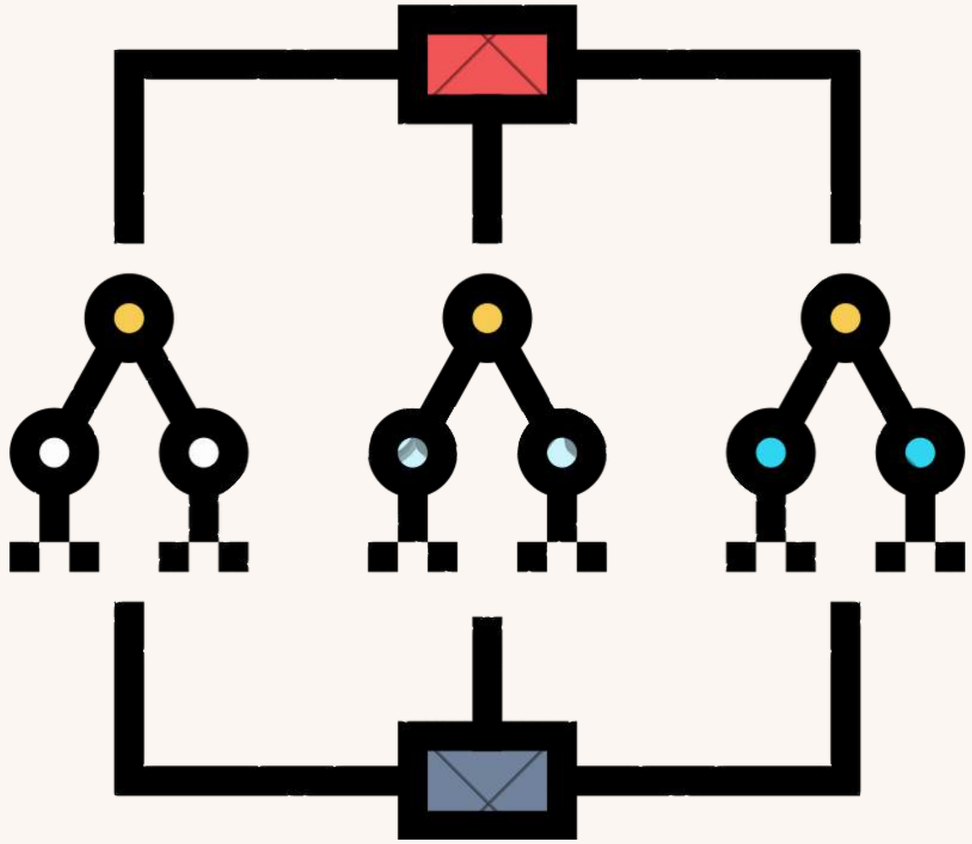
Variation	Precision	Recall	AUC-PR
Baseline	0.0000	0.0000	0.0835
Var 1	0.1333	0.0800	0.1018
Var 2	0.0886	0.2800	0.1148
Var 3	0.1148	0.5600	0.1327
Var 4	0.0872	0.7600	0.1327

Best Model: scale_pos_weight + SMOTE-ENN + Youden's J

Random Forest Classifier



How does it work?



We modified the base random forest classifier to produce 4 variations

We utilised SMOTE-ENN, Class Weights and tuned the model using 2 different indicators

1. Random Forest with SMOTE-ENN
2. Random Forest with SMOTE-ENN and Class Weights
3. andom Forest with SMOTEENN, Class Weighting, and Threshold Tuning (Precision-Recall Curve)
4. Random Forest with SMOTEENN, Class Weighting, and Threshold Tuning (Youden's J statistic)

Evaluation of Random Forest Classifier

```
=== Performance Comparison Across Variations ===
      Metric  Baseline  Variation 1  Variation 2  Variation 3  Variation 4
0  Precision  0.000000     0.076923     0.073333     0.114286     0.073394
1     Recall  0.000000     0.523810     0.523810     0.380952     0.761905
2     AUC-PR  0.122458     0.087609     0.085077     0.085077     0.085077
```

Best Model is (Variation 4) Random Forest with SMOTEENN, Class Weighting, and Threshold Tuning (Youden's J statistic)

Evaluation of the 3 Machine learning methods.

Metrics used:

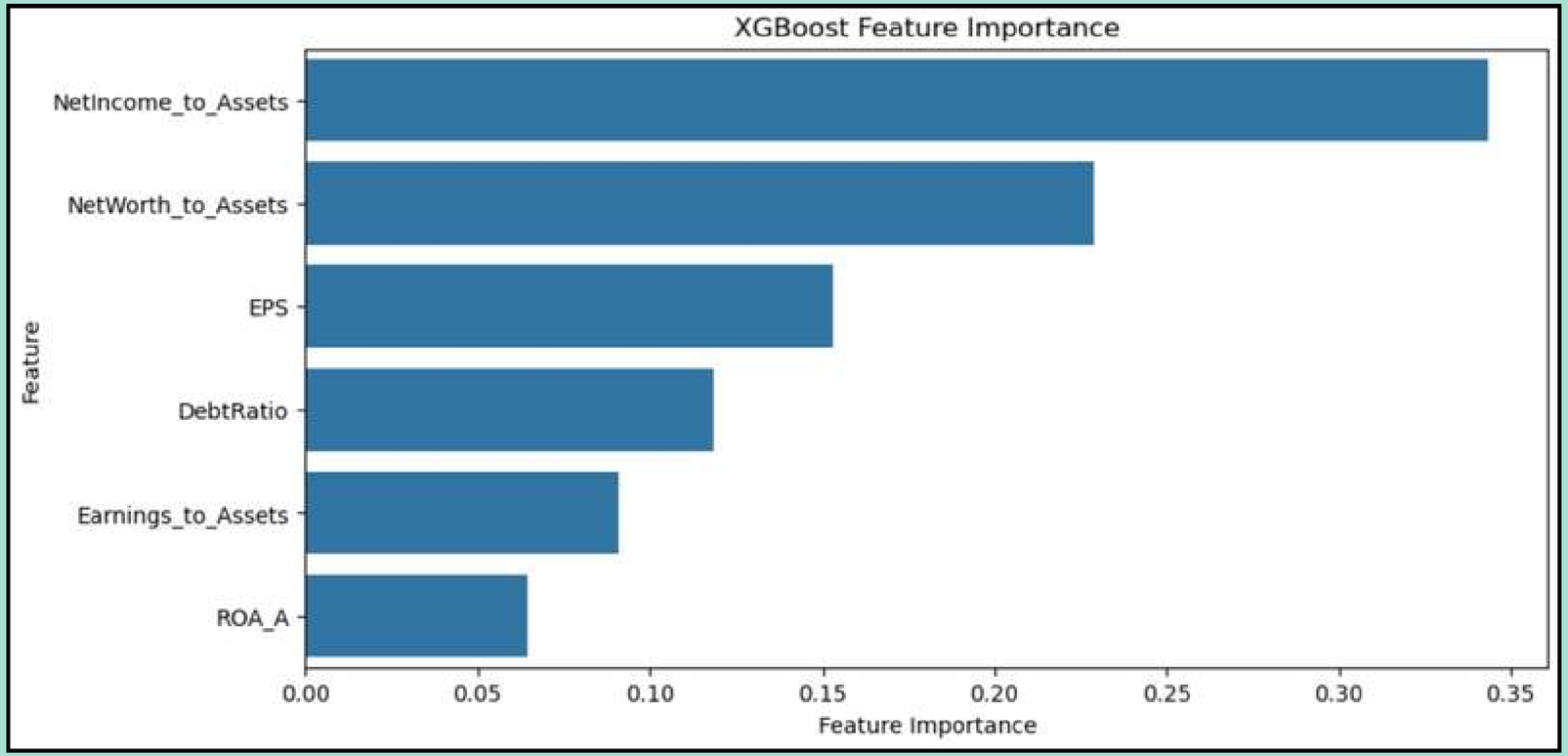
- Precision
- Recall
- AUC-PR

Prioritised Recall and AUC-PR

Machine Learning Model	Best Variation	Precision	Recall	AUC-PR
XGBoost	Variation 4	0.0872	0.76	0.1327
Logistic Regression	Logistic Regression with SMOTE & class weights & Youden's J stats	0.06	0.9	0.12
Random Forest	Random Forest with SMOTEENN, Class Weighting, & Youden's J stats	0.073394	0.761905	0.085077

The best model is XGBoost(Variation 4)

Feature importance



Thank You!