BAX 452: Machine Learning

Final Project: Classifying Company Bankruptcy

Alexa Aguirre, Bernard Arambula, Yumi Jin

Exploratory Data Analysis (EDA)

In [ ]:
```
#Loading all packages
import pandas as pd
import scipy
import seaborn as sns
import statsmodels.api as sm
import statsmodels.formula.api as smf
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor, plot_tree
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, StratifiedKFold, cross
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LassoCV
import statsmodels.api as sm
from statsmodels.tools import add_constant
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.inspection import permutation_importance
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, confusion_matrix, confusi
import os
```

In [ ]:
```
# get working directory
os.getcwd()
```

Out[ ]:
```
'/Users/alexaaguirre/Library/Mobile Documents/com~apple~CloudDocs/BAX 452'
```

In [ ]:
```
#load data
df=pd.read_csv("company.csv")
df.head()
```

Out[ ]:

	Bankrupt?	ROA(C) before interest and depreciation before interest	ROA(A) before interest and % after tax	ROA(B) before interest and depreciation after tax	Operating Gross Margin	Realized Sales Gross Margin	Operating Profit Rate	Pre-t n Intere Ra
0	1	0.370594	0.424389	0.405750	0.601457	0.601457	0.998969	0.7968
1	1	0.464291	0.538214	0.516730	0.610235	0.610235	0.998946	0.7973
2	1	0.426071	0.499019	0.472295	0.601450	0.601364	0.998857	0.7964
3	1	0.399844	0.451265	0.457733	0.583541	0.583541	0.998700	0.7969
4	1	0.465022	0.538432	0.522298	0.598783	0.598783	0.998973	0.7973

5 rows × 96 columns

In [ ]:
```
df.shape
```

Out[ ]:
```
(6819, 96)
```

We have 6,819 rows and 96 columns of data

In [ ]:
```
#viewing column names, integer types and null values
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6819 entries, 0 to 6818
Data columns (total 96 columns):
 #   Column                                                  Non-Null Cou
nt  Dtype
---  ------                                                  -------------
--  -----
 0   Bankrupt?                                               6819 non-nul
l   int64
 1    ROA(C) before interest and depreciation before interest  6819 non-nul
l   float64
 2    ROA(A) before interest and % after tax                  6819 non-nul
l   float64
 3    ROA(B) before interest and depreciation after tax       6819 non-nul
l   float64
 4    Operating Gross Margin                                  6819 non-nul
l   float64
 5    Realized Sales Gross Margin                             6819 non-nul
l   float64
 6    Operating Profit Rate                                   6819 non-nul
l   float64
 7    Pre-tax net Interest Rate                               6819 non-nul
l   float64
 8    After-tax net Interest Rate                             6819 non-nul
l   float64
 9    Non-industry income and expenditure/revenue             6819 non-nul
l   float64
 10   Continuous interest rate (after tax)                    6819 non-nul
l   float64
 11   Operating Expense Rate                                  6819 non-nul
l   float64
 12   Research and development expense rate                   6819 non-nul
l   float64
 13   Cash flow rate                                          6819 non-nul
l   float64
 14   Interest-bearing debt interest rate                     6819 non-nul
l   float64
 15   Tax rate (A)                                            6819 non-nul
l   float64
 16   Net Value Per Share (B)                                 6819 non-nul
l   float64
 17   Net Value Per Share (A)                                 6819 non-nul
l   float64
 18   Net Value Per Share (C)                                 6819 non-nul
l   float64
 19   Persistent EPS in the Last Four Seasons                 6819 non-nul
l   float64
 20   Cash Flow Per Share                                     6819 non-nul
l   float64
 21   Revenue Per Share (Yuan ¥)                              6819 non-nul
l   float64
 22   Operating Profit Per Share (Yuan ¥)                     6819 non-nul
l   float64
 23   Per Share Net profit before tax (Yuan ¥)                6819 non-nul
l   float64
 24   Realized Sales Gross Profit Growth Rate                 6819 non-nul
```

```
l   float64
 25   Operating Profit Growth Rate                           6819 non-nul
l   float64
 26   After-tax Net Profit Growth Rate                       6819 non-nul
l   float64
 27   Regular Net Profit Growth Rate                         6819 non-nul
l   float64
 28   Continuous Net Profit Growth Rate                      6819 non-nul
l   float64
 29   Total Asset Growth Rate                                6819 non-nul
l   float64
 30   Net Value Growth Rate                                  6819 non-nul
l   float64
 31   Total Asset Return Growth Rate Ratio                   6819 non-nul
l   float64
 32   Cash Reinvestment %                                    6819 non-nul
l   float64
 33   Current Ratio                                          6819 non-nul
l   float64
 34   Quick Ratio                                            6819 non-nul
l   float64
 35   Interest Expense Ratio                                 6819 non-nul
l   float64
 36   Total debt/Total net worth                             6819 non-nul
l   float64
 37   Debt ratio %                                           6819 non-nul
l   float64
 38   Net worth/Assets                                       6819 non-nul
l   float64
 39   Long-term fund suitability ratio (A)                   6819 non-nul
l   float64
 40   Borrowing dependency                                   6819 non-nul
l   float64
 41   Contingent liabilities/Net worth                       6819 non-nul
l   float64
 42   Operating profit/Paid-in capital                       6819 non-nul
l   float64
 43   Net profit before tax/Paid-in capital                  6819 non-nul
l   float64
 44   Inventory and accounts receivable/Net value            6819 non-nul
l   float64
 45   Total Asset Turnover                                   6819 non-nul
l   float64
 46   Accounts Receivable Turnover                           6819 non-nul
l   float64
 47   Average Collection Days                                6819 non-nul
l   float64
 48   Inventory Turnover Rate (times)                        6819 non-nul
l   float64
 49   Fixed Assets Turnover Frequency                        6819 non-nul
l   float64
 50   Net Worth Turnover Rate (times)                        6819 non-nul
l   float64
 51   Revenue per person                                     6819 non-nul
l   float64
 52   Operating profit per person                            6819 non-nul
```

```
l   float64
 53   Allocation rate per person                              6819 non-nul
l   float64
 54   Working Capital to Total Assets                         6819 non-nul
l   float64
 55   Quick Assets/Total Assets                               6819 non-nul
l   float64
 56   Current Assets/Total Assets                             6819 non-nul
l   float64
 57   Cash/Total Assets                                       6819 non-nul
l   float64
 58   Quick Assets/Current Liability                          6819 non-nul
l   float64
 59   Cash/Current Liability                                  6819 non-nul
l   float64
 60   Current Liability to Assets                             6819 non-nul
l   float64
 61   Operating Funds to Liability                            6819 non-nul
l   float64
 62   Inventory/Working Capital                               6819 non-nul
l   float64
 63   Inventory/Current Liability                             6819 non-nul
l   float64
 64   Current Liabilities/Liability                           6819 non-nul
l   float64
 65   Working Capital/Equity                                  6819 non-nul
l   float64
 66   Current Liabilities/Equity                              6819 non-nul
l   float64
 67   Long-term Liability to Current Assets                   6819 non-nul
l   float64
 68   Retained Earnings to Total Assets                       6819 non-nul
l   float64
 69   Total income/Total expense                              6819 non-nul
l   float64
 70   Total expense/Assets                                    6819 non-nul
l   float64
 71   Current Asset Turnover Rate                             6819 non-nul
l   float64
 72   Quick Asset Turnover Rate                               6819 non-nul
l   float64
 73   Working capitcal Turnover Rate                          6819 non-nul
l   float64
 74   Cash Turnover Rate                                      6819 non-nul
l   float64
 75   Cash Flow to Sales                                      6819 non-nul
l   float64
 76   Fixed Assets to Assets                                  6819 non-nul
l   float64
 77   Current Liability to Liability                          6819 non-nul
l   float64
 78   Current Liability to Equity                             6819 non-nul
l   float64
 79   Equity to Long-term Liability                           6819 non-nul
l   float64
 80   Cash Flow to Total Assets                               6819 non-nul
```

```
l   float64
 81   Cash Flow to Liability                                 6819 non-nul
l   float64
 82   CFO to Assets                                          6819 non-nul
l   float64
 83   Cash Flow to Equity                                    6819 non-nul
l   float64
 84   Current Liability to Current Assets                    6819 non-nul
l   float64
 85   Liability-Assets Flag                                  6819 non-nul
l   int64
 86   Net Income to Total Assets                             6819 non-nul
l   float64
 87   Total assets to GNP price                              6819 non-nul
l   float64
 88   No-credit Interval                                     6819 non-nul
l   float64
 89   Gross Profit to Sales                                  6819 non-nul
l   float64
 90   Net Income to Stockholder's Equity                     6819 non-nul
l   float64
 91   Liability to Equity                                    6819 non-nul
l   float64
 92   Degree of Financial Leverage (DFL)                     6819 non-nul
l   float64
 93   Interest Coverage Ratio (Interest expense to EBIT)     6819 non-nul
l   float64
 94   Net Income Flag                                        6819 non-nul
l   int64
 95   Equity to Liability                                    6819 non-nul
l   float64
dtypes: float64(93), int64(3)
memory usage: 5.0 MB
```

All data types are numeric(float or integer) and we have no missing values in our data set.

In [ ]:
```
#count, mean, standard deviation, 5 number summary of all numeric columns
df.describe().T
```

Out[ ]:

	count	mean	std	min	25%	50%	75%	max
Bankrupt?	6819.0	0.032263	0.176710	0.0	0.000000	0.000000	0.000000	1.0
ROA(C) before interest and depreciation before interest	6819.0	0.505180	0.060686	0.0	0.476527	0.502706	0.535563	1.0
ROA(A) before interest and % after tax	6819.0	0.558625	0.065620	0.0	0.535543	0.559802	0.589157	1.0
ROA(B) before interest and depreciation after tax	6819.0	0.553589	0.061595	0.0	0.527277	0.552278	0.584105	1.0
Operating Gross Margin	6819.0	0.607948	0.016934	0.0	0.600445	0.605997	0.613914	1.0
...	...	...	...	...	...	...	...	...
Liability to Equity	6819.0	0.280365	0.014463	0.0	0.276944	0.278778	0.281449	1.0
Degree of Financial Leverage (DFL)	6819.0	0.027541	0.015668	0.0	0.026791	0.026808	0.026913	1.0
Interest Coverage Ratio (Interest expense to EBIT)	6819.0	0.565358	0.013214	0.0	0.565158	0.565252	0.565725	1.0
Net Income Flag	6819.0	1.000000	0.000000	1.0	1.000000	1.000000	1.000000	1.0
Equity to Liability	6819.0	0.047578	0.050014	0.0	0.024477	0.033798	0.052838	1.0

96 rows × 8 columns

Here we can examine the mean, standard deviation, and the 5 number summary for every variable in our dataset.

In [ ]:

```
#Pearson Correlation Test

num_cols=df.select_dtypes(include=[np.number]).columns.tolist()
col_list=[]

for col in num_cols:
    if col != 'Banrupt?':
        x,y=col, 'Bankrupt?'

        dtf_noNan=df[df[x].notnull()]

        if x in dtf_noNan.columns and y in dtf_noNan.columns:
            coeff, p=scipy.stats.pearsonr(dtf_noNan[x],dtf_noNan[y])
            coeff, p=round(coeff, 3), round(p,3)

            if coeff>0:
                conclusion="Positive correlation"
```

```
            elif coeff<0:
                conclusion="Negative correlation"
            else:
                conclusion="No correlation"

            if p<0.05:
                print ("Pearson Correlation:", col, coeff, conclusion, "(p-v
                col_list.append(col)
```

Pearson Correlation: Bankrupt? 1.0 Positive correlation (p-value: 0.0)
Pearson Correlation: ROA(C) before interest and depreciation before interest -0.261 Negative correlation (p-value: 0.0)
Pearson Correlation: ROA(A) before interest and % after tax -0.283 Negative correlation (p-value: 0.0)
Pearson Correlation: ROA(B) before interest and depreciation after tax -0.273 Negative correlation (p-value: 0.0)
Pearson Correlation: Operating Gross Margin -0.1 Negative correlation (p-value: 0.0)
Pearson Correlation: Realized Sales Gross Margin -0.099 Negative correlation (p-value: 0.0)
Pearson Correlation: Research and development expense rate -0.024 Negative correlation (p-value: 0.045)
Pearson Correlation: Cash flow rate -0.072 Negative correlation (p-value: 0.0)
Pearson Correlation: Tax rate (A) -0.11 Negative correlation (p-value: 0.0)
Pearson Correlation: Net Value Per Share (B) -0.165 Negative correlation (p-value: 0.0)
Pearson Correlation: Net Value Per Share (A) -0.165 Negative correlation (p-value: 0.0)
Pearson Correlation: Net Value Per Share (C) -0.165 Negative correlation (p-value: 0.0)
Pearson Correlation: Persistent EPS in the Last Four Seasons -0.22 Negative correlation (p-value: 0.0)
Pearson Correlation: Cash Flow Per Share -0.078 Negative correlation (p-value: 0.0)
Pearson Correlation: Operating Profit Per Share (Yuan ¥) -0.142 Negative correlation (p-value: 0.0)
Pearson Correlation: Per Share Net profit before tax (Yuan ¥) -0.201 Negative correlation (p-value: 0.0)
Pearson Correlation: After-tax Net Profit Growth Rate -0.038 Negative correlation (p-value: 0.002)
Pearson Correlation: Regular Net Profit Growth Rate -0.037 Negative correlation (p-value: 0.002)
Pearson Correlation: Total Asset Growth Rate -0.044 Negative correlation (p-value: 0.0)
Pearson Correlation: Net Value Growth Rate 0.065 Positive correlation (p-value: 0.0)
Pearson Correlation: Cash Reinvestment % -0.051 Negative correlation (p-value: 0.0)
Pearson Correlation: Quick Ratio 0.025 Positive correlation (p-value: 0.039)
Pearson Correlation: Debt ratio % 0.25 Positive correlation (p-value: 0.0)
Pearson Correlation: Net worth/Assets -0.25 Negative correlation (p-value: 0.0)
Pearson Correlation: Borrowing dependency 0.177 Positive correlation (p-value: 0.0)
Pearson Correlation: Contingent liabilities/Net worth 0.07 Positive correlation (p-value: 0.0)
Pearson Correlation: Operating profit/Paid-in capital -0.141 Negative correlation (p-value: 0.0)
Pearson Correlation: Net profit before tax/Paid-in capital -0.208 Negative correlation (p-value: 0.0)
Pearson Correlation: Inventory and accounts receivable/Net value 0.075 Positive correlation (p-value: 0.0)

Pearson Correlation: Total Asset Turnover -0.068 Negative correlation (p-value: 0.0)
Pearson Correlation: Fixed Assets Turnover Frequency 0.073 Positive correlation (p-value: 0.0)
Pearson Correlation: Revenue per person 0.04 Positive correlation (p-value: 0.001)
Pearson Correlation: Operating profit per person -0.093 Negative correlation (p-value: 0.0)
Pearson Correlation: Working Capital to Total Assets -0.193 Negative correlation (p-value: 0.0)
Pearson Correlation: Quick Assets/Total Assets -0.086 Negative correlation (p-value: 0.0)
Pearson Correlation: Current Assets/Total Assets -0.045 Negative correlation (p-value: 0.0)
Pearson Correlation: Cash/Total Assets -0.1 Negative correlation (p-value: 0.0)
Pearson Correlation: Cash/Current Liability 0.078 Positive correlation (p-value: 0.0)
Pearson Correlation: Current Liability to Assets 0.194 Positive correlation (p-value: 0.0)
Pearson Correlation: Operating Funds to Liability -0.077 Negative correlation (p-value: 0.0)
Pearson Correlation: Working Capital/Equity -0.147 Negative correlation (p-value: 0.0)
Pearson Correlation: Current Liabilities/Equity 0.154 Positive correlation (p-value: 0.0)
Pearson Correlation: Retained Earnings to Total Assets -0.218 Negative correlation (p-value: 0.0)
Pearson Correlation: Total expense/Assets 0.139 Positive correlation (p-value: 0.0)
Pearson Correlation: Quick Asset Turnover Rate 0.026 Positive correlation (p-value: 0.033)
Pearson Correlation: Fixed Assets to Assets 0.066 Positive correlation (p-value: 0.0)
Pearson Correlation: Current Liability to Equity 0.154 Positive correlation (p-value: 0.0)
Pearson Correlation: Equity to Long-term Liability 0.139 Positive correlation (p-value: 0.0)
Pearson Correlation: Cash Flow to Total Assets -0.07 Negative correlation (p-value: 0.0)
Pearson Correlation: Cash Flow to Liability -0.043 Negative correlation (p-value: 0.0)
Pearson Correlation: CFO to Assets -0.115 Negative correlation (p-value: 0.0)
Pearson Correlation: Cash Flow to Equity -0.059 Negative correlation (p-value: 0.0)
Pearson Correlation: Current Liability to Current Assets 0.171 Positive correlation (p-value: 0.0)
Pearson Correlation: Liability-Assets Flag 0.139 Positive correlation (p-value: 0.0)
Pearson Correlation: Net Income to Total Assets -0.315 Negative correlation (p-value: 0.0)
Pearson Correlation: Total assets to GNP price 0.035 Positive correlation (p-value: 0.004)
Pearson Correlation: Gross Profit to Sales -0.1 Negative correlation (p-value: 0.0)

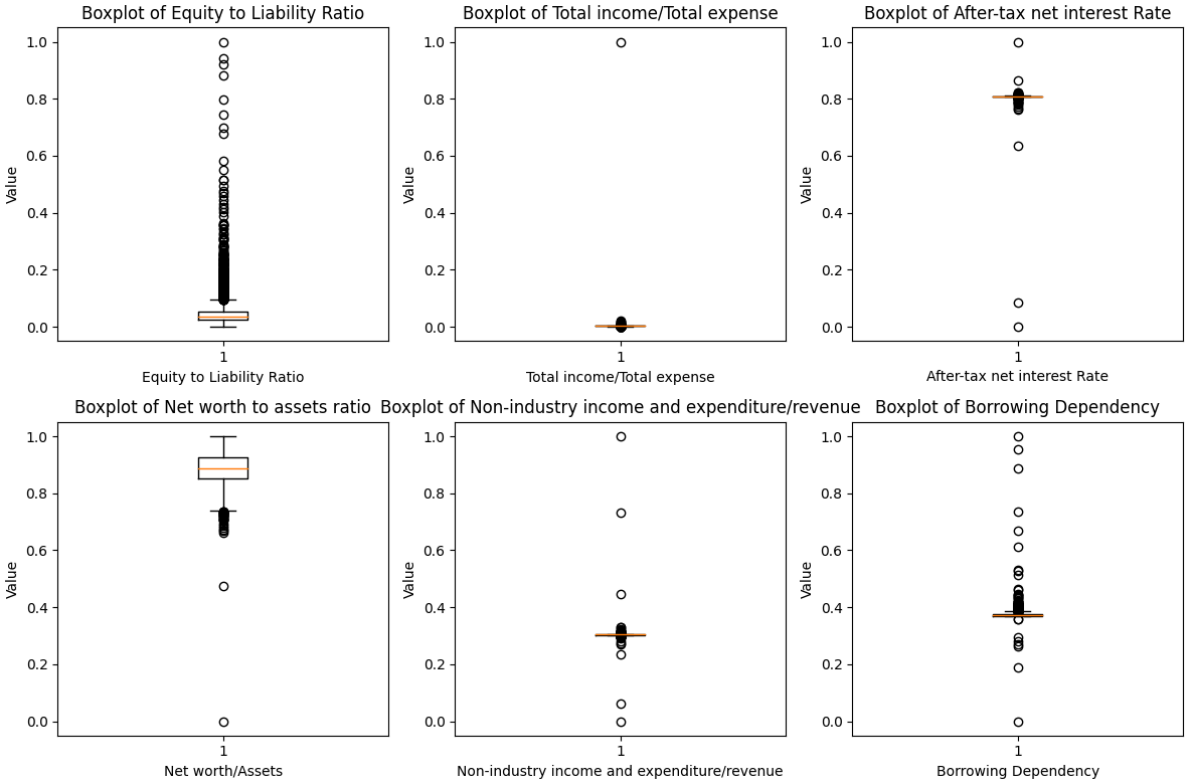
```
Pearson Correlation:  Net Income to Stockholder's Equity -0.181 Negative co
rrelation (p-value: 0.0)
Pearson Correlation:  Liability to Equity 0.167 Positive correlation (p-val
ue: 0.0)
Pearson Correlation:  Equity to Liability -0.083 Negative correlation (p-va
lue: 0.0)
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/scipy/stats/_stats_py.py:4424: ConstantInputWarning: An input array is
constant; the correlation coefficient is not defined.
  warnings.warn(stats.ConstantInputWarning(msg))
```

We examine the correlations between each variable and bankruptcy. Here we can see which variables have a positive and negative correlation with our outcome variable.

In [ ]:

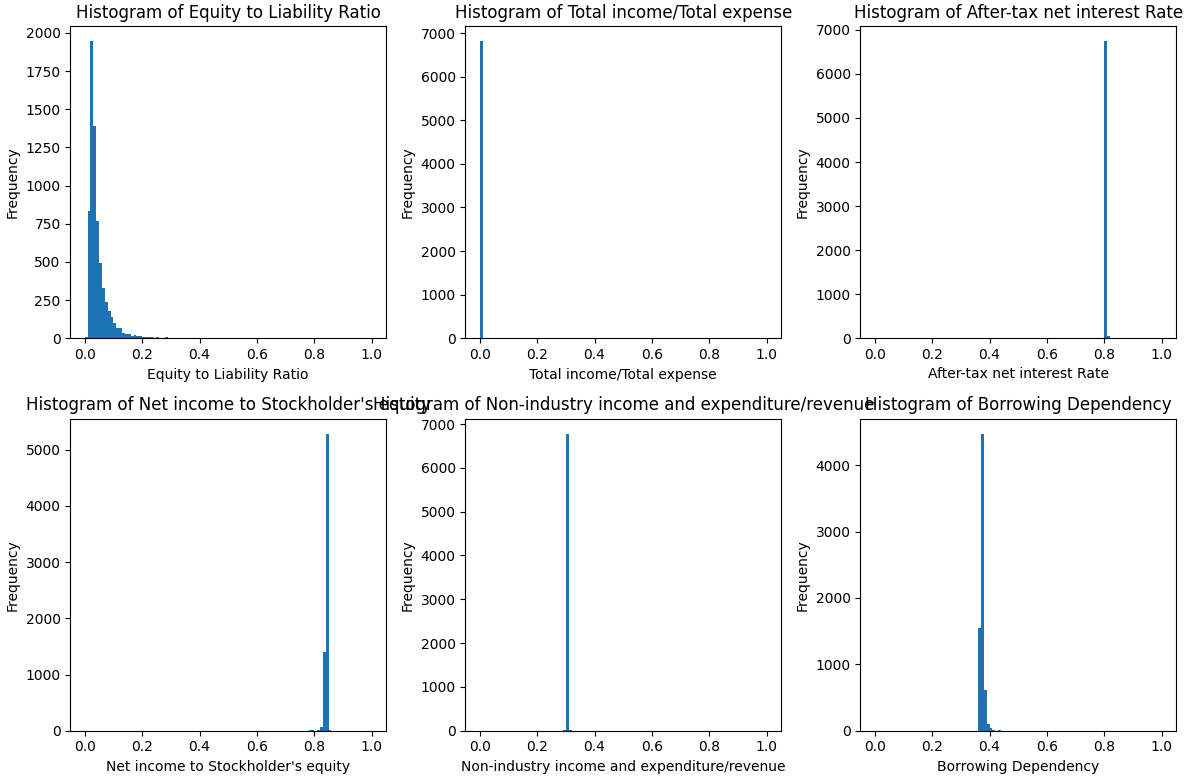
```python
import matplotlib.pyplot as plt

# Create subplots with a 2x3 layout
fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(12, 8))

# Create the first boxplot
axes[0, 0].boxplot(df[' Equity to Liability'])
axes[0, 0].set_xlabel('Equity to Liability Ratio')
axes[0, 0].set_ylabel('Value')
axes[0, 0].set_title('Boxplot of Equity to Liability Ratio')

# Create the second boxplot
axes[0, 1].boxplot(df[' Total income/Total expense'])
axes[0, 1].set_xlabel('Total income/Total expense')
axes[0, 1].set_ylabel('Value')
axes[0, 1].set_title('Boxplot of Total income/Total expense')

# Create the third boxplot
axes[0, 2].boxplot(df[' After-tax net Interest Rate'])
axes[0, 2].set_xlabel('After-tax net interest Rate')
axes[0, 2].set_ylabel('Value')
axes[0, 2].set_title('Boxplot of After-tax net interest Rate')

# Create the fourth boxplot
axes[1, 0].boxplot(df[' Net worth/Assets'])
axes[1, 0].set_xlabel('Net worth/Assets')
axes[1, 0].set_ylabel('Value')
axes[1, 0].set_title('Boxplot of Net worth to assets ratio')

# Create the fifth boxplot
axes[1, 1].boxplot(df[' Non-industry income and expenditure/revenue'])
axes[1, 1].set_xlabel('Non-industry income and expenditure/revenue')
axes[1, 1].set_ylabel('Value')
axes[1, 1].set_title('Boxplot of Non-industry income and expenditure/revenue

# Create the sixth boxplot
axes[1, 2].boxplot(df[' Borrowing dependency'])
axes[1, 2].set_xlabel('Borrowing Dependency')
axes[1, 2].set_ylabel('Value')
axes[1, 2].set_title('Boxplot of Borrowing Dependency')
```

```
# Adjust layout
plt.tight_layout()

# Show the plot
plt.show()
```



In [ ]:
```
# Create subplots with a 2x3 layout
fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(12, 8))

# Create histograms for each variable
axes[0, 0].hist(df[' Equity to Liability'], bins = 100)
axes[0, 0].set_xlabel('Equity to Liability Ratio')
axes[0, 0].set_ylabel('Frequency')
axes[0, 0].set_title('Histogram of Equity to Liability Ratio')

axes[0, 1].hist(df[' Total income/Total expense'], bins = 100)
axes[0, 1].set_xlabel('Total income/Total expense')
axes[0, 1].set_ylabel('Frequency')
axes[0, 1].set_title('Histogram of Total income/Total expense')

axes[0, 2].hist(df[' After-tax net Interest Rate'], bins = 100)
axes[0, 2].set_xlabel('After-tax net interest Rate')
axes[0, 2].set_ylabel('Frequency')
axes[0, 2].set_title('Histogram of After-tax net interest Rate')

axes[1, 0].hist(df[" Net Income to Stockholder's Equity"], bins = 100)  # Ha
axes[1, 0].set_xlabel("Net income to Stockholder's equity")
axes[1, 0].set_ylabel('Frequency')
axes[1, 0].set_title("Histogram of Net income to Stockholder's equity")  # H

axes[1, 1].hist(df[' Non-industry income and expenditure/revenue'], bins = 1
axes[1, 1].set_xlabel('Non-industry income and expenditure/revenue')
```

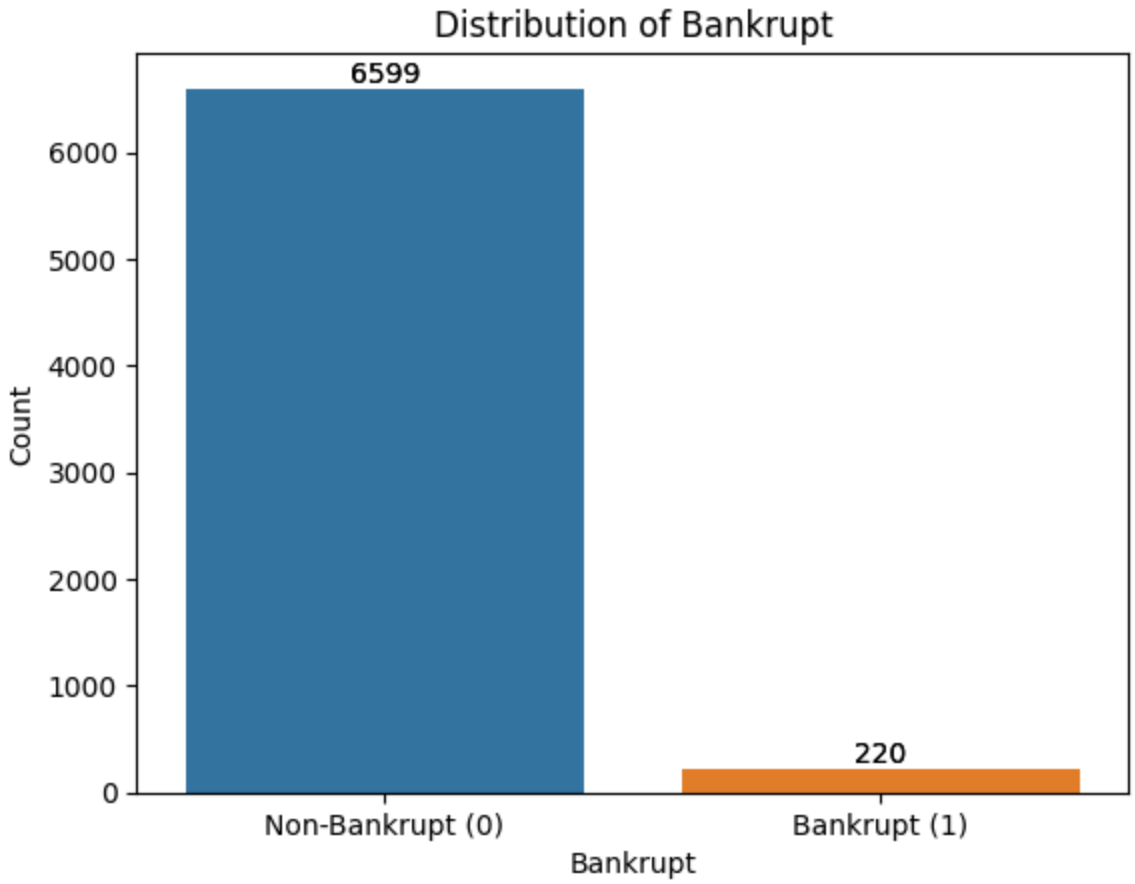
```
axes[1, 1].set_ylabel('Frequency')
axes[1, 1].set_title('Histogram of Non-industry income and expenditure/rever

axes[1, 2].hist(df[' Borrowing dependency'], bins = 100)
axes[1, 2].set_xlabel('Borrowing Dependency')
axes[1, 2].set_ylabel('Frequency')
axes[1, 2].set_title('Histogram of Borrowing Dependency')

# Adjust layout
plt.tight_layout()

# Show the plot
plt.show()
```



Here we don't see very much variation in the data, with selected variable having a very centralized distribution.

In [ ]:

```
# distibution of bankruptcy
label = sns.countplot(data=df, x='Bankrupt?')
sns.countplot(data=df, x='Bankrupt?')
plt.xlabel('Bankrupt')
plt.ylabel('Count')
plt.title('Distribution of Bankrupt')
plt.xticks(ticks=[0, 1], labels=['Non-Bankrupt (0)', 'Bankrupt (1)'])  # Set

# Annotate count values on top of each bar
for p in label.patches:
    label.text(p.get_x() + p.get_width()/2., p.get_height(), f'{p.get_height

plt.show()
```



Here we can see that there are 220 entries in the dataset that have gone bankrupt while 6,599 have not. This imbalance may lead to lower predictive power in some models. Models that are more robust to class imbalance such as random forests may provide better predictions.

Logistic Regression Model Development

We will begin our analysis by fitting the data to a logistic regression model to predict if a company will go bankrupt based off of their attributes. We will use the full model with all of the variables to initially fit our model.

In [ ]:
```
#Full model

X=df.drop('Bankrupt?', axis=1)
y=df[['Bankrupt?']]

# add a constant
X = sm.add_constant(X)

# split test 20% and train 80%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, r

# Fit logistic regression model with standardized values and balancing class
scaler = StandardScaler()
```

```
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('logistic_regression', LogisticRegression(class_weight='balanced'))
])
pipeline.fit(X_train, y_train)
# Transform X_train and X_test using the fitted scaler
X_train_scaled = pipeline.named_steps['scaler'].transform(X_train)
X_test_scaled = pipeline.named_steps['scaler'].transform(X_test)
```

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

In [ ]:
```
#predictions on the test set
pipeline_y_pred = pipeline.predict(X_test)

# out of sample metrics
# Compute Mean Squared Error
mse = mean_squared_error(y_test, pipeline_y_pred)
print("OOS MSE:", mse)
#oos accuracy
accuracy = accuracy_score(y_test, pipeline_y_pred)
print("OOS Accuracy:", accuracy)
```

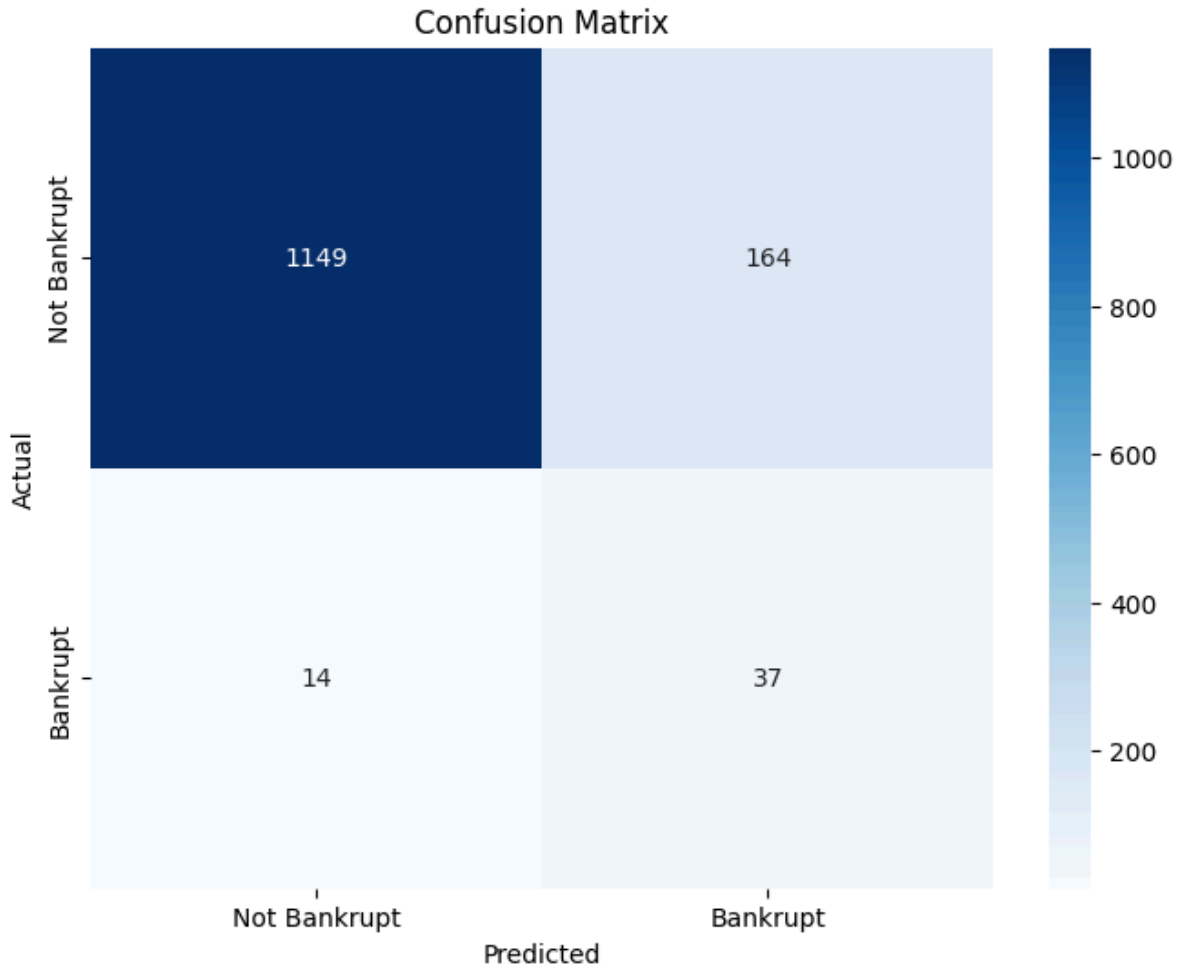
```
OOS MSE: 0.13049853372434017
OOS Accuracy: 0.8695014662756598
```

In [ ]:
```
#out of sample metrics
pipeline_conf_matrix = confusion_matrix(y_test, pipeline_y_pred)
pipeline_class_report = classification_report(y_test, pipeline_y_pred)

print("Classification Report:\n", pipeline_class_report)
print("Confusion Matrix:\n", pipeline_conf_matrix)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.88      0.93      1313
           1       0.18      0.73      0.29        51

    accuracy                           0.87      1364
   macro avg       0.59      0.80      0.61      1364
weighted avg       0.96      0.87      0.90      1364

Confusion Matrix:
 [[1149  164]
 [  14   37]]
```

In [ ]:

```python
# Calculate confusion matrix
conf_matrix = confusion_matrix(y_test, pipeline_y_pred)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
diag_labels = ['Not Bankrupt', 'Bankrupt']
plt.xticks(ticks=[0.5, 1.5], labels=diag_labels)
plt.yticks(ticks=[0.5, 1.5], labels=diag_labels)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



Classification Report Analysis:

- Precision- ratio of true positive predictions to the total number of positive predictions
- Recall (sensitivity)- ratio of true positive preditctions to toal number of actual positive occurances
- F1 - the harmonic mean of precision and recall, higher indicates better performance
- support- number of occurances of each class in the dataset
- accuracy- how correct the model is overall
- macro average- unweighted average of precision, recall, and F-1 score
- weighted average- average of precision, recall, and F-1 score weighted by the influence from each class

Confusion Matrix:

- True Positives (classified as bankrupt and are bankrupt): 37
- True Negatives (classified as not bankrupt and aren't bankrupt): 1149
- False Postives (Classified as bankrupt and aren't bankrupt): 164

- False Negatives (Classified as not bankrupt and are bankrupt): 14

We can see that the overall accuracy of the model out of sample is high at 97% although we can see that the precision, recall, and F-1 scores are significantly lower for predicting those that went bankrupt correctly.

In [ ]:
```
#displaying which variables have the most effect on predicting bankruptcy
coefficients = pipeline.named_steps['logistic_regression'].coef_[0]
coef = pd.DataFrame({'Feature':  X_train.columns, 'Coefficient': coefficient
coef = coef.sort_values(by='Coefficient', ascending=False)
coef
```

Out[ ]:

	Feature	Coefficient
39	Borrowing dependency	1.628462
41	Operating profit/Paid-in capital	1.284512
36	Debt ratio %	1.064730
21	Operating Profit Per Share (Yuan ¥)	1.061312
17	Net Value Per Share (C)	0.781528
...	...	...
94	Equity to Liability	-1.193754
90	Liability to Equity	-1.349637
15	Net Value Per Share (B)	-1.426199
45	Accounts Receivable Turnover	-1.438907
18	Persistent EPS in the Last Four Seasons	-2.150432

95 rows × 2 columns

Interpretation:

Here the coefficients illustrate the increase/decrease in odds of going bankrupt for a unit increase in each feature.

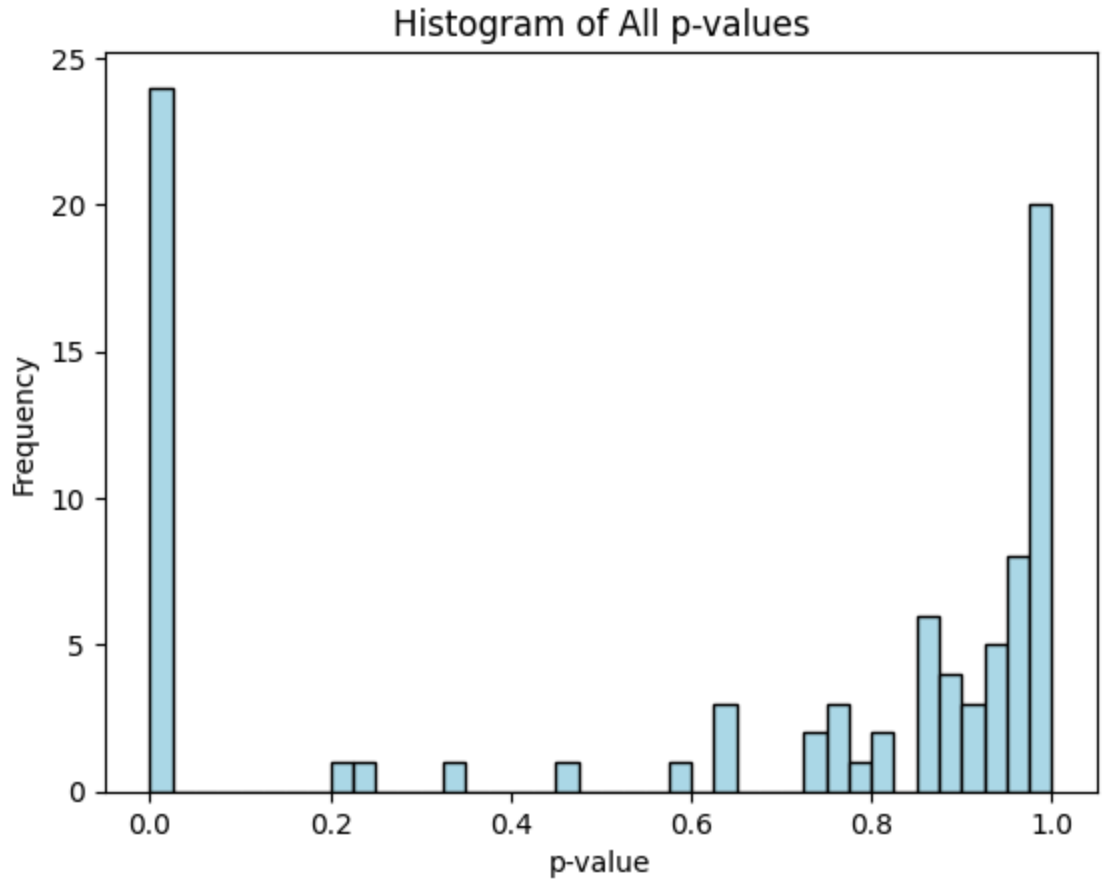
False Discovery Rate Analysis: Benjamini-Hochberg Procedure

We will now use the BH procedure to control for the number of false positives in identifying significant variables to predict bankruptcy.

In [ ]:
```
from sklearn.linear_model import LogisticRegression
import numpy as np
from scipy.stats import norm

# Compute standard errors
```

```
std_err = np.sqrt(np.diag(np.linalg.inv(np.dot(X_train.T, X_train))))

# Compute z-statistics
z_stats = coefficients / std_err

# Compute p-values
pvals = 2 * (1 - norm.cdf(np.abs(z_stats)))

# Plot the histogram of all p-values
plt.hist(pvals, bins=40,color="lightblue", edgecolor="black")
plt.xlabel("p-value")
plt.ylabel("Frequency")
plt.title("Histogram of All p-values")
plt.show()
```

```
/var/folders/tn/vsw4vgdx4fz3r22myj1yg_j80000gn/T/ipykernel_1031/1532341225.
py:6: RuntimeWarning: invalid value encountered in sqrt
  std_err = np.sqrt(np.diag(np.linalg.inv(np.dot(X_train.T, X_train))))
```



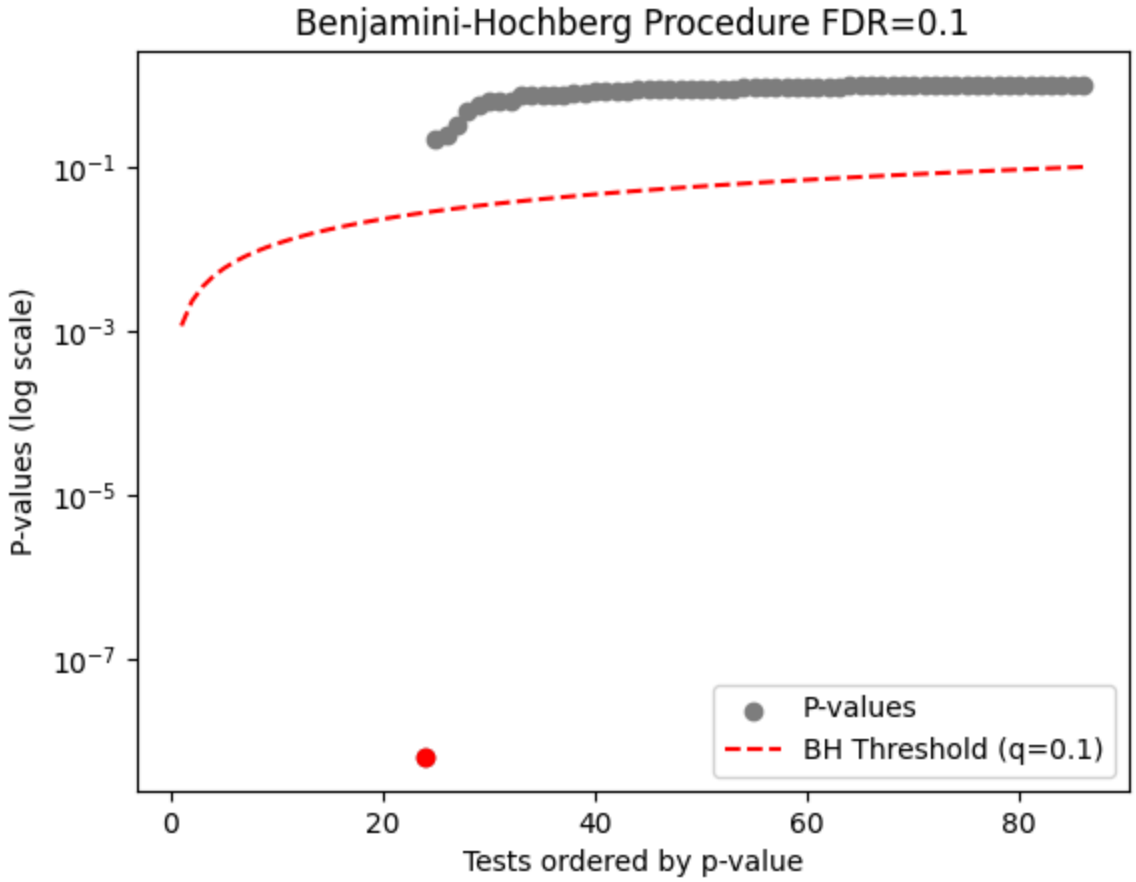
In [ ]:
```
# Estimate the number of true discoveries at q=0.1

# Number of tests
m = len(pvals)

# Desired false discovery rate
q = 0.1

# Sort p-values and compute critical BH threshold
sorted_p_values = np.sort(pvals)
```

```
critical_values = [q * (i+1) / m for i in range(m)]

# Find the largest p-value that is smaller than the BH critical value
significant_p_values_mask = sorted_p_values <= critical_values
max_index = np.where(significant_p_values_mask)[0].max() if significant_p_va
significant_p_values = sorted_p_values[:max_index+1]

# Estimate the number of true discoveries
num_true_discoveries = len(significant_p_values)
print(f"The estimated number of true discoveries when q=0.1 is {num_true_dis
```

```
The estimated number of true discoveries when q=0.1 is 24.
```

In [ ]:

```
def fdr_bh(pvals, q=0.1, plotit=False):
    pvals = np.array(pvals)
    pvals = pvals[~np.isnan(pvals)]  # Remove NaN values
    N = len(pvals)

    # Sort p-values and compute BH critical values
    sorted_pvals = np.sort(pvals)
    ranks = np.arange(1, N+1)
    bh_critical_values = (ranks / N) * q

    # Find the largest p-value where p-value < BH critical value
    below_threshold = sorted_pvals <= bh_critical_values
    max_index = np.where(below_threshold)[0].max() if any(below_threshold) e
    alpha = sorted_pvals[max_index] if max_index != -1 else 0

    if plotit:
        # Plotting
        plt.scatter(np.arange(1, N+1), sorted_pvals, color='grey', label='P-
        if max_index != -1:
            plt.scatter(np.arange(1, max_index+2), sorted_pvals[:max_index+1
        plt.plot(np.arange(1, N+1), bh_critical_values, 'r--', label=f'BH Th
        plt.xlabel('Tests ordered by p-value')
        plt.ylabel('P-values (log scale)')
        plt.yscale('log')
        plt.title('Benjamini-Hochberg Procedure FDR=0.1')
        plt.legend()
        plt.show()

    return alpha


alpha_threshold=fdr_bh(pvals, q = 0.1, plotit = True)
```



In [ ]:
```
# finding the number or true discoveries when q=0.1
true_dis=(pvals<alpha_threshold).sum().sum()
print(f"Number of 'true' discoveries: {true_dis} when q=0.1.")
```

```
Number of 'true' discoveries: 23 when q=0.1.
```

In [ ]:
```
selected_features = X.columns[pvals < alpha_threshold]
selected_features
```

Out[ ]:
```
Index([' Operating Expense Rate', ' Research and development expense rate',
       ' Interest-bearing debt interest rate', ' Revenue Per Share (Yuan
¥)',
       ' Total Asset Growth Rate', ' Net Value Growth Rate', ' Current Rati
o',
       ' Quick Ratio', ' Total debt/Total net worth',
       ' Accounts Receivable Turnover', ' Average Collection Days',
       ' Inventory Turnover Rate (times)', ' Fixed Assets Turnover Frequenc
y',
       ' Revenue per person', ' Allocation rate per person',
       ' Quick Assets/Current Liability', ' Cash/Current Liability',
       ' Inventory/Current Liability',
       ' Long-term Liability to Current Assets',
       ' Current Asset Turnover Rate', ' Quick Asset Turnover Rate',
       ' Cash Turnover Rate', ' Total assets to GNP price'],
      dtype='object')
```

In [ ]:
```
X_reduced = X[selected_features]
logreg = LogisticRegression(max_iter=10000, solver='liblinear')
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
```

```
scores_reduced = cross_val_score(logreg, X_reduced, y, cv=cv, scoring='accur
average_performance_reduced = np.mean(scores_reduced)

X_train_reduced, X_test_reduced, y_train_reduced, y_test_reduced = train_tes

pipeline.fit(X_train_reduced, y_train_reduced)

pipeline_reduced = pipeline.predict(X_test_reduced)
pipeline_reduced_conf_matrix = confusion_matrix(y_test_reduced, pipeline_rec
pipeline_reduced_class_report = classification_report(y_test_reduced, pipeli

print("Reduced Model Cross-Validation Accuracy:", average_performance_reduce
print("Classification Report:\n", pipeline_reduced_class_report)
print("Confusion Matrix:\n", pipeline_reduced_conf_matrix)
```

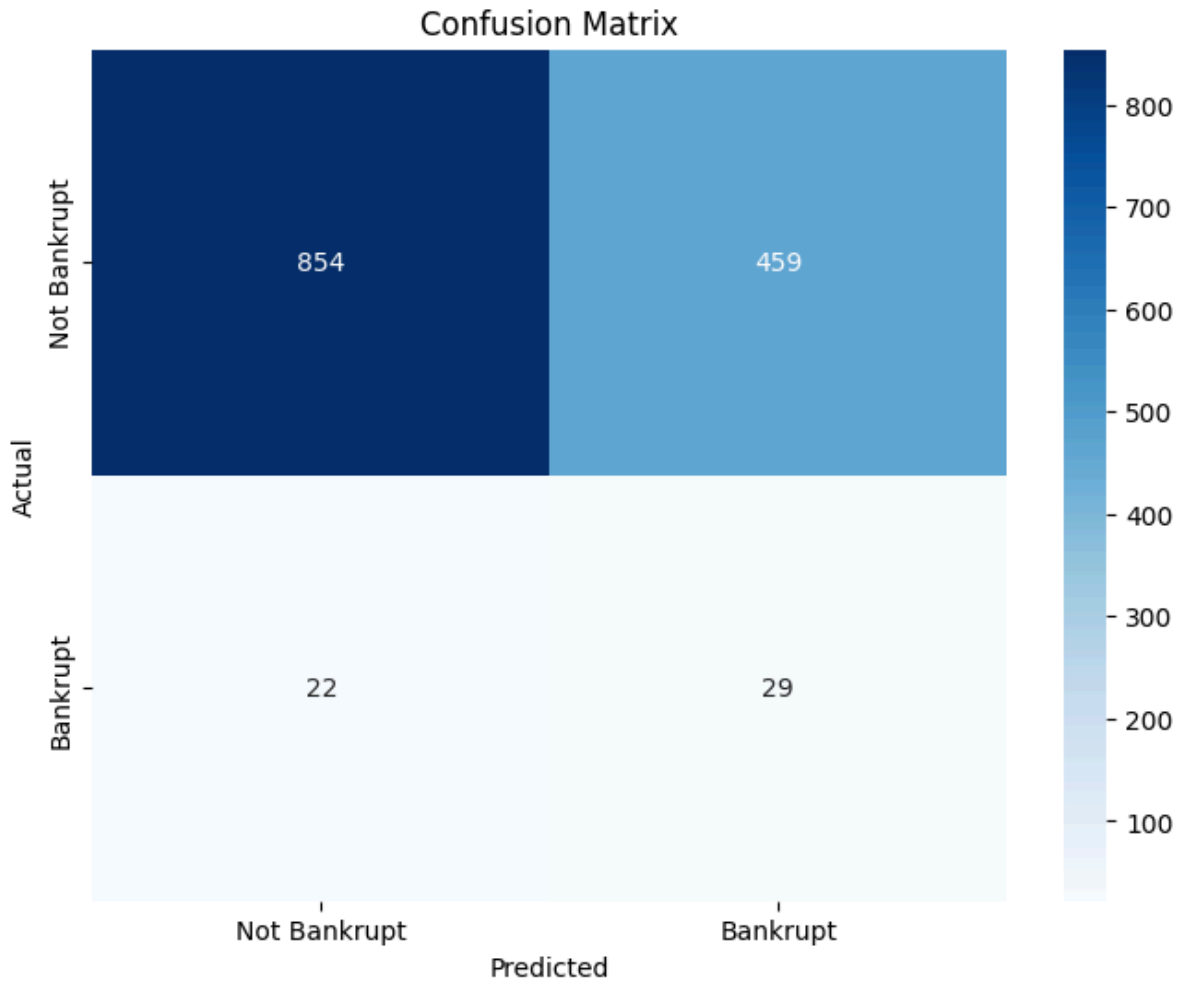
```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vect
or y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vect
or y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vect
or y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vect
or y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vect
or y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vect
or y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vect
or y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vect
or y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vect
or y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

```
Reduced Model Cross-Validation Accuracy: 0.9661240800788903
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.65      0.78      1313
           1       0.06      0.57      0.11        51

    accuracy                           0.65      1364
   macro avg       0.52      0.61      0.44      1364
weighted avg       0.94      0.65      0.76      1364

Confusion Matrix:
 [[854 459]
 [ 22  29]]
```

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [ ]:

```python
# Calculate confusion matrix visual
conf_matrix = confusion_matrix(y_test, pipeline_reduced)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
diag_labels = ['Not Bankrupt', 'Bankrupt']
plt.xticks(ticks=[0.5, 1.5], labels=diag_labels)
plt.yticks(ticks=[0.5, 1.5], labels=diag_labels)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



Confusion Matrix:

- True Positives (classified as bankrupt and are bankrupt): 29
- True Negatives (classified as not bankrupt and aren't bankrupt): 854
- False Postives (Classified as bankrupt and aren't bankrupt): 459
- False Negatives (Classified as not bankrupt and are bankrupt): 22

Here we can see that the reduced model using the significant variables from BH procedure increase the out of sample redictive performance for not bankrupt but decreased for baknrupt. The overall predicitve performance of the model slightly decreased.

Regularization: Lasso CV

In [ ]:
```
from sklearn.linear_model import LogisticRegressionCV
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler
import numpy as np
```

```python
# Assuming you have X_train_scaled and y_train defined
alphas = np.logspace(-6, 6, 13)
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
lasso_cv = LogisticRegressionCV(Cs=alphas, penalty='l1', solver='liblinear',

# Fit the LassoCV model
lasso_cv.fit(X_train_scaled, y_train)

# Get the optimal C value (inverse of alpha) from LassoCV
optimal_C = lasso_cv.C_[0]
optimal_alpha = 1 / optimal_C
print("Optimal Alpha:", optimal_alpha)
```

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vect
or y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to con
verge, increase the number of iterations.
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to con
verge, increase the number of iterations.
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to con
verge, increase the number of iterations.
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to con
verge, increase the number of iterations.
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to con
verge, increase the number of iterations.
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to con
verge, increase the number of iterations.
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to con
verge, increase the number of iterations.
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to con
verge, increase the number of iterations.
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to con
verge, increase the number of iterations.
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to con
verge, increase the number of iterations.
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to con
verge, increase the number of iterations.
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to con
verge, increase the number of iterations.
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to con
```

```
verge, increase the number of iterations.
  warnings.warn(
Optimal Alpha: 100.0
```

In [ ]:
```
print("Lasso coefficients:", lasso_cv.coef_)
print("Intercept:", lasso_cv.intercept_)
```

```
Lasso coefficients: [[ 0.          0.         -0.0026802   0.          0.
0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.         -0.36418637  0.          0.          0.          0.
   0.          0.          0.          0.          0.        ]]
Intercept: [-3.09301121]
```

In [ ]:
```
# Assuming you have only one target class
coefficients = lasso_cv.coef_[0]

# Pair variable names with lasso coefficients
columns = df.drop('Bankrupt?', axis=1)
variable_names = list(columns)
features = dict(zip(variable_names, coefficients))
```

In [ ]:
```
# Filter to only show non-zero coefficients
non_zero_features = {var: coef for var, coef in features.items() if coef !=
non_zero_features
```

Out[ ]:
```
{' ROA(B) before interest and depreciation after tax': -0.00268019993231827
1,
 ' Debt ratio %': 0.17100195608613583,
 ' Net worth/Assets': -0.09478659625588029,
 ' Borrowing dependency': 0.03190801171913811,
 ' Net Income to Total Assets': -0.36418636773199015}
```

In [ ]:
```
# Extract keys from non_zero_features dictionary
selected_features = list(non_zero_features.keys())

# Subset X_train and X_test with selected features
X_train_selected = X_train[selected_features]
X_test_selected = X_test[selected_features]

# Train logistic regression model with selected features
logistic_model = LogisticRegression(class_weight='balanced')
logistic_model.fit(X_train_selected, y_train)
```

```
# Evaluate logistic regression model
train_accuracy = logistic_model.score(X_train_selected, y_train)
test_accuracy = logistic_model.score(X_test_selected, y_test)

print("Train Accuracy:", train_accuracy)
print("Test Accuracy:", test_accuracy)
```

```
Train Accuracy: 0.8438130155820348
Test Accuracy: 0.8321114369501467
```

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [ ]:
```
from sklearn.linear_model import Lasso
# set up model on training data with the optimal alpha
#lasso = Lasso(alpha = optimal_alpha)
#lasso_model = lasso.fit(X_train_scaled, y_train)

y_pred_lasso=logistic_model.predict(X_test_selected)
# calculate in-sample mse and r2
lasso_mse = mean_squared_error(y_test, y_pred_lasso)

print("Lasso MSE:", lasso_mse)
```

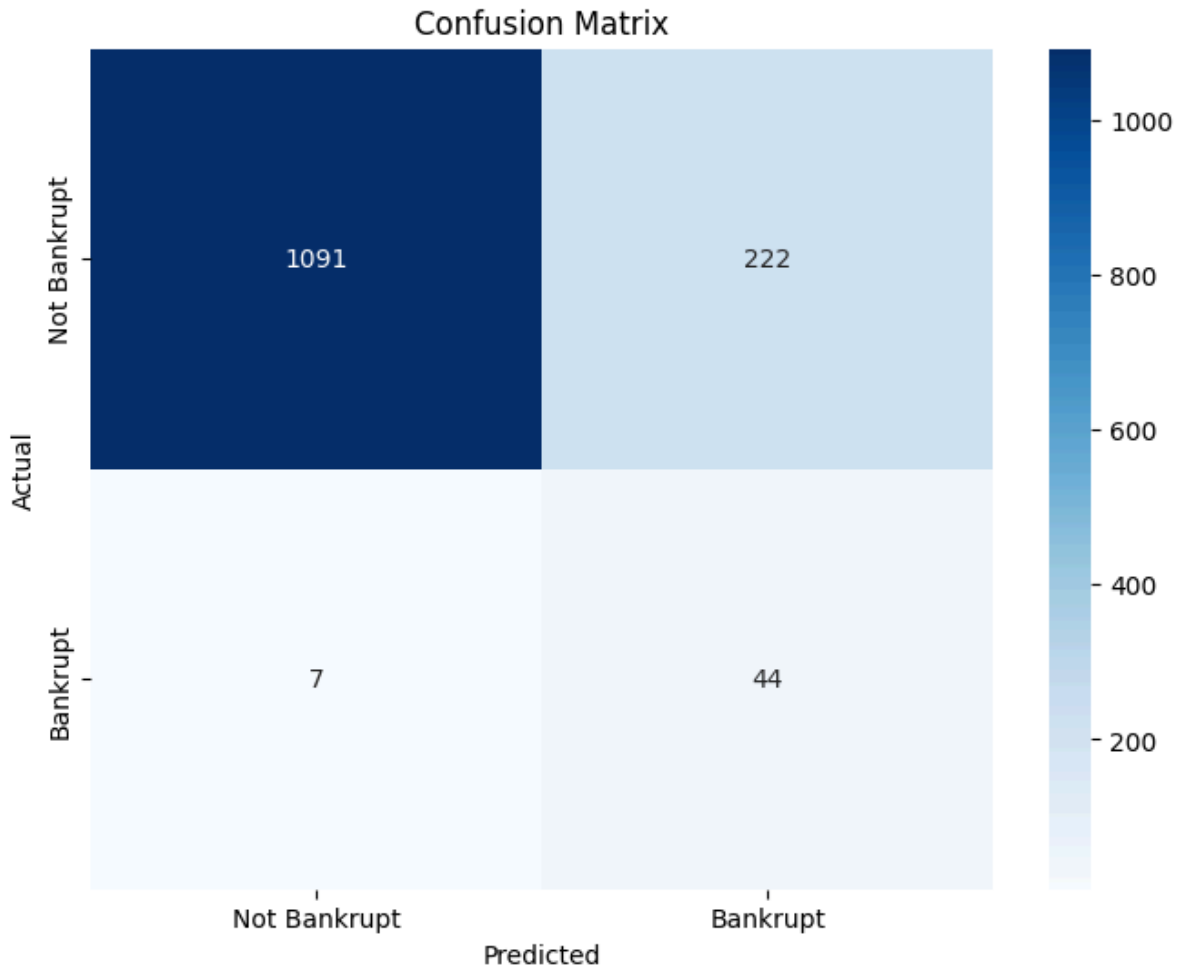
```
Lasso MSE: 0.16788856304985336
```

In [ ]:
```
# Make predictions on the test set
predictions_reduced = logistic_model.predict(X_test_selected)

# Evaluate model performance
conf_matrix_reduced = confusion_matrix(y_test, predictions_reduced)
class_report_reduced = classification_report(y_test, predictions_reduced)

# Print performance metrics
print("Reduced Model Cross-Validation Accuracy:", average_performance_reduce
print("Classification Report:\n", class_report_reduced)
print("Confusion Matrix:\n", conf_matrix_reduced)
```

```
Reduced Model Cross-Validation Accuracy: 0.9661240800788903
Classification Report:
               precision    recall  f1-score   support

           0       0.99      0.83      0.91      1313
           1       0.17      0.86      0.28        51

    accuracy                           0.83      1364
   macro avg       0.58      0.85      0.59      1364
weighted avg       0.96      0.83      0.88      1364

Confusion Matrix:
 [[1091  222]
 [   7   44]]
```

In [ ]:
```
# Calculate confusion matrix
conf_matrix_lasso = confusion_matrix(y_test, predictions_reduced)  # Assumir
```

```
# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_lasso, annot=True, fmt='d', cmap='Blues')
diag_labels = ['Not Bankrupt', 'Bankrupt']
plt.xticks(ticks=[0.5, 1.5], labels=diag_labels)
plt.yticks(ticks=[0.5, 1.5], labels=diag_labels)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



Confusion Matrix:

- True Positives (classified as bankrupt and are bankrupt): 44
- True Negatives (classified as not bankrupt and aren't bankrupt): 1091
- False Postives (Classified as bankrupt and aren't bankrupt): 222
- False Negatives (Classified as not bankrupt and are bankrupt): 7

Classification Decision Tree

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
des_tree_model = DecisionTreeClassifier(class_weight='balanced',random_state


# Fit the model
des_tree_model.fit(X_train_scaled, y_train)

# Make predictions
y_pred_dt = des_tree_model.predict(X_test_scaled)

# Evaluate the model

accuracy_dt = accuracy_score(y_test, y_pred_dt)
print('Accuracy:', accuracy_dt)
mse = mean_squared_error(y_test, y_pred_dt)
print('Mean Squared Error (MSE):', mse)
```

```
Accuracy: 0.9530791788856305
Mean Squared Error (MSE): 0.0469208211143695
```

In [ ]:
```
# running decision tree using cross validation (in sample)
from sklearn.model_selection import cross_val_score,cross_val_predict

# 5x k-fold cross-validation
cv_scores = cross_val_score(des_tree_model, X_train_scaled, y_train, cv=5)

# Cross-validation scores
print("Cross-Validation Scores:", cv_scores)

# Average cross-validation score
mean_cv_score = cv_scores.mean()
print("Average Cross Validation Score:", mean_cv_score)
```

```
Cross-Validation Scores: [0.95233731 0.94958753 0.95692026 0.95600367 0.952
33731]
Average Cross Validation Score: 0.9534372135655362
```

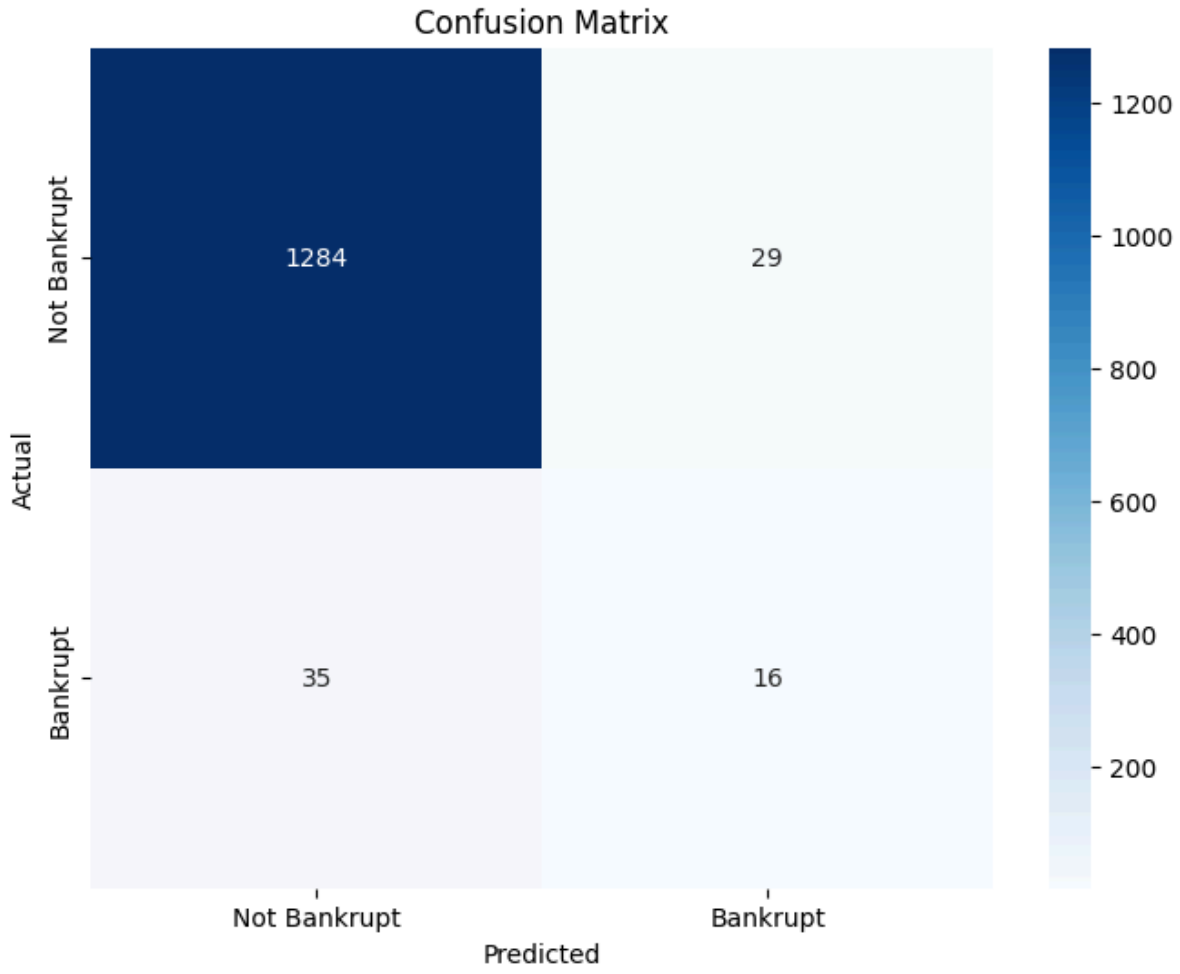
In [ ]:
```
# Out of sample prediction
from sklearn.metrics import accuracy_score

# Perform cross-validated predictions (let's say k=5)
y_pred_cv = cross_val_predict(des_tree_model, X_train_scaled, y_train, cv=5)

# Fit the model on the entire training data
des_tree_model.fit(X_train_scaled, y_train)

# Make predictions on the test data
y_pred_test = des_tree_model.predict(X_test_scaled)

# Evaluate the model on the out-of-sample test data
accuracy_test = accuracy_score(y_test, y_pred_test)
print('Accuracy on Test Data:', accuracy_test)

# Evaluate the cross-validated predictions
accuracy_cv = accuracy_score(y_train, y_pred_cv)
print('Accuracy from Cross-Validation:', accuracy_cv)
```

```
Accuracy on Test Data: 0.9530791788856305
Accuracy from Cross-Validation: 0.9534372135655362
```

In [ ]:
```
conf_matrix = confusion_matrix(y_test, y_pred_dt)
class_report = classification_report(y_test, y_pred_dt)


print("Classification Report:\n", class_report)
print("Confusion Matrix:\n",conf_matrix)
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.97      0.98      0.98      1313
           1       0.36      0.31      0.33        51

    accuracy                           0.95      1364
   macro avg       0.66      0.65      0.65      1364
weighted avg       0.95      0.95      0.95      1364

Confusion Matrix:
 [[1284   29]
 [  35   16]]
```

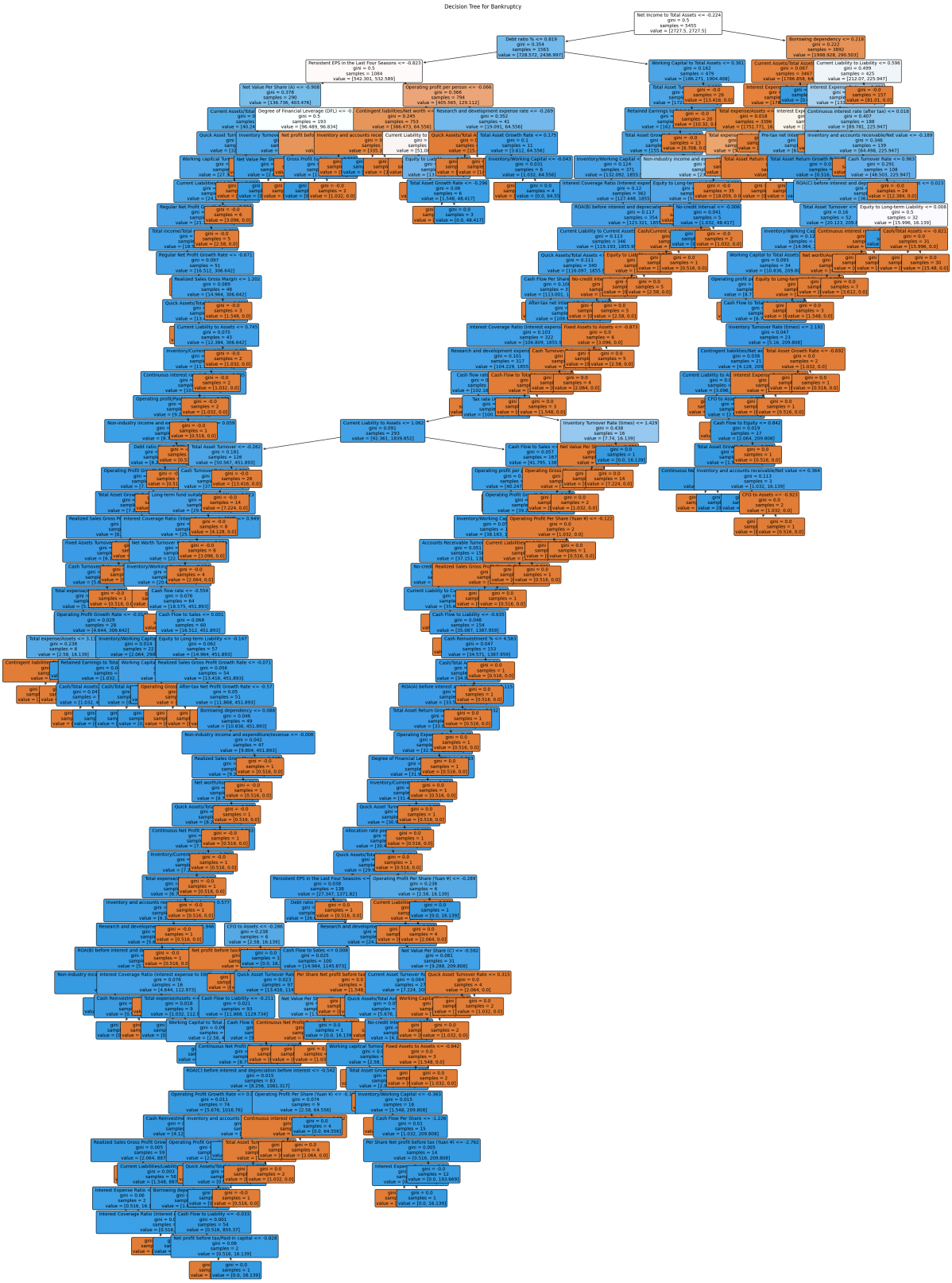
In [ ]:
```
# Calculate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred_dt)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
diag_labels = ['Not Bankrupt', 'Bankrupt']
plt.xticks(ticks=[0.5, 1.5], labels=diag_labels)
plt.yticks(ticks=[0.5, 1.5], labels=diag_labels)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



Confusion Matrix:

- True Positives (classified as bankrupt and are bankrupt): 16
- True Negatives (classified as not bankrupt and aren't bankrupt): 1284
- False Postives (Classified as bankrupt and aren't bankrupt): 29
- False Negatives (Classified as not bankrupt and are bankrupt): 35

In [ ]:
```
#plot tree
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt


plt.figure(figsize=(30,40))
plot_tree(des_tree_model, feature_names=X.columns, filled=True, rounded=True
plt.title("Decision Tree for Bankruptcy")
plt.tight_layout()
plt.show()
```



Random Forest Classification Model

In [ ]:
```
# set seed for reproducability
np.random.seed(seed=10)
```

In [ ]:
```
#using same splits from logistic regression model

#scale data
scalar=StandardScaler()
X_train=scalar.fit_transform(X_train)
X_test=scalar.transform(X_test)

#assign weights to account for class imbalance
rf_model = RandomForestClassifier(class_weight='balanced',n_estimators=100)
rf_model.fit(X_train, y_train)
```

```
/var/folders/tn/vsw4vgdx4fz3r22myj1yg_j80000gn/T/ipykernel_1031/2546931895.
py:10: DataConversionWarning: A column-vector y was passed when a 1d array
was expected. Please change the shape of y to (n_samples,), for example usi
ng ravel().
  rf_model.fit(X_train, y_train)
```

Out[ ]:
```
▾ RandomForestClassifier
RandomForestClassifier(class_weight='balanced')
```

In [ ]:
```
#oos validation

#predictions on the test set
rf_pred = rf_model.predict(X_test)

# Compute Mean Squared Error
mse = mean_squared_error(y_test, rf_pred)
print("OOS Mean Squared Error:", mse)

# Calculate the accuracy score
accuracy = accuracy_score(y_test, rf_pred)
print("OOS Accuracy Score:", accuracy)
```

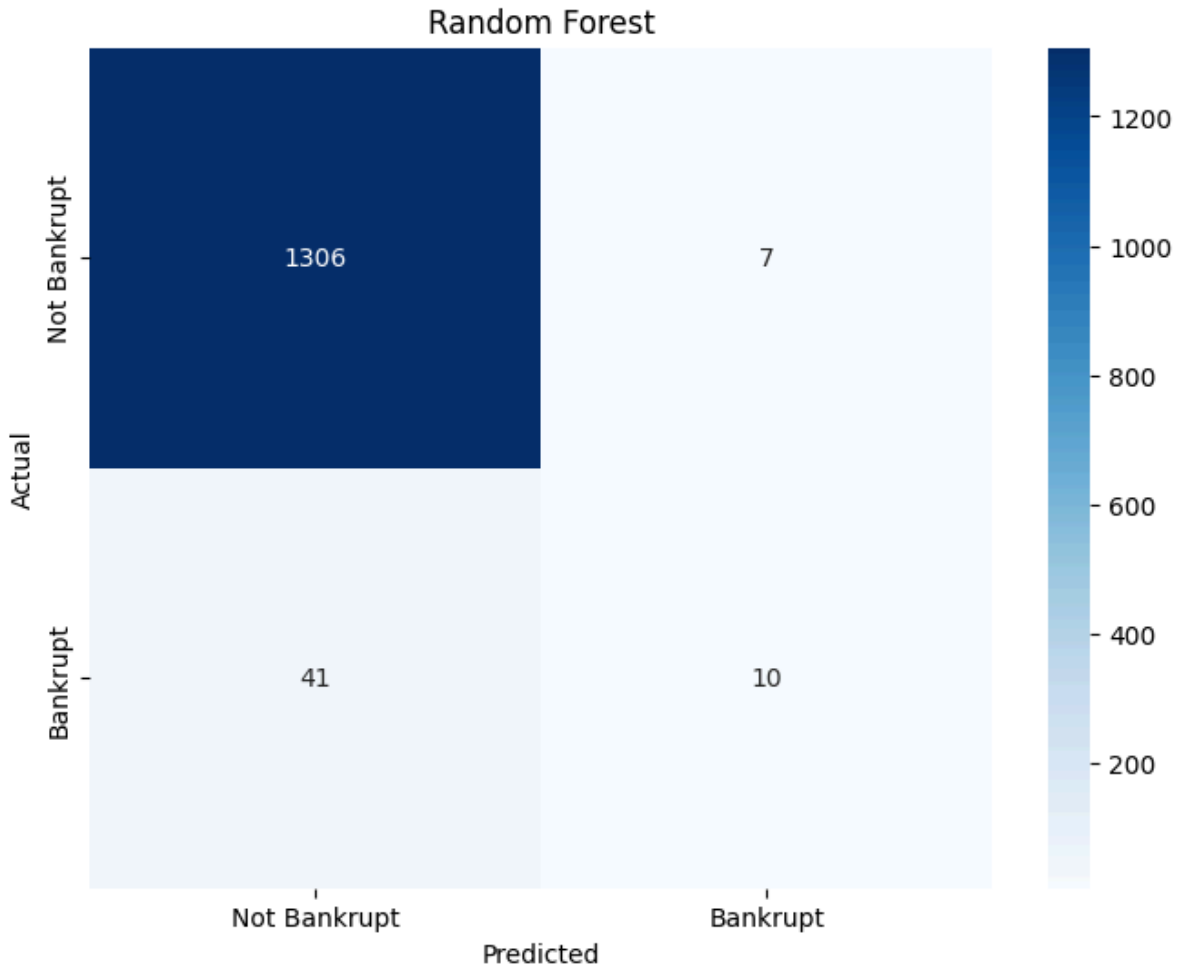
```
OOS Mean Squared Error: 0.03519061583577713
OOS Accuracy Score: 0.9648093841642229
```

In [ ]:
```
conf_matrix = confusion_matrix(y_test, rf_pred)
class_report = classification_report(y_test, rf_pred)


print("Classification Report:\n", class_report)
print("Confusion Matrix:\n",conf_matrix)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      1313
           1       0.59      0.20      0.29        51

    accuracy                           0.96      1364
   macro avg       0.78      0.60      0.64      1364
weighted avg       0.96      0.96      0.96      1364

Confusion Matrix:
 [[1306    7]
 [  41   10]]
```

In [ ]:

```python
# Calculate confusion matrix
conf_matrix = confusion_matrix(y_test, rf_pred)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
diag_labels = ['Not Bankrupt', 'Bankrupt']
plt.xticks(ticks=[0.5, 1.5], labels=diag_labels)
plt.yticks(ticks=[0.5, 1.5], labels=diag_labels)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Random Forest')
plt.show()
```



Confusion Matrix:

- True Positives (classified as bankrupt and are bankrupt): 10
- True Negatives (classified as not bankrupt and aren't bankrupt): 1306
- False Postives (Classified as bankrupt and aren't bankrupt): 7
- False Negatives (Classified as not bankrupt and are bankrupt): 41

Next we will try to adjust the threshold for classification in order to better account for the class imbalance and try to balance the trade of between precision and recall in classifying banks as bankrupt.

In [ ]:
```
# Define the step size for threshold values
step_size = 0.05

# Define a range of threshold values to iterate over
thresholds = np.arange(0, 1.01, step_size)

best_threshold = None
best_f1_score = 0

# Iterate over each threshold value
for threshold in thresholds:
```

```
    # Convert probabilities to binary predictions based on the threshold
    binary_predictions = (rf_model.predict_proba(X_test)[:, 1] >= threshold)

    # Compute the classification report for the binary predictions
    new_class_report = classification_report(y_test, binary_predictions, out

    # Extract F1-score for the positive class
    new_f1_score = new_class_report['1']['f1-score']

    # Update the best threshold and F1-score if needed
    if new_f1_score > best_f1_score:
        best_f1_score = new_f1_score
        best_threshold = threshold

print("Best Threshold:", best_threshold)
print("Best F1-Score:", best_f1_score)
```

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Preci
sion and F-score are ill-defined and being set to 0.0 in labels with no pre
dicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Preci
sion and F-score are ill-defined and being set to 0.0 in labels with no pre
dicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Preci
sion and F-score are ill-defined and being set to 0.0 in labels with no pre
dicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
Best Threshold: 0.15000000000000002
Best F1-Score: 0.48437499999999994
```

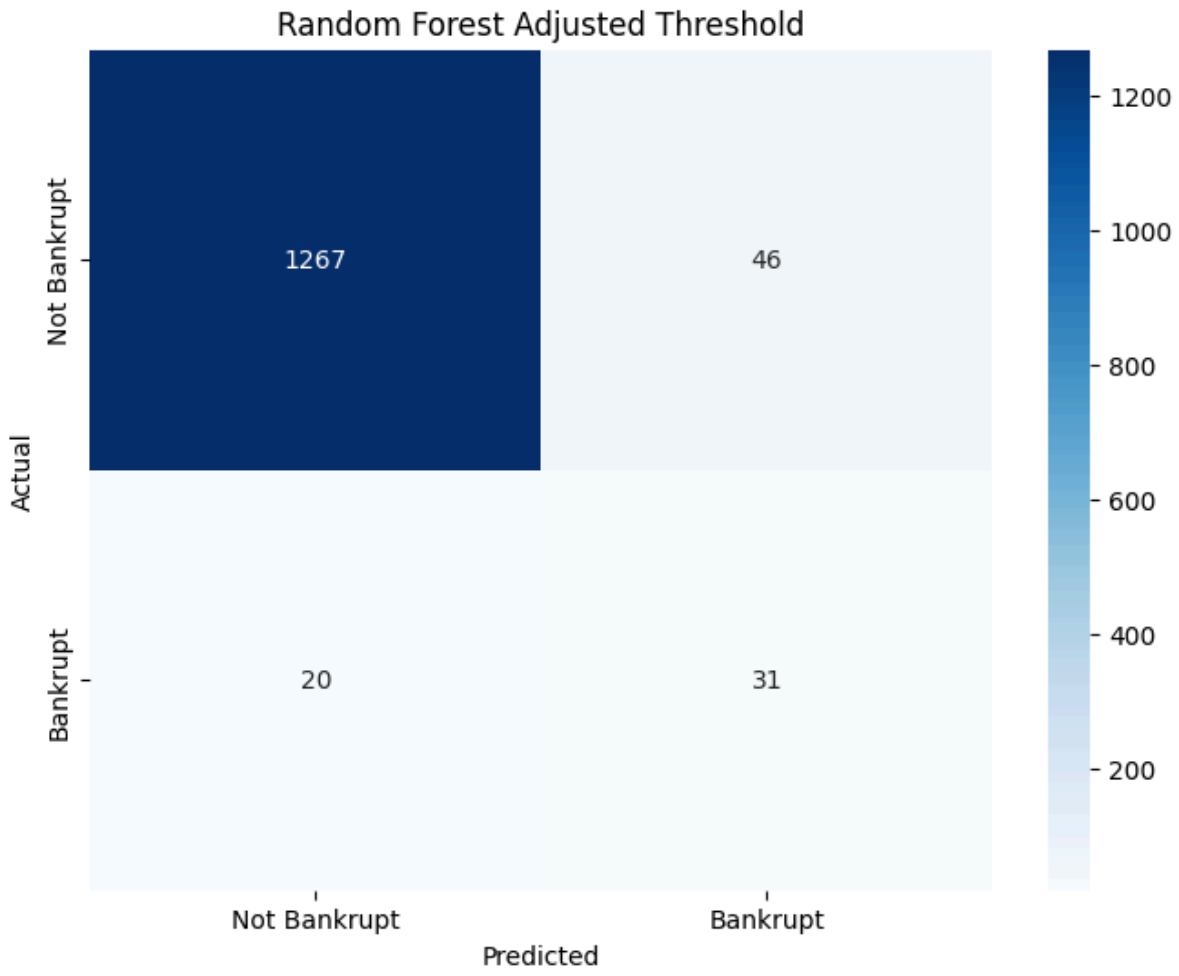
```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Preci
sion and F-score are ill-defined and being set to 0.0 in labels with no pre
dicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Preci
sion and F-score are ill-defined and being set to 0.0 in labels with no pre
dicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Preci
sion and F-score are ill-defined and being set to 0.0 in labels with no pre
dicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Preci
sion and F-score are ill-defined and being set to 0.0 in labels with no pre
dicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Preci
sion and F-score are ill-defined and being set to 0.0 in labels with no pre
dicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Preci
sion and F-score are ill-defined and being set to 0.0 in labels with no pre
dicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Preci
sion and F-score are ill-defined and being set to 0.0 in labels with no pre
dicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Preci
sion and F-score are ill-defined and being set to 0.0 in labels with no pre
dicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Preci
sion and F-score are ill-defined and being set to 0.0 in labels with no pre
dicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Preci
sion and F-score are ill-defined and being set to 0.0 in labels with no pre
dicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Preci
sion and F-score are ill-defined and being set to 0.0 in labels with no pre
dicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
```

```
ages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Preci
sion and F-score are ill-defined and being set to 0.0 in labels with no pre
dicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [ ]:
```python
# Compute predictions using the best threshold
best_threshold_predictions = (rf_model.predict_proba(X_test)[:, 1] >= best_t

# Print the classification report for the best threshold
print("Classification Report for the Best Threshold:")
print(classification_report(y_test, best_threshold_predictions))

# Compute the confusion matrix for the best threshold
best_threshold_confusion_matrix = confusion_matrix(y_test, best_threshold_pr

# Print the confusion matrix for the best threshold
print("Confusion Matrix for the Best Threshold:")
print(best_threshold_confusion_matrix)
```

```
Classification Report for the Best Threshold:
              precision    recall  f1-score   support

           0       0.98      0.96      0.97      1313
           1       0.40      0.61      0.48        51

    accuracy                           0.95      1364
   macro avg       0.69      0.79      0.73      1364
weighted avg       0.96      0.95      0.96      1364

Confusion Matrix for the Best Threshold:
[[1267   46]
 [  20   31]]
```

In [ ]:
```python
# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(best_threshold_confusion_matrix, annot=True, fmt='d', cmap='Blue
diag_labels = ['Not Bankrupt', 'Bankrupt']
plt.xticks(ticks=[0.5, 1.5], labels=diag_labels)
plt.yticks(ticks=[0.5, 1.5], labels=diag_labels)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Random Forest Adjusted Threshold')
plt.show()
```



Confusion Matrix:

- True Positives (classified as bankrupt and are bankrupt): 31
- True Negatives (classified as not bankrupt and aren't bankrupt): 1267
- False Postives (Classified as bankrupt and aren't bankrupt): 46
- False Negatives (Classified as not bankrupt and are bankrupt): 20

Next we will calculate mean decrease in impurity and permutation feature importance to identify the most important variables in classifying companys as bankrupt.

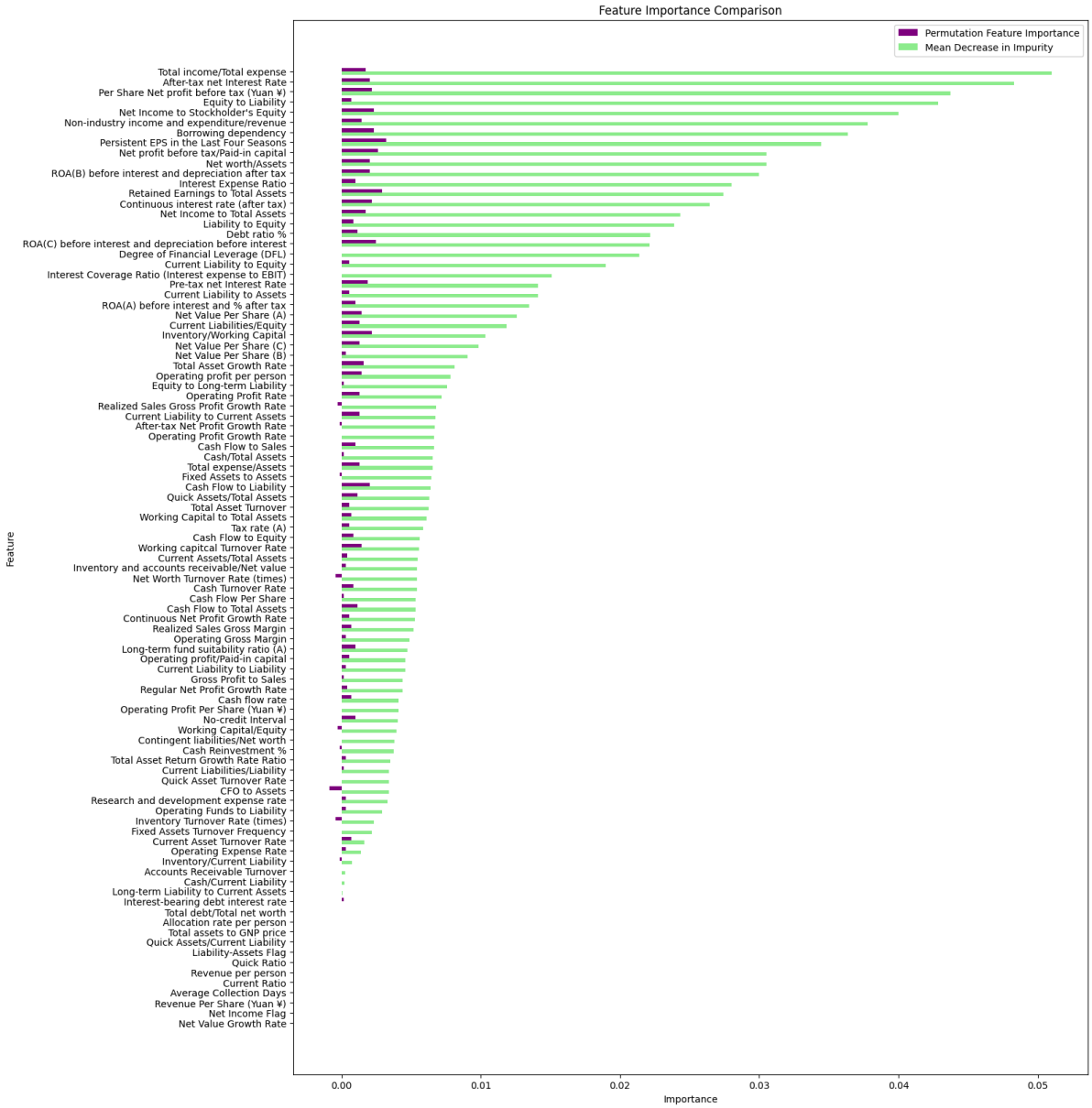
In [ ]:
```
# Variable importance

#mean decrease in impurity
mdi=rf_model.feature_importances_

#Permutation Feature Importance
pfi=permutation_importance(rf_model, X_test, y_test,random_state=42)
pfi_mean=pfi.importances_mean

#dataframe to compare mdi and permutation for each feature
feature_importance_df = pd.DataFrame({
    'Feature': X.columns,
    'MDI': mdi,
```

```
    'Permutation': pfi_mean
})
print(feature_importance_df)
```

```
                                                Feature       MDI  Permutatio
n
0    ROA(C) before interest and depreciation befor...  0.022141     0.00249
3
1              ROA(A) before interest and % after tax  0.013469     0.00102
6
2    ROA(B) before interest and depreciation after...  0.029998     0.00205
3
3                             Operating Gross Margin  0.004889     0.00029
3
4                        Realized Sales Gross Margin  0.005178     0.00073
3
..                                               ...       ...
...
90                            Liability to Equity  0.023889     0.00088
0
91                Degree of Financial Leverage (DFL)  0.021367     0.00000
0
92   Interest Coverage Ratio (Interest expense to ...  0.015087     0.00000
0
93                                  Net Income Flag  0.000000     0.00000
0
94                            Equity to Liability  0.042867     0.00073
3

[95 rows x 3 columns]
```

In [ ]:

```
#sort feature importance
feature_importance_df_sorted = feature_importance_df.sort_values(by='MDI', a

# Create bar chart
plt.figure(figsize=(15, 20))

# Set the width of the bars
bar_width = 0.35

# Set the positions for the bars
index = np.arange(len(feature_importance_df_sorted))

# Plot Permutation Feature Importance
plt.barh(index, feature_importance_df_sorted['Permutation'], bar_width, colo

# Plot Mean Decrease in Impurity (MDI) next to the Permutation Feature Impor
plt.barh(index + bar_width, feature_importance_df_sorted['MDI'], bar_width,

plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Feature Importance Comparison')
plt.yticks(index + bar_width / 2, feature_importance_df_sorted['Feature'])
plt.legend()
plt.gca().invert_yaxis()  # Invert y-axis to show the most important feature
plt.show()
```



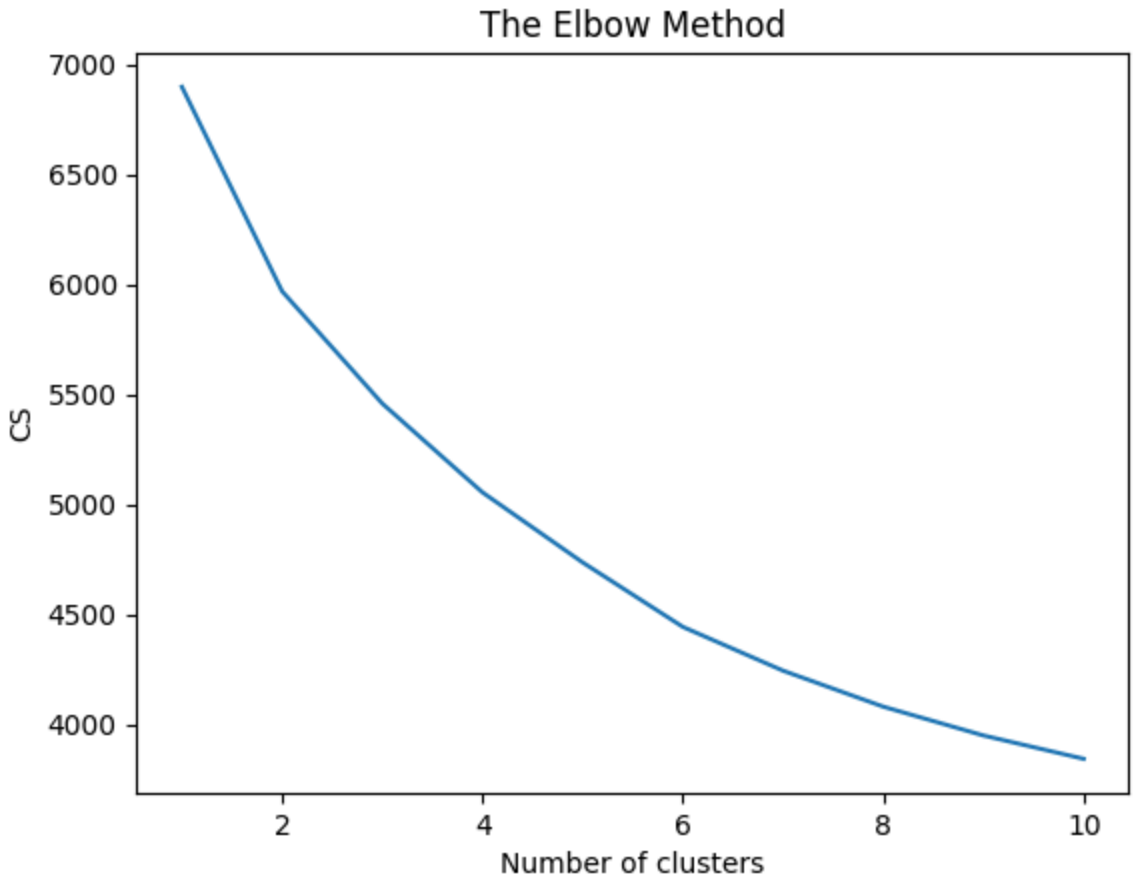
K-Means Clustering

Here we use k-means as a method of exploration in classifying groups based off of similarities.

In [ ]:
```
#prepare data for clusters
df_cluster=df.drop('Bankrupt?', axis=1)
```

In [ ]:
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import KMeans

# Scale the data
ms = MinMaxScaler()
scaled_data = ms.fit_transform(df_cluster)
scaled_df = pd.DataFrame(scaled_data, columns=df_cluster.columns)
```

In [ ]:
```
#choosing optimal K using the elbow method
cs = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_ir
    kmeans.fit(scaled_df)
    cs.append(kmeans.inertia_)
plt.plot(range(1, 11), cs)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('CS')
plt.show()
```



Using the elbow method we can group observations into 2 cluster.

In [ ]:
```
#creating clusters using k=2
kmeans = KMeans(n_clusters=2, random_state=0)
kmeans.fit(scaled_df)
#attach cluster labels to df
scaled_df['kmeans_cluster'] = kmeans.labels_
scaled_df.head()
```

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n
_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` expl
icitly to suppress the warning
  warnings.warn(
```

Out[ ]:

	ROA(C) before interest and depreciation before interest	ROA(A) before interest and % after tax	ROA(B) before interest and depreciation after tax	Operating Gross Margin	Realized Sales Gross Margin	Operating Profit Rate	Pre-tax net Interest Rate	After-ta ne Interes Rat
0	0.370594	0.424389	0.405750	0.601457	0.601457	0.998969	0.796887	0.80880
1	0.464291	0.538214	0.516730	0.610235	0.610235	0.998946	0.797380	0.80930
2	0.426071	0.499019	0.472295	0.601450	0.601364	0.998857	0.796403	0.80838
3	0.399844	0.451265	0.457733	0.583541	0.583541	0.998700	0.796967	0.80896
4	0.465022	0.538432	0.522298	0.598783	0.598783	0.998973	0.797366	0.80930

5 rows × 96 columns

In [ ]:
```
#count number of observations in each cluster
scaled_df['kmeans_cluster'].value_counts()
```

Out[ ]:
```
1    4694
0    2125
Name: kmeans_cluster, dtype: int64
```

In [ ]:
```
#adding back in bankrupt column with cluster
scaled_df['Bankrupt?']=df['Bankrupt?']
scaled_df.head()
```

Out[ ]:

	ROA(C) before interest and depreciation before interest	ROA(A) before interest and % after tax	ROA(B) before interest and depreciation after tax	Operating Gross Margin	Realized Sales Gross Margin	Operating Profit Rate	Pre-tax net Interest Rate	After-ta ne Interes Rat
0	0.370594	0.424389	0.405750	0.601457	0.601457	0.998969	0.796887	0.80880
1	0.464291	0.538214	0.516730	0.610235	0.610235	0.998946	0.797380	0.80930
2	0.426071	0.499019	0.472295	0.601450	0.601364	0.998857	0.796403	0.80838
3	0.399844	0.451265	0.457733	0.583541	0.583541	0.998700	0.796967	0.80896
4	0.465022	0.538432	0.522298	0.598783	0.598783	0.998973	0.797366	0.80930

5 rows × 97 columns

In [ ]:
```
br_c1= scaled_df[(scaled_df['Bankrupt?'] == 1) & (scaled_df['kmeans_cluster'
count_br_c1 = br_c1.shape[0]
count_br_c1
print("Number of bankrupt instances in cluster 1:", count_br_c1)
```

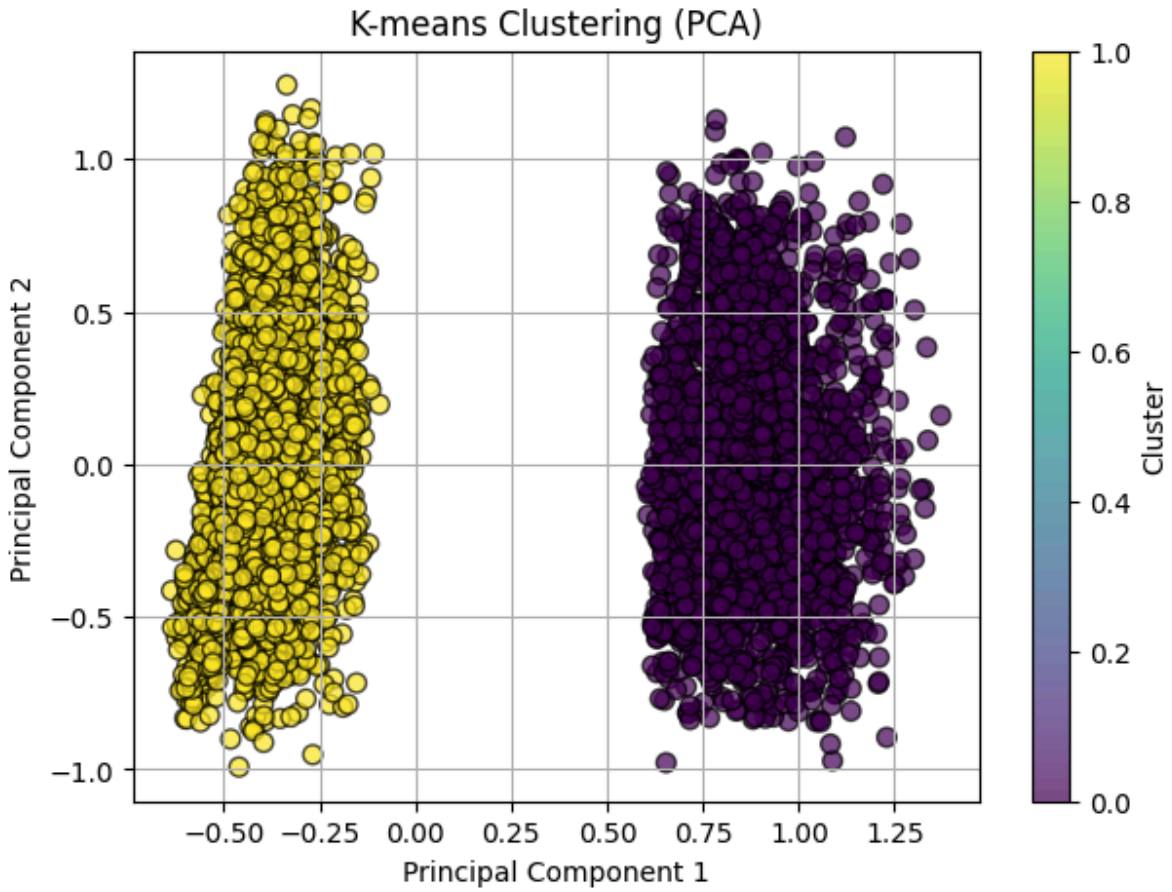
```
Number of bankrupt instances in cluster 1: 130
```

In [ ]:
```
br_c0= scaled_df[(scaled_df['Bankrupt?'] == 1) & (scaled_df['kmeans_cluster'
count_br_c0 = br_c0.shape[0]
count_br_c0
print("Number of bankrupt instances in cluster 0:", count_br_c0)
```

```
Number of bankrupt instances in cluster 0: 90
```

In [ ]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(scaled_df)
```

In [ ]:
```
# Plot data points with cluster assignments in the reduced 2D space
plt.figure(figsize=(7, 5))

# Plot data points
labels = kmeans.labels_
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=labels, cmap='viridis', edgecolors='

# Add labels and title
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('K-means Clustering (PCA)')

# Add legend
plt.colorbar(label='Cluster')
plt.grid(True)
plt.show()
```



Understanding variables of each cluster

In [ ]:
```
cluster_0=scaled_df[scaled_df['kmeans_cluster']==0]
summary0=cluster_0.describe()
summary0
```

Out[ ]:

	ROA(C) before interest and depreciation before interest	ROA(A) before interest and % after tax	ROA(B) before interest and depreciation after tax	Operating Gross Margin	Realized Sales Gross Margin	Operating Profit Rate	Pr
count	2125.000000	2125.000000	2125.000000	2125.000000	2125.000000	2125.000000	212
mean	0.502626	0.555828	0.550595	0.604070	0.604055	0.999010	
std	0.053865	0.057419	0.054121	0.008970	0.008957	0.000111	
min	0.066933	0.057185	0.054821	0.555939	0.555939	0.998111	
25%	0.478038	0.537178	0.528722	0.598560	0.598553	0.998972	
50%	0.500951	0.559093	0.550511	0.602293	0.602286	0.999006	
75%	0.526690	0.581880	0.575566	0.607533	0.607489	0.999054	
max	0.864964	0.942706	0.932598	0.657548	0.657548	0.999706	

8 rows × 97 columns

In [ ]:
```
# Extract only the mean row
mean_c0 = summary0.loc['mean']
```

In [ ]:
```
cluster_1=scaled_df[scaled_df['kmeans_cluster']==1]
summary1=cluster_1.describe()
summary1
```

Out[ ]:

	ROA(C) before interest and depreciation before interest	ROA(A) before interest and % after tax	ROA(B) before interest and depreciation after tax	Operating Gross Margin	Realized Sales Gross Margin	Operating Profit Rate
count	4694.000000	4694.000000	4694.000000	4694.000000	4694.000000	4694.000000
mean	0.506336	0.559891	0.554944	0.609703	0.609683	0.998640
std	0.063505	0.068982	0.064655	0.019243	0.019223	0.015680
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.475467	0.534807	0.526527	0.601976	0.601976	0.998967
50%	0.504022	0.560565	0.553456	0.608286	0.608246	0.999034
75%	0.539609	0.593218	0.588830	0.616071	0.616026	0.999118
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

8 rows × 97 columns

In [ ]:
```
# Extract only the mean row
mean_c1 = summary1.loc['mean']
```

In [ ]:
```
#finding the difference in means between each cluster
mean_df=pd.DataFrame({"Cluster 0 Means": mean_c0, "Cluster 1 Means": mean_c1
mean_df
```

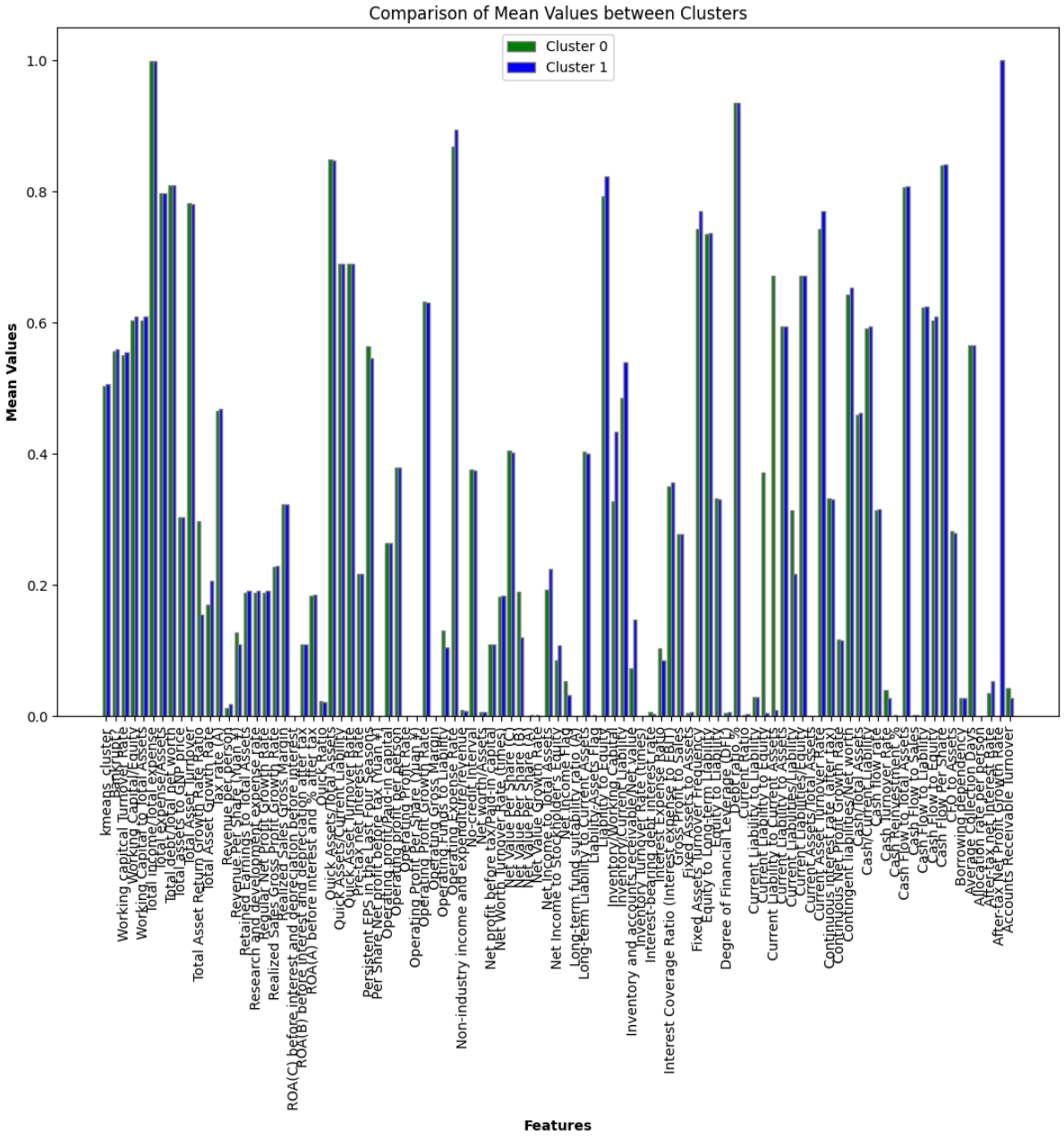
Out[ ]:

	Cluster 0 Means	Cluster 1 Means	Difference in Means:
ROA(C) before interest and depreciation before interest	0.502626	0.506336	-0.003710
ROA(A) before interest and % after tax	0.555828	0.559891	-0.004063
ROA(B) before interest and depreciation after tax	0.550595	0.554944	-0.004348
Operating Gross Margin	0.604070	0.609703	-0.005633
Realized Sales Gross Margin	0.604055	0.609683	-0.005629
...	...	...	...
Interest Coverage Ratio (Interest expense to EBIT)	0.565229	0.565416	-0.000188
Net Income Flag	0.000000	0.000000	0.000000
Equity to Liability	0.035219	0.053173	-0.017954
kmeans_cluster	0.000000	1.000000	-1.000000
Bankrupt?	0.042353	0.027695	0.014658

97 rows × 3 columns

In [ ]:

```
# Get feature names
features = sorted(mean_c1.index,reverse=True)

# Set bar width
bar_width = 0.35

# Set position of bar on X axis
r1 = np.arange(len(features))
r2 = [x + bar_width for x in r1]

# Plotting the bar chart, *** correted check with homie
plt.figure(figsize=(13, 9))
plt.bar(r1, mean_c0, color='g', width=bar_width, edgecolor='grey', label='Cl
plt.bar(r2, mean_c1, color='b', width=bar_width, edgecolor='grey', label='Cl

# Adding labels
plt.xlabel('Features', fontweight='bold')
plt.ylabel('Mean Values', fontweight='bold')
plt.xticks([r + bar_width/2 for r in range(len(features))],features, rotatic

# Adding legend
plt.legend()

# Title
plt.title('Comparison of Mean Values between Clusters')

# Show plot
plt.show()
```


Comparison of Mean Values between Clusters
Cluster 0
Cluster 1
Mean Values
Features
kmeans_cluster
Bankrupt?
Working Capital Turnover Rate
Working Capital to Total Assets
Total Income/Total Expense
Total Expense/Assets
Total assets to GNP price
Total Asset Turnover
Total Asset Return Growth Rate Ratio
Tax rate (A)
Revenue per person
Retained Earnings to Total Assets
Research and development expense rate
Regular Net Profit Growth Rate
Realized Sales Gross Profit Growth Rate
ROA(C) before interest and depreciation before interest
ROA(B) before interest and depreciation after tax
ROA(A) before interest and % after tax
Quick Ratio
Quick Assets/Total Assets
Quick Assets/Current Liability
Quick Asset Turnover Rate
Persistent EPS in the Last Four Seasons
Per Share Net profit before tax (Yuan ¥)
Operating profit/Paid-in capital
Operating Profit Rate
Operating Profit Per Share (Yuan ¥)
Operating Profit Growth Rate
Operating Funds to Liability
Operating Expense Rate
Non-industry income and expenditure/revenue
Net worth/Assets
Net profit before tax/Paid-in capital
Net Worth Turnover Rate (times)
Net Value Per Share (C)
Net Value Per Share (B)
Net Value Per Share (A)
Net Value Growth Rate
Net Worth Turnover Rate (times)
Net to Stock Holders Equity
Net Income to Total Assets
Net Income Flag
Long-term fund suitability ratio (A)
Long-term Liability to Current Assets
Liability-Assets Flag
Liability to Equity
Inventory/Working Capital
Inventory/Current Liability
Inventory and accounts receivable/Net value
Inventory Turnover Rate (times)
Interest-bearing debt interest rate
Interest Expense Ratio
Interest Coverage Ratio (Interest expense to EBIT)
Gross Profit to Sales
Fixed Assets to Assets
Fixed Assets Turnover Frequency
Equity to Long-term Liability
Equity to Liability
Degree of Financial Leverage (DFL)
Current Ratio
Current Liability to Liability
Current Liability to Equity
Current Liability to Assets
Current Liabilities/Liability
Current Liabilities/Equity
Current Asset Turnover Rate
Continuous Net Profit Growth Rate
Contingent liabilities/Net worth
Cash/Total Assets
Cash/Current Liability
Cash Turnover Rate
Cash Reinvestment %
Cash Flow to Sales
Cash Flow to Total Assets
Cash Flow to Liability
Cash Flow to Equity
Cash Flow per Share
Cash Flow Rate
Cash/Total Assets
Borrowing Dependency
Average Collection Days
Allocation rate per person
After-tax Net Interest Rate
Accounts Receivable Turnover