Introduction

This is a binary classification problem to predict company bankruptcy using data from the Taiwan Economic Journal for the years 1999–2009. The dataset is highly imbalanced, a common characteristic for default/bankruptcy prediction problems. All predictors in this dataset are numeric, primarily consisting of financial indicators from financial statements. There are no missing values in this dataset.

The dataset can be accessed here: https://www.kaggle.com/datasets/fedesoriano/company-bankruptcy-prediction/data

The full dataset comprises 6,819 observations with 96 predictors, which poses a "Curse of Dimensionality" problem. Therefore, my goal is to develop a powerful model that distinguishes between events (bankruptcies) and non-events while ensuring a robust set of variables through a comprehensive variable selection process. I aim to achieve a model with an AUC-ROC greater than 90% on the test set using fewer than 10 final predictors. This approach adheres to the principle of "Parsimony" and avoids the "Curse of Dimensionality."

My Plan for Variable Selection:

1. Screening: Remove near-zero variance (NZV) and zero variance (ZV) predictors, and eliminate highly correlated variables to reduce redundancy.
2. Filtering Method: Use automatic Weight of Evidence (WoE) binning via Decision Trees and drop variables based on Information Value (IV).
3. Wrapper Method: Apply Recursive Feature Elimination (RFE) to perform an initial broad reduction of the feature set. RFE quickly eliminates a large number of irrelevant features based on their importance scores from an estimator.
4. Performance-Oriented Method: Use Backward Trimming (BART) to ensure that the final feature set is optimized for performance, potentially addressing any multicollinearity issues or feature interactions that RFE might not fully capture.

After the variable selection process, I will use a Random Forest classifier with hyperparameter tuning to train the model.

1. Import Data and Simple Manipulation

In [1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score, StratifiedKFold, GridS
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.preprocessing import RobustScaler
from sklearn.feature_selection import VarianceThreshold, RFECV, RFE
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import roc_auc_score, roc_curve, auc, confusion_matrix, f1_score
from sklearn.pipeline import Pipeline
import matplotlib.pyplot as plt
import seaborn as sns
import re
from optbinning import BinningProcess
from feature_engine.selection import SmartCorrelatedSelection, RecursiveFeatureElimination
from sklearn.tree import DecisionTreeClassifier
import numpy as np
```

```
(CVXPY) Jul 31 03:03:23 PM: Encountered unexpected exception importing solver GLOP:
RuntimeError('Unrecognized new version of ortools (9.10.4067). Expected < 9.10.0. Please open
a feature request on cvxpy to enable support for this version.')
(CVXPY) Jul 31 03:03:23 PM: Encountered unexpected exception importing solver PDLP:
RuntimeError('Unrecognized new version of ortools (9.10.4067). Expected < 9.10.0. Please open
a feature request on cvxpy to enable support for this version.')
```

In [2]:
```
data = pd.read_csv('Bankruptcy_Prediction.csv')
```

In [3]:
```
data.head(5)
```

Out[3]:

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

5 rows × 96 columns

All of the predictors are numeric.

In [4]:
```
numeric_count = len(data.select_dtypes(include=['number']).columns)
numeric_count
```

Out[4]:
```
96
```

In [5]:
```
character_count = len(data.select_dtypes(include=['object', 'string']).columns)
character_count
```

Out[5]:
```
0
```

In [6]:
```
data.columns[0:10]
```

Out[6]:
```
Index(['Bankrupt?', ' ROA(C) before interest and depreciation before interest',
       ' ROA(A) before interest and % after tax',
       ' ROA(B) before interest and depreciation after tax',
       ' Operating Gross Margin', ' Realized Sales Gross Margin',
       ' Operating Profit Rate', ' Pre-tax net Interest Rate',
       ' After-tax net Interest Rate',
       ' Non-industry income and expenditure/revenue'],
      dtype='object')
```

In [7]:
```
def clean_column_name(col_name):
    # Remove spaces and special characters, replace them with underscores
    col_name = re.sub(r'[^A-Za-z0-9]+', '_', col_name)
    # Remove leading and trailing underscores
    col_name = col_name.strip('_')
    return col_name
```

Clean the names of all columns to be in standard format

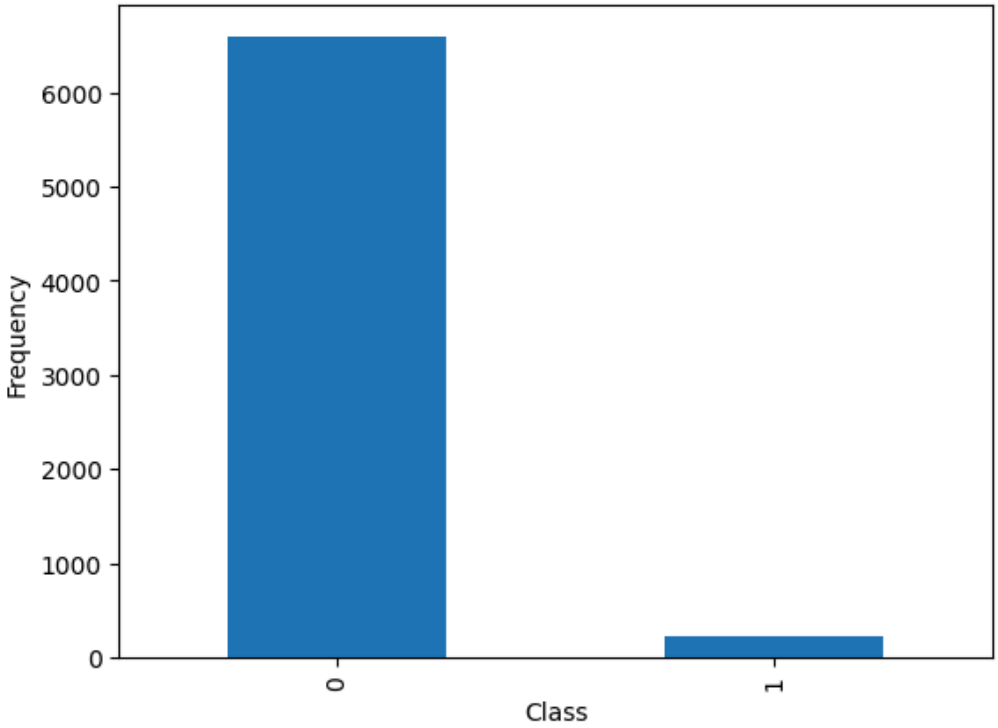
In [8]:
```
data.columns = [clean_column_name(col) for col in data.columns]
data.columns[0:5]
```

Out[8]:
```
Index(['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'],
      dtype='object')
```

The dataset is highly imbalanced.

In [9]:
```
data['Bankrupt'].value_counts().plot.bar()
plt.xlabel('Class')
plt.ylabel('Frequency')
data['Bankrupt'].value_counts()
```

Out[9]:
```
Bankrupt
0    6599
1     220
Name: count, dtype: int64
```



2. Data Preparation, Feature Engineering, Variable Selection

The target variable is 'Bankrupt' and should be converted to categorical

In [10]:
```
data['Bankrupt'] = data['Bankrupt'].astype('category')
```

There is no missing values in the dataset

In [11]:
```
missing_values_count = data.isnull().sum().sum()
print(missing_values_count)
```

```
0
```

Split the data into training and test sets before any preprocessing

In [12]:
```
X = data.drop(columns=['Bankrupt'])
y = data['Bankrupt']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=123, s
```

2.1 Preprocessing (ZV, Robust Scaler)

Define preprocessing steps.

1. remove Zero Variance variables
2. use Robust Scaler to handle outliers
3. No need to impute missingness because there are no missing values

In [13]:
```
preprocessing = Pipeline([
    ('ZV', VarianceThreshold(threshold=0.000)), # Zero Variance
     ('robust_scaler', RobustScaler()), # Outlier Treatment
    #('imputer', IterativeImputer(max_iter=10, random_state=0))
])
```

Apply preprocessing steps to the training data

In [14]:
```
X_train_preprocessed = preprocessing.fit_transform(X_train)
df_X_train_preprocessed = pd.DataFrame(X_train_preprocessed,columns=preprocessing.get_feature
```

2.2 Correlation

Remove highly correlated variables

They basically give the same info if they have Pearson Correlation > 90%

In [15]:
```
# set up the selector
tr = SmartCorrelatedSelection(
    variables=None,
    method="pearson",
    threshold=0.9,
    missing_values="raise",
    selection_method="model_performance",
    estimator=DecisionTreeClassifier(random_state=1),
    scoring='roc_auc',
    cv=5,
)
```

In [16]:
```
Xt = tr.fit_transform(df_X_train_preprocessed, y_train)
```

List of the variables that highly correlated to each other

In [17]:
```
tr.correlated_feature_sets_
```

Out[17]:
```
[{'After_tax_Net_Profit_Growth_Rate', 'Regular_Net_Profit_Growth_Rate'},
 {'After_tax_net_Interest_Rate',
  'Continuous_interest_rate_after_tax',
  'Non_industry_income_and_expenditure_revenue',
  'Pre_tax_net_Interest_Rate'},
 {'Borrowing_dependency', 'Liability_to_Equity'},
 {'Cash_flow_rate', 'Operating_Funds_to_Liability'},
 {'Current_Liabilities_Equity', 'Current_Liability_to_Equity'},
 {'Current_Liabilities_Liability', 'Current_Liability_to_Liability'},
 {'Debt_ratio', 'Net_worth_Assets'},
 {'Gross_Profit_to_Sales',
  'Operating_Gross_Margin',
  'Realized_Sales_Gross_Margin'},
 {'Net_Income_to_Total_Assets',
  'ROA_A_before_interest_and_after_tax',
  'ROA_B_before_interest_and_depreciation_after_tax'},
 {'Net_Value_Per_Share_A', 'Net_Value_Per_Share_B', 'Net_Value_Per_Share_C'},
 {'Net_profit_before_tax_Paid_in_capital',
  'Per_Share_Net_profit_before_tax_Yuan',
  'Persistent_EPS_in_the_Last_Four_Seasons'},
 {'Operating_Profit_Per_Share_Yuan', 'Operating_profit_Paid_in_capital'},
 {'Revenue_Per_Share_Yuan', 'Working_capitcal_Turnover_Rate'}]
```

List of the variables to drop after the process of remove high correlation

In [18]:
```
tr.features_to_drop_
```

Out[18]:
```
['After_tax_Net_Profit_Growth_Rate',
 'After_tax_net_Interest_Rate',
 'Continuous_interest_rate_after_tax',
 'Non_industry_income_and_expenditure_revenue',
 'Borrowing_dependency',
 'Operating_Funds_to_Liability',
 'Current_Liability_to_Equity',
 'Current_Liability_to_Liability',
 'Debt_ratio',
 'Gross_Profit_to_Sales',
 'Operating_Gross_Margin',
 'Net_Income_to_Total_Assets',
 'ROA_B_before_interest_and_depreciation_after_tax',
 'Net_Value_Per_Share_B',
 'Net_Value_Per_Share_C',
 'Net_profit_before_tax_Paid_in_capital',
 'Per_Share_Net_profit_before_tax_Yuan',
 'Operating_Profit_Per_Share_Yuan',
 'Revenue_Per_Share_Yuan']
```

In [19]:
```
df_X_train_preprocessed_corr = df_X_train_preprocessed.drop(columns=tr.features_to_drop_)
```

After removing varialbles due to Zero-Variance and Highly Correlation, we reduce the number of features from 96 to 75.

The variables that are removed are repeated with other vairables in term of info or giving no useful info at all

In [20]:
```
df_X_train_preprocessed_corr.shape
```

Out[20]:
```
(5114, 75)
```

Columns were removed due to Zero-Variance and Highly Correlation

In [21]:
```
set(X_train.columns) - set(df_X_train_preprocessed_corr.columns)
```

Out[21]:
```
{'After_tax_Net_Profit_Growth_Rate',
 'After_tax_net_Interest_Rate',
 'Borrowing_dependency',
 'Continuous_interest_rate_after_tax',
 'Current_Liability_to_Equity',
 'Current_Liability_to_Liability',
 'Debt_ratio',
 'Gross_Profit_to_Sales',
 'Net_Income_Flag',
 'Net_Income_to_Total_Assets',
 'Net_Value_Per_Share_B',
 'Net_Value_Per_Share_C',
 'Net_profit_before_tax_Paid_in_capital',
 'Non_industry_income_and_expenditure_revenue',
 'Operating_Funds_to_Liability',
 'Operating_Gross_Margin',
 'Operating_Profit_Per_Share_Yuan',
 'Per_Share_Net_profit_before_tax_Yuan',
 'ROA_B_before_interest_and_depreciation_after_tax',
 'Revenue_Per_Share_Yuan'}
```

2.3 Filtering with Binning and Predictive Power

We move on to do the Feature Engineering step.

Applying Automatic Binning Process via Decision Tree.

After that, we remove variables based on Information Value (IV) aka Filtering Method in Variable Selection

In [22]:
```
selection_criteria = {
    "iv": {"strategy": "highest"}
}
```

In [23]:
```
variable_names = list(df_X_train_preprocessed_corr.columns)
len(variable_names)
```

Out[23]: 75

In [24]:
```
binning_process = BinningProcess(variable_names=variable_names,
                                 selection_criteria=selection_criteria
                                 #,max_n_bins=6
                                )
```

In [25]:
```
binning_process.fit(df_X_train_preprocessed_corr, y_train)
```

Out[25]:
```
BinningProcess(selection_criteria={'iv': {'strategy': 'highest'}},
               variable_names=['ROA_C_before_interest_and_depreciation_before_int
erest',
                               'ROA_A_before_interest_and_after_tax',
                               'Realized_Sales_Gross_Margin',
                               'Operating_Profit_Rate',
                               'Pre_tax_net_Interest_Rate',
                               'Operating_Expense_Rate',
                               'Research_and_development_expense_rate',
```

The output of binning process top give info on IV, number of Bins, GINI to remove some variables

In [26]:
```
summary = binning_process.summary()
# Convert the summary to a DataFrame
summary_df = pd.DataFrame(summary)
```

```
summary_df.selected = True
# Sort the DataFrame by IV in descending order
sorted_summary_df = summary_df.sort_values(by='iv', ascending=False)
sorted_summary_df.head(20)
```

Out[26]:

	name	dtype	status	selected	n_bins	iv	
52	Retained_Earnings_to_Total_Assets	numerical	OPTIMAL	True	7	2.980088	0.2
11	Persistent_EPS_in_the_Last_Four_Seasons	numerical	OPTIMAL	True	8	2.899285	0.2
23	Interest_Expense_Ratio	numerical	OPTIMAL	True	8	2.82633	0.2
73	Interest_Coverage_Ratio_Interest_expense_to_EBIT	numerical	OPTIMAL	True	8	2.719549	0.2
72	Degree_of_Financial_Leverage_DFL	numerical	OPTIMAL	True	7	2.701392	0.2
1	ROA_A_before_interest_and_after_tax	numerical	OPTIMAL	True	7	2.670315	0.
0	ROA_C_before_interest_and_depreciation_before_...	numerical	OPTIMAL	True	9	2.669505	0.2
53	Total_income_Total_expense	numerical	OPTIMAL	True	8	2.422825	0.2
4	Pre_tax_net_Interest_Rate	numerical	OPTIMAL	True	8	2.346766	0.2
74	Equity_to_Liability	numerical	OPTIMAL	True	8	2.320075	0.2
25	Net_worth_Assets	numerical	OPTIMAL	True	8	2.319994	0.2
24	Total_debt_Total_net_worth	numerical	OPTIMAL	True	7	2.293904	0.2
70	Net_Income_to_Stockholder_s_Equity	numerical	OPTIMAL	True	8	2.151471	0.2
18	Net_Value_Growth_Rate	numerical	OPTIMAL	True	6	1.997417	0.2
10	Net_Value_Per_Share_A	numerical	OPTIMAL	True	8	1.898265	0.1
22	Quick_Ratio	numerical	OPTIMAL	True	7	1.897452	0.2
71	Liability_to_Equity	numerical	OPTIMAL	True	6	1.894411	0.2
46	Inventory_Working_Capital	numerical	OPTIMAL	True	9	1.686785	0.1
43	Quick_Assets_Current_Liability	numerical	OPTIMAL	True	8	1.619635	0.1
21	Current_Ratio	numerical	OPTIMAL	True	8	1.557376	0.1

Example of Weight-of-Evidence (WoE) output binning for "Retained_Earnings_to_Total_Assets"

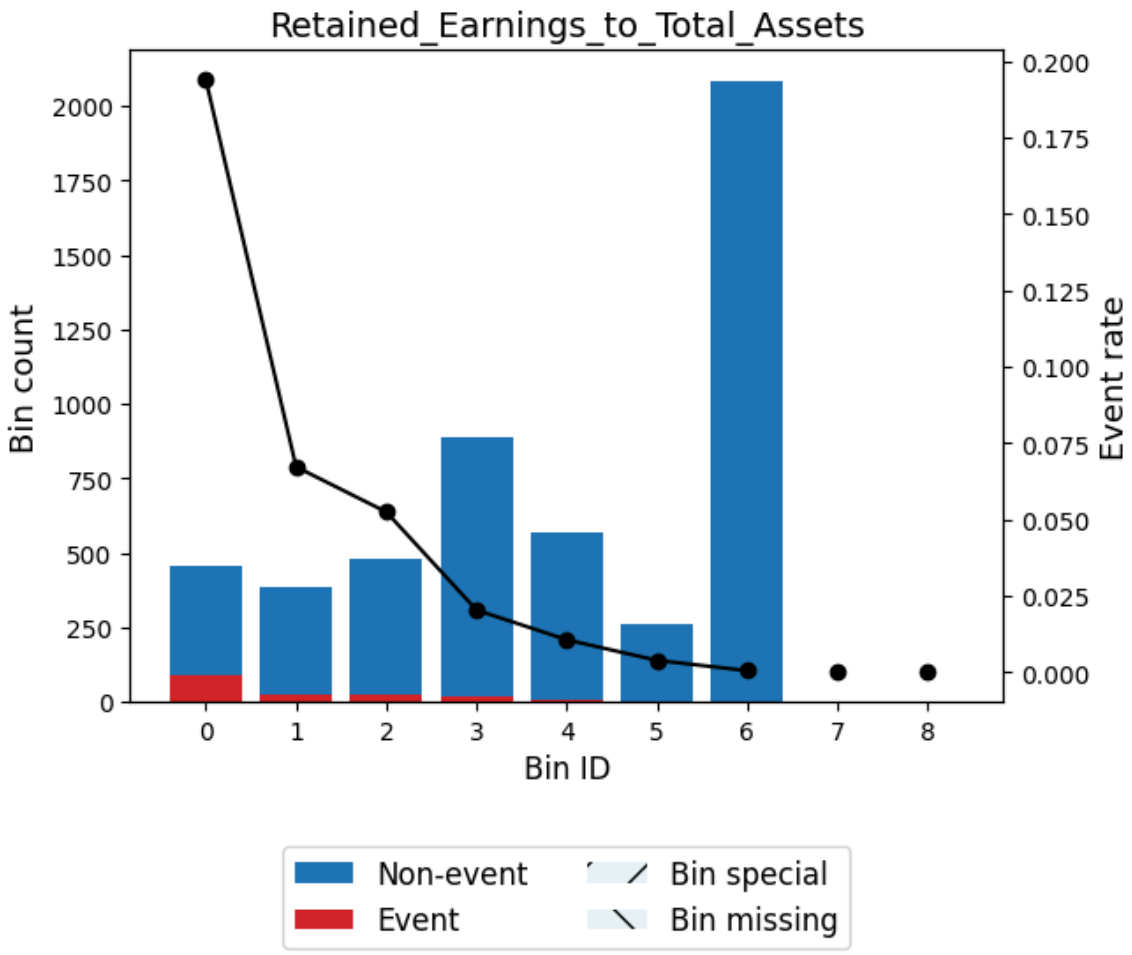
In [27]:

```
optb = binning_process.get_binned_variable("Retained_Earnings_to_Total_Assets")
optb.binning_table.build()
```

Out[27]:

	Bin	Count	Count (%)	Non-event	Event	Event rate	WoE	IV	JS
0	(-inf, -1.67)	454	0.088776	366	88	0.193833	-1.975699	0.907595	0.097984
1	[-1.67, -0.88)	387	0.075675	361	26	0.067183	-0.770214	0.065185	0.007952
2	[-0.88, -0.45)	477	0.093273	452	25	0.052411	-0.506189	0.030464	0.003768
3	[-0.45, -0.11)	886	0.173250	868	18	0.020316	0.474825	0.031480	0.003898
4	[-0.11, 0.07)	567	0.110872	561	6	0.010582	1.136966	0.087538	0.010389
5	[0.07, 0.17)	260	0.050841	259	1	0.003846	2.155833	0.099757	0.010506
6	[0.17, inf)	2083	0.407313	2082	1	0.000480	4.240089	1.758070	0.132000
7	Special	0	0.000000	0	0	0.000000	0.0	0.000000	0.000000
8	Missing	0	0.000000	0	0	0.000000	0.0	0.000000	0.000000
Totals		5114	1.000000	4949	165	0.032264		2.980088	0.266497

In [28]:
```
optb.binning_table.plot(metric="event_rate")
```



Filtering Variable Selection based on IV with threshold = 1.0

In [29]:
```
iv_threshold = 1.0
filtered_summary_df = sorted_summary_df[sorted_summary_df['iv'] > iv_threshold]
```

After this step, we can further reduce variables from 75 to 34 variables

In [30]:
```
selected_filter_IV = list(filtered_summary_df['name'])
len(selected_filter_IV)
```

Out[30]: 34

In [31]:
```
df_X_train_preprocessed_WoE = binning_process.transform(df_X_train_preprocessed_corr, metric=
```

In [32]:
```
df_X_train_preprocessed_WoE.shape
```

Out[32]: (5114, 75)

In [33]:
```
df_X_train_preprocessed_WoE = df_X_train_preprocessed_WoE[selected_filter_IV]
```

2.4 Wrapper Variable Selection using Recursive Feature Elimination (RFE)

In [34]:
```
rfe = RFECV(estimator=DecisionTreeClassifier(random_state=123),
              step=1,
              cv=5,
              scoring='roc_auc',
              min_features_to_select = 10
            )
df_X_train_preprocessed_WoE_RFE = rfe.fit_transform(df_X_train_preprocessed_WoE, y_train)
```

In [35]:
```
selected_features = df_X_train_preprocessed_WoE.columns[rfe.support_]
selected_features
```

Out[35]:
```
Index(['Persistent_EPS_in_the_Last_Four_Seasons', 'Interest_Expense_Ratio',
       'Interest_Coverage_Ratio_Interest_expense_to_EBIT',
       'ROA_C_before_interest_and_depreciation_before_interest',
       'Total_income_Total_expense', 'Pre_tax_net_Interest_Rate',
       'Total_debt_Total_net_worth', 'Net_Value_Growth_Rate',
       'Net_Value_Per_Share_A', 'Quick_Ratio', 'Inventory_Working_Capital',
       'Quick_Assets_Current_Liability', 'Current_Liability_to_Current_Assets',
       'Operating_Profit_Rate', 'Working_capitcal_Turnover_Rate',
       'Working_Capital_Equity', 'Operating_profit_per_person',
       'Cash_Total_Assets', 'Cash_Flow_to_Liability',
       'Current_Liability_to_Assets', 'Cash_flow_rate'],
      dtype='object')
```

In [36]:
```
df_X_train_preprocessed_WoE_RFE = pd.DataFrame(df_X_train_preprocessed_WoE_RFE, columns=selec
df_X_train_preprocessed_WoE_RFE.shape
```

Out[36]: (5114, 21)

After RFE, we can significant reduce variables to 21 variables

2.5 Backward Trimming Performance-based Selection (BART)

In [37]:
```
def backward_variable_selection(data, target_var, random_state=0):
    # Set seed for reproducibility
    np.random.seed(random_state)
    # Split data into test set (20%) and the remaining (80%)
    train_valid, test = train_test_split(data, test_size=0.2, stratify=data[target_var], rand
    # Split the remaining 80% into training (80% of 80%) and validation (20% of 80%)
    train, valid = train_test_split(train_valid, test_size=0.2, stratify=train_valid[target_v
    # Backward variable selection based on AUC using Random Forest
    rf_model = RandomForestClassifier(random_state=random_state)
    rf_param = {
    'max_depth': np.arange(2, 16, 2),
    'n_estimators': np.arange(50, 500, 25),
```

```
    'min_samples_split' : [2, 5, 7, 10, 12],
    'min_samples_leaf' : [1, 3, 5, 7, 10]
    }
    rf_model_CV = RandomizedSearchCV(estimator=rf_model, param_distributions=rf_param,
                                     n_iter=20, scoring='roc_auc', cv=5,
                                     random_state=123, n_jobs = -2)

    rf_model_CV.fit(train.drop(columns=[target_var]), train[target_var])
    predictions = rf_model_CV.predict_proba(valid.drop(columns=[target_var]))[:, 1]

    AUC = roc_auc_score(valid[target_var], predictions)
    removed_predictors = []
    best_AUCs = [AUC]

    X_train_old = train.drop(columns=[target_var])
    y_train = train[target_var]
    X_valid_old = valid.drop(columns=[target_var])
    y_valid = valid[target_var]

    while X_train_old.shape[1] > 1:
        AUCs_temp_vec = []

        for j in range(X_train_old.shape[1]):
            X_train_new = X_train_old.drop(X_train_old.columns[j], axis=1)
            X_valid_new = X_valid_old[X_train_new.columns]  # Ensure the same columns are use

            rf_model.fit(X_train_new, y_train)
            predictions = rf_model.predict_proba(X_valid_new)[:, 1]
            AUC = roc_auc_score(y_valid, predictions)
            AUCs_temp_vec.append(AUC)
        best_index = np.argmax(AUCs_temp_vec)
        removed_predictors.append(X_train_old.columns[best_index])
        best_AUCs.append(AUCs_temp_vec[best_index])
        X_train_old = X_train_old.drop(X_train_old.columns[best_index], axis=1)
        X_valid_old = X_valid_old[X_train_old.columns]  # Ensure the validation set matches t

    remaining_predictors = X_train_old.columns.tolist()
    removed_predictors.extend(remaining_predictors)

    if len(remaining_predictors) > 0:
        rf_model.fit(X_train_old, y_train)
        X_valid_new = valid[remaining_predictors]
    else:
        rf_model = RandomForestClassifier(random_state=random_state)
        rf_model.fit(train[[target_var]], y_train)
        X_valid_new = valid[[target_var]]

    predictions = rf_model.predict_proba(X_valid_new)[:, 1]
    AUC = roc_auc_score(valid[target_var], predictions)
    best_AUCs.append(AUC)

    return removed_predictors, best_AUCs
```

In [38]:
```
combine_df_train_processed = pd.concat([df_X_train_preprocessed_WoE_RFE.reset_index(drop=True
```

In [39]:
```
combine_df_train_processed.columns
```

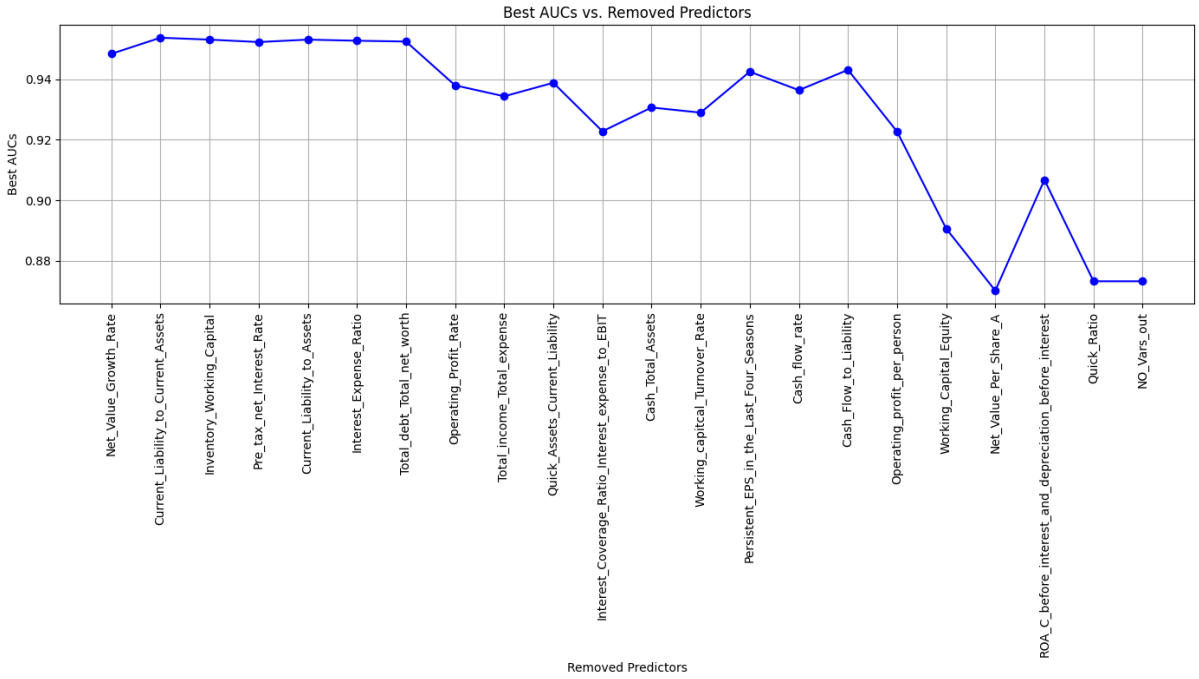
Out[39]:
```
Index(['Persistent_EPS_in_the_Last_Four_Seasons', 'Interest_Expense_Ratio',
       'Interest_Coverage_Ratio_Interest_expense_to_EBIT',
       'ROA_C_before_interest_and_depreciation_before_interest',
       'Total_income_Total_expense', 'Pre_tax_net_Interest_Rate',
       'Total_debt_Total_net_worth', 'Net_Value_Growth_Rate',
       'Net_Value_Per_Share_A', 'Quick_Ratio', 'Inventory_Working_Capital',
       'Quick_Assets_Current_Liability', 'Current_Liability_to_Current_Assets',
       'Operating_Profit_Rate', 'Working_capitcal_Turnover_Rate',
       'Working_Capital_Equity', 'Operating_profit_per_person',
       'Cash_Total_Assets', 'Cash_Flow_to_Liability',
       'Current_Liability_to_Assets', 'Cash_flow_rate', 'Bankrupt'],
      dtype='object')
```

In [40]:
```
result = backward_variable_selection(combine_df_train_processed, "Bankrupt")
removed_predictors, best_AUCs = result
```

In [41]:
```
removed_predictors.append("NO_Vars_out")
```

The plot shows how the best AUC changes when a predictor is removed step-by-step

In [42]:
```
# Plotting
plt.figure(figsize=(14, 8))
plt.plot(removed_predictors, best_AUCs, marker='o', linestyle='-', color='b')
plt.xticks(rotation=90)
plt.xlabel('Removed Predictors')
plt.ylabel('Best AUCs')
plt.title('Best AUCs vs. Removed Predictors')
plt.grid(True)
plt.tight_layout()
# Display the plot
plt.show()
```



after Performance-based variable selection, we only have 10 variables left to enter the Model Fitting

In [43]:
```
df_final_features_train = df_X_train_preprocessed_WoE_RFE.drop(columns=['Net_Value_Growth_Rat
                                                                        'Current_Liability_t
                                                                        'Inventory_Working_C
                                                                        'Pre_tax_net_Interes
                                                                        'Current_Liability_t
                                                                        'Interest_Expense_Ra
                                                                        'ROA_C_before_intere
                                                                        'Total_debt_Total_ne
```

```
                                                                'Persistent_EPS_in_t
                                                                'Cash_Flow_to_Liabil
                                                                'Quick_Assets_Curren

df_final_features_train.shape
```

Out[43]: (5114, 10)

The final list of variables has 10 variables after comprehensive variable selection process

In [44]:
```
df_final_features_train.columns
```

Out[44]:
```
Index(['Interest_Coverage_Ratio_Interest_expense_to_EBIT',
       'Total_income_Total_expense', 'Net_Value_Per_Share_A', 'Quick_Ratio',
       'Operating_Profit_Rate', 'Working_capitcal_Turnover_Rate',
       'Working_Capital_Equity', 'Operating_profit_per_person',
       'Cash_Total_Assets', 'Cash_flow_rate'],
      dtype='object')
```

3. Model Training & Fitting

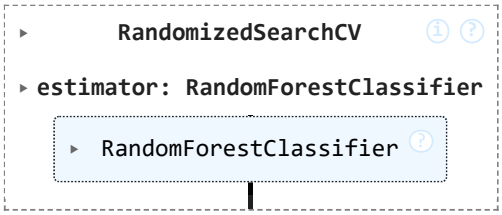
Random Forest Classifier with hyper-parameter tuning

In [45]:
```
rf = RandomForestClassifier(random_state=123)
# Create the parameter grid
rf_param_grid = {
    'max_depth': np.arange(2, 16, 2),
    'n_estimators': np.arange(50, 600, 25),
    'min_samples_split' : [2, 3, 5, 7, 10, 12],
    'min_samples_leaf' : [1, 2, 3, 5, 7, 10]
}
```

In [46]:
```
# Perform RandomizedSearchCV
RF_randomized_roc_auc = RandomizedSearchCV(estimator=rf, param_distributions=rf_param_grid,
                                           n_iter=20, scoring='roc_auc', cv=5,
                                           random_state=123, n_jobs = -2)
```

In [47]:
```
RF_randomized_roc_auc.fit(df_final_features_train, y_train)
```

Out[47]:



In [48]:
```
print(RF_randomized_roc_auc.best_estimator_)
```

```
RandomForestClassifier(max_depth=6, min_samples_leaf=5, min_samples_split=7,
                       n_estimators=150, random_state=123)
```

In [49]:
```
print(RF_randomized_roc_auc.best_score_)
```

```
0.9345209951028501
```

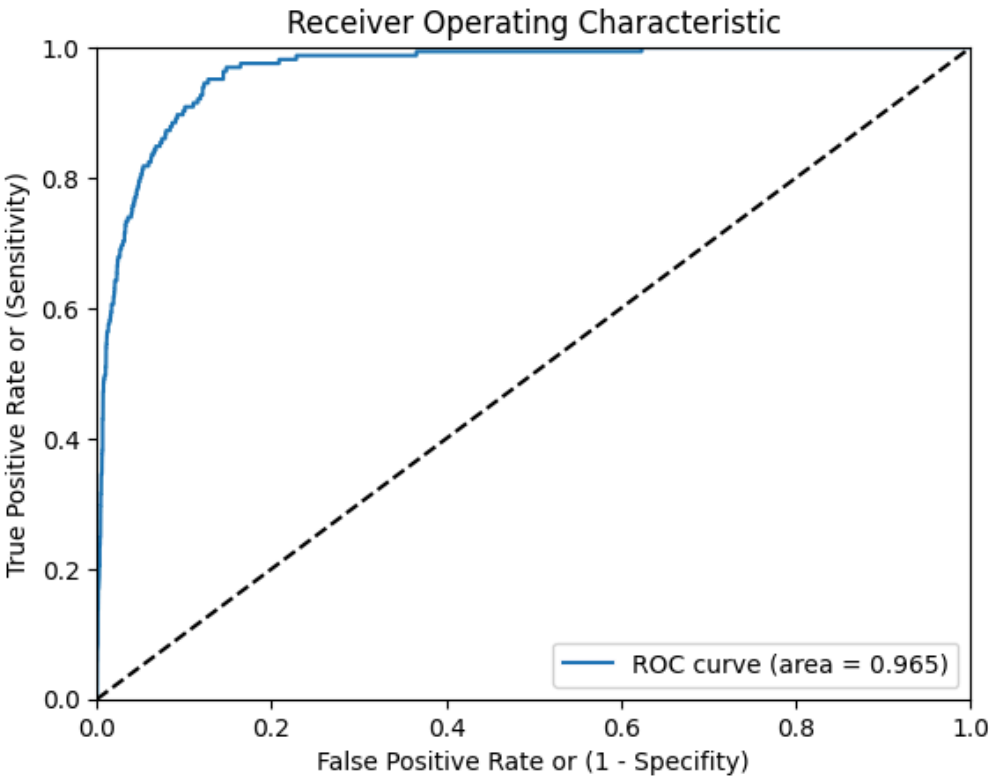
In [50]:
```
y_pred_train_RF = RF_randomized_roc_auc.predict_proba(df_final_features_train)
```

In [51]:
```
y_pred_proba = y_pred_train_RF[:, 1]
y_pred_proba
fpr, tpr, _ = roc_curve(y_train, y_pred_proba)
roc_auc = auc(fpr, tpr)
roc_auc
```

Out[51]: 0.965364291531194

In [52]:
```
roc_auc = auc(fpr, tpr)
# Plot ROC curve
plt.plot(fpr, tpr, label='ROC curve (area = %0.3f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')  # random predictions curve
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate or (1 - Specifity)')
plt.ylabel('True Positive Rate or (Sensitivity)')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
```

Out[52]: <matplotlib.legend.Legend at 0x18a56eced20>



4. Applying pre-processing data steps to Test Set

Applying the same data preperation steps to test set and predict the labels for test set

In [53]:
```
X_test_preprocessed = preprocessing.transform(X_test)
df_X_test_preprocessed = pd.DataFrame(X_test_preprocessed,columns=preprocessing.get_feature_n
```

In [54]:
```
df_X_test_preprocessed.shape
```

Out[54]: (1705, 94)

In [55]:
```
df_X_test_preprocessed_corr = df_X_test_preprocessed.drop(columns=tr.features_to_drop_)
df_X_test_preprocessed_corr.shape
```

Out[55]: (1705, 75)

In [56]:
```
df_X_test_preprocessed_WoE = binning_process.transform(df_X_test_preprocessed_corr, metric="w
```

In [57]:
```
df_X_test_preprocessed_WoE = df_X_test_preprocessed_WoE[selected_filter_IV]
```

In [58]:
```
df_X_test_preprocessed_WoE.shape
```

Out[58]: (1705, 34)

In [59]:
```
df_X_test_preprocessed_WoE_RFE = rfe.transform(df_X_test_preprocessed_WoE)
```

In [60]:
```
df_X_test_preprocessed_WoE_RFE = pd.DataFrame(df_X_test_preprocessed_WoE_RFE, columns=selecte
df_X_test_preprocessed_WoE_RFE.shape
```

Out[60]: (1705, 21)

In [61]:
```
df_final_features_test = df_X_test_preprocessed_WoE_RFE.drop(columns=[    'Net_Value_Growth_Ra
                                                                           'Current_Liability_t
                                                                           'Inventory_Working_C
                                                                           'Pre_tax_net_Interes
                                                                           'Current_Liability_t
                                                                           'Interest_Expense_Ra
                                                                           'ROA_C_before_intere
                                                                           'Total_debt_Total_ne
                                                                           'Persistent_EPS_in_t
                                                                           'Cash_Flow_to_Liabil
                                                                           'Quick_Assets_Curren

df_final_features_test.shape
```

Out[61]: (1705, 10)

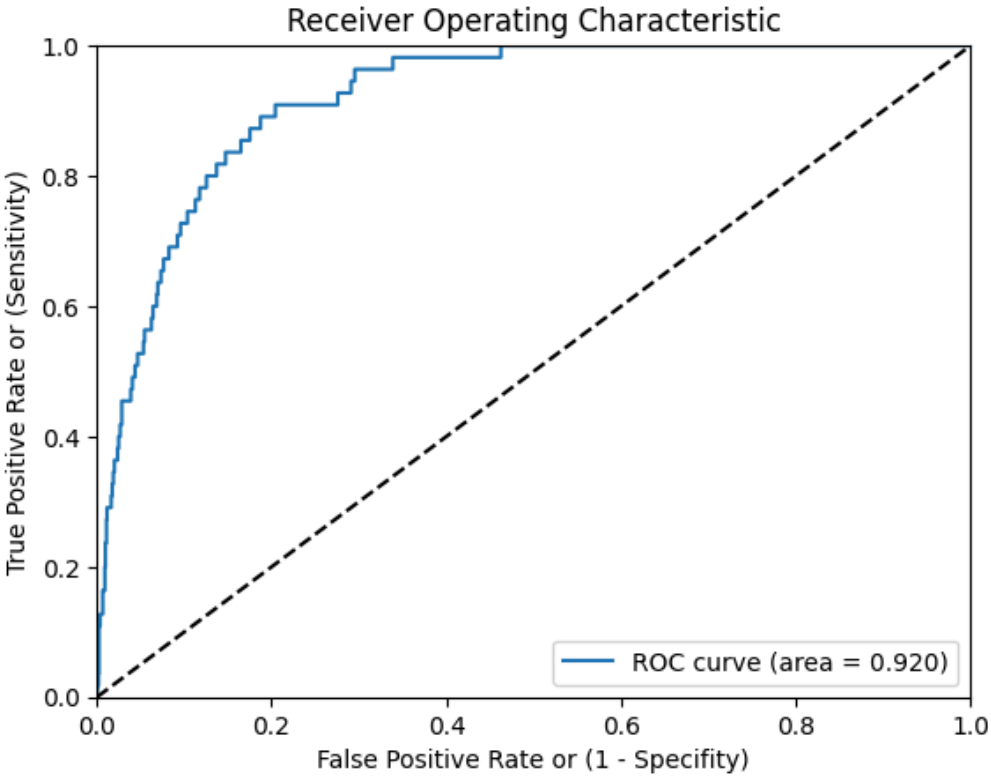
In [62]:
```
y_pred_test_RF = RF_randomized_roc_auc.predict_proba(df_final_features_test)
```

In [63]:
```
y_pred_proba_test = y_pred_test_RF[:, 1]
fpr, tpr, _ = roc_curve(y_test, y_pred_proba_test)
roc_auc = auc(fpr, tpr)
roc_auc
```

Out[63]: 0.9199008264462809

In [64]:
```
roc_auc = auc(fpr, tpr)
# Plot ROC curve
plt.plot(fpr, tpr, label='ROC curve (area = %0.3f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')  # random predictions curve
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate or (1 - Specifity)')
plt.ylabel('True Positive Rate or (Sensitivity)')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
```

Out[64]: <matplotlib.legend.Legend at 0x18a5884b650>



In summary, the final model with only 10 variables can achieve AUC-ROC = 92% on the Test Set