In [25]:
```
# It is for Data manipulation and numerical computing
import pandas as pd
import numpy as np

# It is for Train-test splitting and feature scaling
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# It is for Model evaluation metrics for classification
from sklearn.metrics import classification_report, f1_score, roc_auc_score

# It is for Classical machine learning models
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, StackingClassifier, VotingClassifier

# It is for Gradient boosting model
from xgboost import XGBClassifier

# It is for Dimensionality reduction technique
from sklearn.decomposition import PCA

# It is for Deep learning wrapper for scikit-learn compatibility
from scikeras.wrappers import KerasClassifier

# It is for Keras components for building neural networks
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam

# It is for Data visualisation libraries
import matplotlib.pyplot as plt
import seaborn as sns

# It is for Class imbalance handling technique
from imblearn.combine import SMOTETomek

# It is for Suppress warning messages for cleaner output
import warnings
warnings.filterwarnings("ignore")
```

This cell imports all required libraries for data handling, preprocessing, machine learning, deep learning, class imbalance treatment, dimensionality reduction, visualisation, and model evaluation used throughout the analysis.

In [26]:
```
# Loaded the dataset
df = pd.read_csv(r"C:\Users\Ayush\Desktop\data.csv")
```

In [27]:
```
df.head()# Displayed first five rows
```

Out[27]:

	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	...	Net Income to Total Assets	Total assets to GNP price	No-credit Interval	Gross Profit to Sales	Net Inc Stockhol E
0	1	0.370594	0.424389	0.405750	0.601457	0.601457	0.998969	0.796887	0.808809	0.302646	...	0.716845	0.009219	0.622879	0.601453	0.82
1	1	0.464291	0.538214	0.516730	0.610235	0.610235	0.998946	0.797380	0.809301	0.303556	...	0.795297	0.008323	0.623652	0.610237	0.83
2	1	0.426071	0.499019	0.472295	0.601450	0.601364	0.998857	0.796403	0.808388	0.302035	...	0.774670	0.040003	0.623841	0.601449	0.83
3	1	0.399844	0.451265	0.457733	0.583541	0.583541	0.998700	0.796967	0.808966	0.303350	...	0.739555	0.003252	0.622929	0.583538	0.83
4	1	0.465022	0.538432	0.522298	0.598783	0.598783	0.998973	0.797366	0.809304	0.303475	...	0.795016	0.003878	0.623521	0.598782	0.83

5 rows × 96 columns

Dataset Preview

This table displays the first five observations of the dataset. Each row represents a firm, and each column corresponds to a financial ratio or indicator used in bankruptcy prediction.

The target variable **Bankrupt?** is binary, indicating the bankruptcy status of each firm. The remaining columns consist of financial performance, profitability, leverage, liquidity, and cash flow metrics. The dataset contains a total of 96 variables.

In [28]:
```
df.info()# Displayed dataset structure and data types
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6819 entries, 0 to 6818
Data columns (total 96 columns):
 #   Column                                                  Non-Null Count  Dtype  
---  ------                                                  --------------  -----  
 0   Bankrupt?                                               6819 non-null   int64  
 1    ROA(C) before interest and depreciation before interest  6819 non-null   float64
 2    ROA(A) before interest and % after tax                  6819 non-null   float64
 3    ROA(B) before interest and depreciation after tax       6819 non-null   float64
 4    Operating Gross Margin                                  6819 non-null   float64
 5    Realized Sales Gross Margin                             6819 non-null   float64
 6    Operating Profit Rate                                   6819 non-null   float64
 7    Pre-tax net Interest Rate                               6819 non-null   float64
 8    After-tax net Interest Rate                             6819 non-null   float64
 9    Non-industry income and expenditure/revenue             6819 non-null   float64
 10   Continuous interest rate (after tax)                    6819 non-null   float64
 11   Operating Expense Rate                                  6819 non-null   float64
 12   Research and development expense rate                   6819 non-null   float64
 13   Cash flow rate                                          6819 non-null   float64
 14   Interest-bearing debt interest rate                     6819 non-null   float64
 15   Tax rate (A)                                            6819 non-null   float64
 16   Net Value Per Share (B)                                 6819 non-null   float64
 17   Net Value Per Share (A)                                 6819 non-null   float64
 18   Net Value Per Share (C)                                 6819 non-null   float64
 19   Persistent EPS in the Last Four Seasons                 6819 non-null   float64
 20   Cash Flow Per Share                                     6819 non-null   float64
 21   Revenue Per Share (Yuan ¥)                              6819 non-null   float64
 22   Operating Profit Per Share (Yuan ¥)                     6819 non-null   float64
 23   Per Share Net profit before tax (Yuan ¥)                6819 non-null   float64
 24   Realized Sales Gross Profit Growth Rate                 6819 non-null   float64
 25   Operating Profit Growth Rate                            6819 non-null   float64
 26   After-tax Net Profit Growth Rate                        6819 non-null   float64
 27   Regular Net Profit Growth Rate                          6819 non-null   float64
 28   Continuous Net Profit Growth Rate                       6819 non-null   float64
 29   Total Asset Growth Rate                                 6819 non-null   float64
 30   Net Value Growth Rate                                   6819 non-null   float64
 31   Total Asset Return Growth Rate Ratio                    6819 non-null   float64
 32   Cash Reinvestment %                                     6819 non-null   float64
 33   Current Ratio                                           6819 non-null   float64
 34   Quick Ratio                                             6819 non-null   float64
 35   Interest Expense Ratio                                  6819 non-null   float64
 36   Total debt/Total net worth                              6819 non-null   float64
 37   Debt ratio %                                            6819 non-null   float64
 38   Net worth/Assets                                        6819 non-null   float64
 39   Long-term fund suitability ratio (A)                    6819 non-null   float64
 40   Borrowing dependency                                    6819 non-null   float64
 41   Contingent liabilities/Net worth                        6819 non-null   float64
 42   Operating profit/Paid-in capital                        6819 non-null   float64
 43   Net profit before tax/Paid-in capital                   6819 non-null   float64
```

```
44   Inventory and accounts receivable/Net value            6819 non-null   float64
45   Total Asset Turnover                                   6819 non-null   float64
46   Accounts Receivable Turnover                           6819 non-null   float64
47   Average Collection Days                                6819 non-null   float64
48   Inventory Turnover Rate (times)                        6819 non-null   float64
49   Fixed Assets Turnover Frequency                        6819 non-null   float64
50   Net Worth Turnover Rate (times)                        6819 non-null   float64
51   Revenue per person                                     6819 non-null   float64
52   Operating profit per person                            6819 non-null   float64
53   Allocation rate per person                             6819 non-null   float64
54   Working Capital to Total Assets                        6819 non-null   float64
55   Quick Assets/Total Assets                              6819 non-null   float64
56   Current Assets/Total Assets                            6819 non-null   float64
57   Cash/Total Assets                                      6819 non-null   float64
58   Quick Assets/Current Liability                         6819 non-null   float64
59   Cash/Current Liability                                 6819 non-null   float64
60   Current Liability to Assets                            6819 non-null   float64
61   Operating Funds to Liability                           6819 non-null   float64
62   Inventory/Working Capital                              6819 non-null   float64
63   Inventory/Current Liability                            6819 non-null   float64
64   Current Liabilities/Liability                          6819 non-null   float64
65   Working Capital/Equity                                 6819 non-null   float64
66   Current Liabilities/Equity                             6819 non-null   float64
67   Long-term Liability to Current Assets                  6819 non-null   float64
68   Retained Earnings to Total Assets                      6819 non-null   float64
69   Total income/Total expense                             6819 non-null   float64
70   Total expense/Assets                                   6819 non-null   float64
71   Current Asset Turnover Rate                            6819 non-null   float64
72   Quick Asset Turnover Rate                              6819 non-null   float64
73   Working capitcal Turnover Rate                         6819 non-null   float64
74   Cash Turnover Rate                                     6819 non-null   float64
75   Cash Flow to Sales                                     6819 non-null   float64
76   Fixed Assets to Assets                                 6819 non-null   float64
77   Current Liability to Liability                         6819 non-null   float64
78   Current Liability to Equity                            6819 non-null   float64
79   Equity to Long-term Liability                          6819 non-null   float64
80   Cash Flow to Total Assets                              6819 non-null   float64
81   Cash Flow to Liability                                 6819 non-null   float64
82   CFO to Assets                                          6819 non-null   float64
83   Cash Flow to Equity                                    6819 non-null   float64
84   Current Liability to Current Assets                    6819 non-null   float64
85   Liability-Assets Flag                                  6819 non-null   int64
86   Net Income to Total Assets                             6819 non-null   float64
87   Total assets to GNP price                              6819 non-null   float64
88   No-credit Interval                                     6819 non-null   float64
89   Gross Profit to Sales                                  6819 non-null   float64
90   Net Income to Stockholder's Equity                     6819 non-null   float64
91   Liability to Equity                                    6819 non-null   float64
92   Degree of Financial Leverage (DFL)                     6819 non-null   float64
```

```
 93   Interest Coverage Ratio (Interest expense to EBIT)      6819 non-null   float64
 94   Net Income Flag                                         6819 non-null   int64
 95   Equity to Liability                                     6819 non-null   float64
dtypes: float64(93), int64(3)
memory usage: 5.0 MB
```

Dataset Structure Overview

This cell examines the structure of the dataset, including the number of rows, columns, data types, and memory usage.

In [29]:
```
df.describe() # Generated descriptive statistics for all features
```

Out[29]:

	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	...	Net Income to Total Assets	Total assets to GNP price
count	6819.000000	6819.000000	6819.000000	6819.000000	6819.000000	6819.000000	6819.000000	6819.000000	6819.000000	6819.000000	...	6819.000000	6.819000e+03
mean	0.032263	0.505180	0.558625	0.553589	0.607948	0.607929	0.998755	0.797190	0.809084	0.303623	...	0.807760	1.862942e+07
std	0.176710	0.060686	0.065620	0.061595	0.016934	0.016916	0.013010	0.012869	0.013601	0.011163	...	0.040332	3.764501e+08
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000e+00
25%	0.000000	0.476527	0.535543	0.527277	0.600445	0.600434	0.998969	0.797386	0.809312	0.303466	...	0.796750	9.036205e-04
50%	0.000000	0.502706	0.559802	0.552278	0.605997	0.605976	0.999022	0.797464	0.809375	0.303525	...	0.810619	2.085213e-03
75%	0.000000	0.535563	0.589157	0.584105	0.613914	0.613842	0.999095	0.797579	0.809469	0.303585	...	0.826455	5.269777e-03
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...	1.000000	9.820000e+09

8 rows × 96 columns

Descriptive Statistics

This cell computes summary statistics (mean, standard deviation, minimum, maximum, and quartiles) for all numerical variables.

In [30]:
```
df.isnull().sum() # Counted missing values in each column
```

Out[30]:
```
Bankrupt?                                                  0
 ROA(C) before interest and depreciation before interest   0
 ROA(A) before interest and % after tax                    0
 ROA(B) before interest and depreciation after tax         0
 Operating Gross Margin                                    0
                                                          ..
 Liability to Equity                                       0
 Degree of Financial Leverage (DFL)                        0
 Interest Coverage Ratio (Interest expense to EBIT)        0
 Net Income Flag                                           0
 Equity to Liability                                       0
Length: 96, dtype: int64
```

Missing Value Check

This cell checks whether any variables contain missing values.

In [31]:
```
df.duplicated().sum() # Counted duplicated rows
```

Out[31]:
```
0
```

Duplicate Record Check

This cell checks for duplicate observations in the dataset.

In [32]:
```
# Computed absolute correlation between each numerical feature and the target variable
cor = df.corr(numeric_only=True)["Bankrupt?"].abs()

# Selected features with correlation greater than the defined threshold
cols = cor[cor > 0.2].index.tolist()  # Threshold = 0.2

# Created a filtered DataFrame containing only the selected features
filtered_df = df[cols]
```

Feature Selection Based on Correlation

This cell computes the absolute Pearson correlation between each numerical feature and the target variable **Bankrupt?**. Features with correlation values greater than a specified threshold are selected to form a reduced dataset.

In [34]:
```
# Removed the target variable to create the feature matrix
X_no_target = df.drop("Bankrupt?", axis=1)

# Extracted the target variable for bankruptcy classification
```

```
y = df["Bankrupt?"]
```

Separation of Features and Target Variable

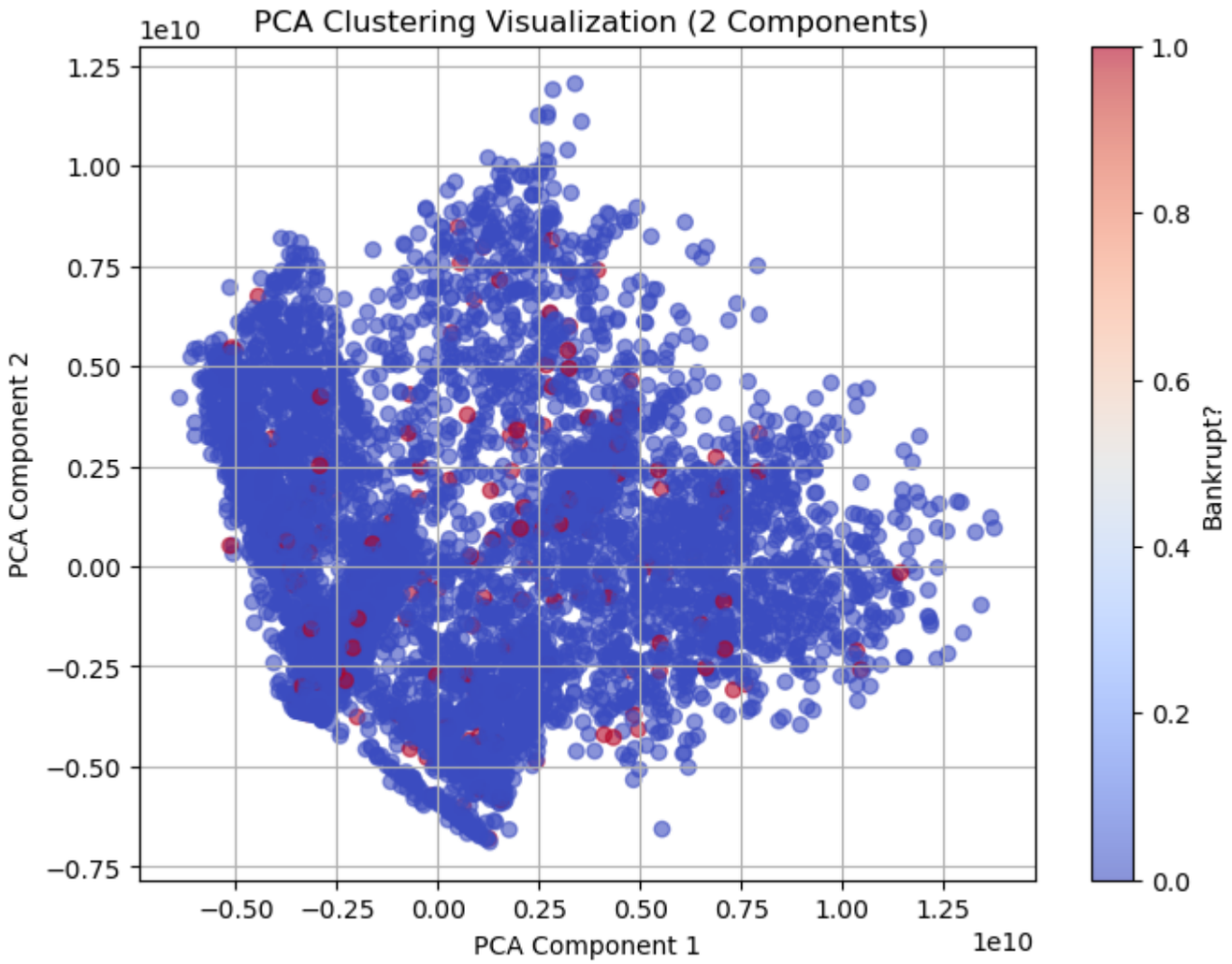
This cell separates the independent variables (features) from the dependent variable (**Bankrupt?**) in preparation for dimensionality reduction and subsequent modelling steps.

In [35]:
```
# Initialised PCA to retain 95% of the total variance
pca = PCA(n_components=0.95, random_state=42)

# Fitted PCA on the feature set and transform the data
X_pca = pca.fit_transform(X_no_target)
```

Principal Component Analysis (PCA) Fitting

This cell applies Principal Component Analysis (PCA) to the feature set to reduce dimensionality while retaining 95% of the total variance.

In [36]:
```
# Initialise PCA with two components for 2D visualisation
pca_vis = PCA(n_components=2)

# Fit PCA on the feature set and transform it into two dimensions
X_pca_vis = pca.fit_transform(X_no_target)

# Create a scatter plot of the two PCA components
plt.figure(figsize=(8, 6))
plt.scatter(
    X_pca_vis[:, 0],
    X_pca_vis[:, 1],
    c=y,                    # Colour points by bankruptcy status
    cmap='coolwarm',
    alpha=0.6
)

# Set plot title and axis labels
plt.title("PCA Clustering Visualization (2 Components)")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")

# Add grid and colour bar for clarity
plt.grid(True)
plt.colorbar(label="Bankrupt?")

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



PCA Clustering Visualisation (2 Components)

This figure presents a two-dimensional projection of the dataset obtained using Principal Component Analysis (PCA). The original high-dimensional feature space is reduced to two principal components (PCA Component 1 and PCA Component 2) solely for visualisation purposes.

Each point in the plot represents an individual firm. Points are coloured according to the binary target variable **Bankrupt?**, where different colours indicate bankrupt and non-bankrupt observations.

The plot illustrates how observations are distributed in the reduced feature space and highlights the degree of overlap between the two classes. No clear linear separation between bankrupt and non-bankrupt firms is observed in the two-dimensional PCA space, indicating that bankruptcy classification likely depends on complex, higher-dimensional relationships rather than simple low-dimensional boundaries.

It is important to note that this PCA transformation is used only for exploratory visual analysis and is not utilised directly for model training or prediction.

In [37]:
```
# Separated feature variables from the target variable
X = df.drop("Bankrupt?", axis=1)   # Feature matrix
y = df["Bankrupt?"]                # Target vector

# Imported function for splitting the dataset
from sklearn.model_selection import train_test_split

# Splitted data into training and testing sets with stratification
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,      # 20% of data used for testing
    stratify=y,         # Preserved class distribution
    random_state=42     # Ensure the reproducibility
)
```

Train–Test Split

This cell separates the dataset into feature variables and the target variable (**Bankrupt?**) and then splits the data into training and testing sets using a stratified sampling approach.

In [38]:
```
# Initialised the standard scaler
scaler = StandardScaler()

# Fitted the scaler on the training data and transform it
X_train_scaled = scaler.fit_transform(X_train)

# Applied the same scaling transformation to the test data
X_test_scaled = scaler.transform(X_test)
```

Feature Scaling

This cell applies standardisation to the feature variables so that each feature has a mean of zero and a standard deviation of one. The scaler is fitted on the training data and then applied to the test data.

In [39]:
```
# Initialised SMOTETomek with a fixed random state for reproducibility
smt = SMOTETomek(random_state=42)

# Applied SMOTETomek to the scaled training data
X_train_res, y_train_res = smt.fit_resample(X_train_scaled, y_train)

# Displayed class distribution before resampling
print("Before balancing:\n", y_train.value_counts())

# Displayed class distribution after resampling
```

```
print("\nAfter balancing:\n", y_train_res.value_counts())
```

```
Before balancing:
 Bankrupt?
0    5279
1     176
Name: count, dtype: int64

After balancing:
 Bankrupt?
0    5278
1    5278
Name: count, dtype: int64
```

Class Distribution Before and After Resampling

This output shows the class distribution of the target variable **Bankrupt?** in the training dataset before and after applying the SMOTETomek resampling technique.

Before resampling, the dataset is highly imbalanced, with a substantially larger number of non-bankrupt observations compared to bankrupt observations.

After applying SMOTETomek, the training dataset becomes balanced, with an equal number of observations in each class. This balanced dataset is used for subsequent model training.

In [40]:
```
# Define a function to create the deep neural network architecture
def create_dnn():
    # Initialise a sequential neural network model
    model = Sequential([
        # Input layer and first hidden layer
        Dense(128, activation='relu', input_shape=(X_train_res.shape[1],)),

        # Hidden layers with ReLU activation
        Dense(256, activation='relu'),
        Dense(512, activation='relu'),
        Dense(128, activation='relu'),
        Dense(64, activation='relu'),
        Dense(32, activation='relu'),
        Dense(16, activation='relu'),
        Dense(8, activation='relu'),
        Dense(4, activation='relu'),

        # Output layer with sigmoid activation for binary classification
        Dense(1, activation='sigmoid')
    ])

    # Compile the model with Adam optimizer and binary cross-entropy loss
    model.compile(
        optimizer=Adam(),
        loss='binary_crossentropy',
```

```
        metrics=['accuracy']
    )

    # Return the compiled model
    return model


# Wrap the Keras model for compatibility with scikit-learn
keras_model = KerasClassifier(
    model=create_dnn,
    epochs=150,                 # Maximum number of training epochs
    batch_size=128,             # Number of samples per gradient update
    validation_split=0.2,       # Proportion of training data used for validation
    verbose=0,                  # Suppress training output
    callbacks=[
        EarlyStopping(
            patience=20,        # Stopped training if validation performance does not improve
            restore_best_weights=True
        )
    ]
)
```

Deep Neural Network (DNN) Model Definition and Configuration

This cell defines a deep neural network architecture for binary classification using Keras and wraps the model using `KerasClassifier` to enable integration with scikit-learn workflows.

In [41]:
```
# Initialised Random Forest classifier with class balancing
rf = RandomForestClassifier(
    class_weight="balanced",
    random_state=42
)

# Initialised XGBoost classifier with log loss as the evaluation metric
xgb = XGBClassifier(
    eval_metric="logloss",
    use_label_encoder=False,
    random_state=42
)

# Initialised Logistic Regression classifier with class balancing and increased iterations
lr = LogisticRegression(
    class_weight="balanced",
    max_iter=2000
)
```

Machine Learning Model Initialisation

This cell initialises classical machine learning models used for bankruptcy classification, including Random Forest, XGBoost, and Logistic Regression. Class weighting is applied where applicable to account for class imbalance.

In [42]:
```
# Stored models in a dictionary for iterative training and evaluation
models = {
    "DNN": keras_model,
    "Random Forest": rf,
    "XGBoost": xgb,
    "Logistic Regression": lr
}

# Initialised list to store evaluation results
results = []

# Iterated through each model
for name, model in models.items():
    print(f"\nTraining {name}...")

    # Trained the model using the resampled training data
    model.fit(X_train_res, y_train_res)

    # Generated class predictions on the test data
    y_pred = model.predict(X_test_scaled)

    # Generated predicted probabilities for the positive class
    y_proba = model.predict_proba(X_test_scaled)[:, 1]

    # Computed evaluation metrics
    f1 = f1_score(y_test, y_pred)
    auc = roc_auc_score(y_test, y_proba)

    # Stored model name and performance metrics
    results.append((name, f1, auc))

    # Displayed classification results
    print("\n====", name, "====")
    print(classification_report(y_test, y_pred, digits=4))
    print("F1 Score:", f1)
    print("ROC AUC:", auc)
```

```
Training DNN...

==== DNN ====
              precision    recall  f1-score   support

           0     0.9781    0.9833    0.9807      1320
           1     0.4054    0.3409    0.3704        44

    accuracy                         0.9626      1364
   macro avg     0.6918    0.6621    0.6756      1364
weighted avg     0.9597    0.9626    0.9610      1364

F1 Score: 0.37037037037037035
ROC AUC: 0.833961776859504

Training Random Forest...

==== Random Forest ====
              precision    recall  f1-score   support

           0     0.9877    0.9742    0.9809      1320
           1     0.4516    0.6364    0.5283        44

    accuracy                         0.9633      1364
   macro avg     0.7197    0.8053    0.7546      1364
weighted avg     0.9704    0.9633    0.9663      1364

F1 Score: 0.5283018867924528
ROC AUC: 0.9441546143250689

Training XGBoost...

==== XGBoost ====
              precision    recall  f1-score   support

           0     0.9855    0.9773    0.9814      1320
           1     0.4545    0.5682    0.5051        44

    accuracy                         0.9641      1364
   macro avg     0.7200    0.7727    0.7432      1364
weighted avg     0.9684    0.9641    0.9660      1364

F1 Score: 0.5050505050505051
ROC AUC: 0.937534435261708

Training Logistic Regression...

==== Logistic Regression ====
              precision    recall  f1-score   support
```

```
           0     0.9924    0.8864    0.9364      1320
           1     0.1892    0.7955    0.3057        44

    accuracy                         0.8834      1364
   macro avg     0.5908    0.8409    0.6210      1364
weighted avg     0.9665    0.8834    0.9160      1364

F1 Score: 0.3056768558951965
ROC AUC: 0.9159263085399449
```

Baseline Model Performance Summary

This output reports classification performance for four baseline models (DNN, Random Forest, XGBoost, and Logistic Regression) evaluated on the held-out test set.

For each model, the following metrics are presented:

- **Precision, recall, and F1-score** for each class (0 = non-bankrupt, 1 = bankrupt)
- **Accuracy**, **macro average**, and **weighted average** scores
- Overall **F1 Score** and **ROC AUC** based on predicted probabilities

Across the four models:

- **Random Forest** achieves the strongest balance of minority-class performance, producing the highest reported **F1 score for the bankrupt class** (F1 ≈ 0.5283) and the highest **ROC AUC** (≈ 0.9442).
- **XGBoost** performs similarly but with slightly lower minority-class recall and overall F1 compared to Random Forest (F1 ≈ 0.5051; ROC AUC ≈ 0.9375).
- The **DNN** produces a lower minority-class recall and F1 than tree-based models (F1 ≈ 0.3704; ROC AUC ≈ 0.8340).
- **Logistic Regression** shows high recall for the bankrupt class but low precision, resulting in a lower minority-class F1 (F1 ≈ 0.3057), while still achieving a high ROC AUC (≈ 0.9159), indicating ranking ability despite threshold-based errors.

These results provide a baseline comparison prior to evaluating ensemble approaches (e.g., stacking and voting).

In [43]:
```
# Imported stacking classifier for building stacking ensembles
from sklearn.ensemble import StackingClassifier

# Stacked ensemble using Random Forest and XGBoost as base models,
# with Logistic Regression as the final meta-classifier
stack1 = StackingClassifier(
    estimators=[("rf", rf), ("xgb", xgb)],
    final_estimator=LogisticRegression()
)

# Stacked ensemble using Random Forest, XGBoost, and Logistic Regression as base models,
# with Logistic Regression as the final meta-classifier
stack2 = StackingClassifier(
    estimators=[("rf", rf), ("xgb", xgb), ("lr", lr)],
```

```
    final_estimator=LogisticRegression()
)

# Stacked ensemble using XGBoost and Logistic Regression as base models,
# with Random Forest as the final meta-classifier
stack3 = StackingClassifier(
    estimators=[("xgb", xgb), ("lr", lr)],
    final_estimator=RandomForestClassifier()
)

# Softed voting ensemble that averages predicted probabilities from base models
voting = VotingClassifier(
    estimators=[("rf", rf), ("xgb", xgb), ("lr", lr)],
    voting='soft'
)
```

Ensemble Model Definition (Stacking and Voting)

This cell defines ensemble classifiers to combine predictions from multiple base models. Stacking ensembles are created by training a meta-classifier on the outputs of selected base estimators, while a soft voting ensemble combines predicted probabilities across models.

In [44]:
```
# Imported ROC curve utilities and confusion matrix visualisation
from sklearn.metrics import roc_curve, RocCurveDisplay, ConfusionMatrixDisplay

# Defined models to evaluate (base models + stacking/voting ensembles)
models = {
    "Random Forest": rf,
    "XGBoost": xgb,
    "Logistic Regression": lr,
    "Stack RF+XGB": stack1,
    "Stack RF+XGB+LR": stack2,
    "Stack XGB+LR (RF Final)": stack3,
    "Voting Ensemble": voting
}

# Initialised list to store model evaluation results
results = []

# Defined a function to plot ROC curve and confusion matrix for a given model
def plot_model_evaluation(model_name, y_true, y_pred, y_proba):
    # Create a figure with two subplots: ROC curve (left) and confusion matrix (right)
    fig, axs = plt.subplots(1, 2, figsize=(12, 5))
    fig.suptitle(f"📊 Evaluation for {model_name}", fontsize=14)

    # ---- ROC Curve ----
    # Computed false positive rate and true positive rate across thresholds
    fpr, tpr, _ = roc_curve(y_true, y_proba)
```

```
# Plotted ROC curve and display AUC in the legend
axs[0].plot(fpr, tpr, label=f"AUC: {roc_auc_score(y_true, y_proba):.2f}")

# Plotted diagonal reference line for random classifier performance
axs[0].plot([0, 1], [0, 1], 'k--')

# Labeled ROC plot
axs[0].set_title("ROC Curve")
axs[0].set_xlabel("False Positive Rate")
axs[0].set_ylabel("True Positive Rate")
axs[0].legend()

# ---- Confusion Matrix ----
# Plotted confusion matrix using true labels and predicted classes
ConfusionMatrixDisplay.from_predictions(
    y_true, y_pred,
    ax=axs[1],
    cmap="Blues",
    colorbar=False
)
axs[1].set_title("Confusion Matrix")

# Improved layout spacing and display the plots
plt.tight_layout()
plt.show()
```

Ensemble Evaluation Setup and Visual Diagnostics

This cell defines a set of machine learning and ensemble models to be evaluated and creates a reusable evaluation function. The evaluation function generates:

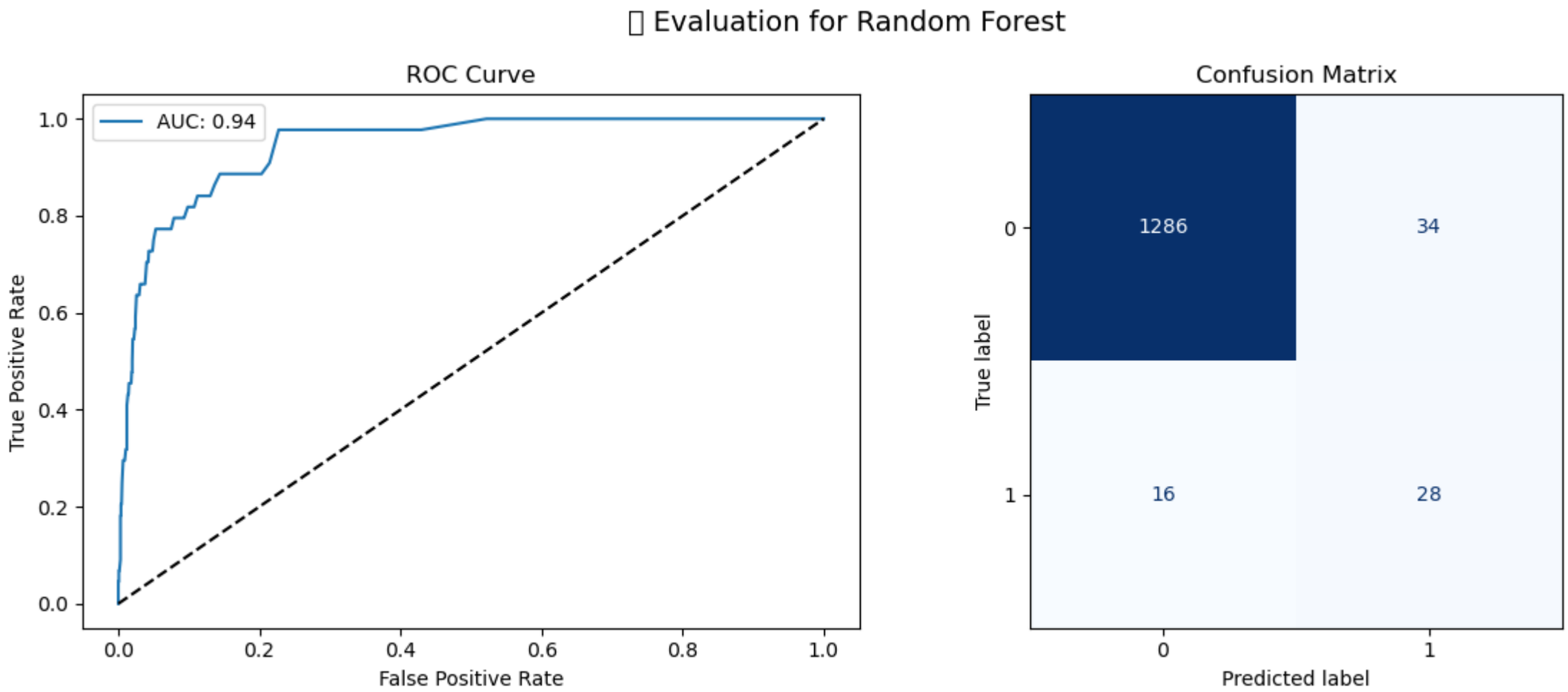
1. an ROC curve using predicted probabilities, and
2. a confusion matrix using predicted class labels,

to support visual comparison of model performance.

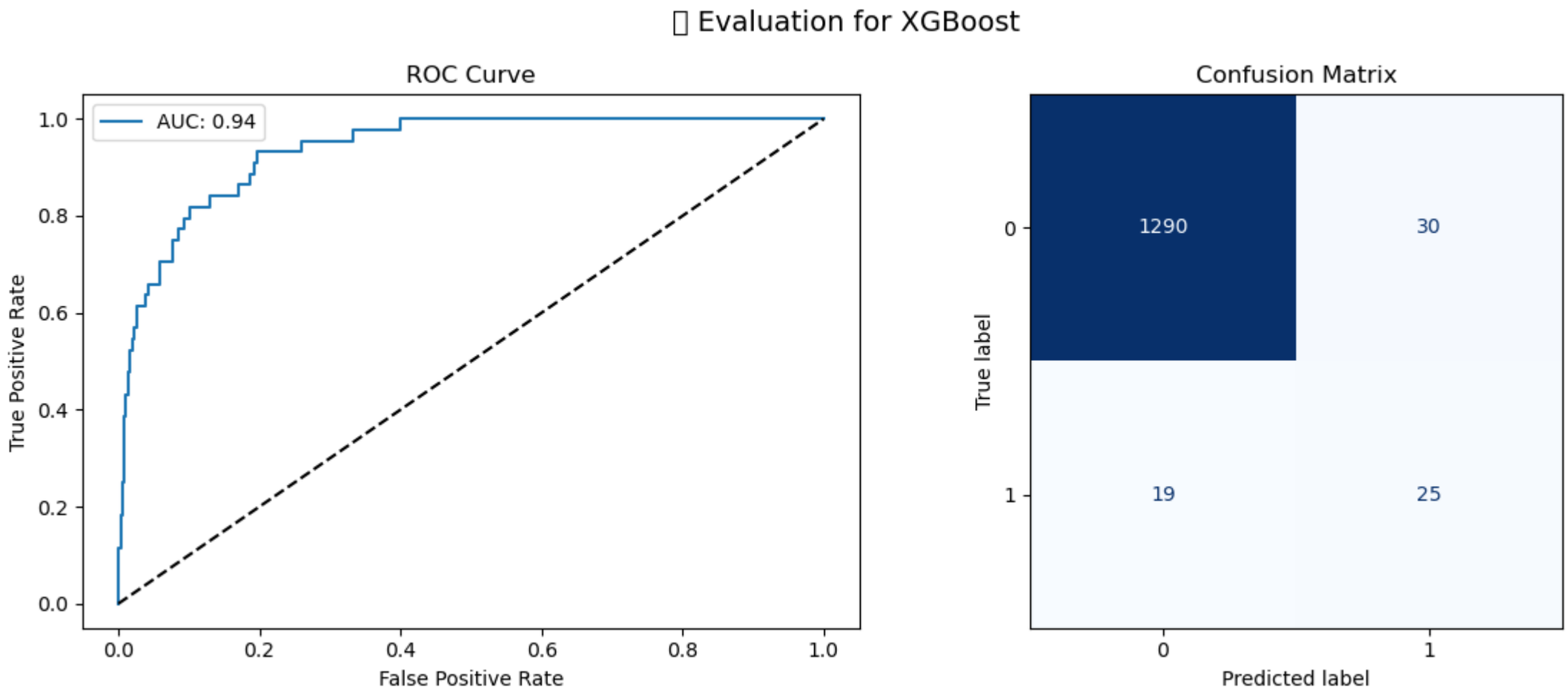
In [45]:
```
# Iterated through each base and ensemble model
for name, model in models.items():
    print(f"\nTraining {name}...")

    # Trained the model on the resampled training data
    model.fit(X_train_res, y_train_res)

    # Generated class predictions on the scaled test data
    y_pred = model.predict(X_test_scaled)

    # Generated predicted probabilities for the positive class
```

```
    y_proba = model.predict_proba(X_test_scaled)[:, 1]

    # Computed evaluation metrics
    f1 = f1_score(y_test, y_pred)
    auc = roc_auc_score(y_test, y_proba)

    # Stored model name and performance metrics
    results.append((name, f1, auc))

    # Displayed classification report
    print(f"\n=== {name} ===")
    print(classification_report(y_test, y_pred, digits=4))

    # Generated ROC curve and confusion matrix plots
    plot_model_evaluation(name, y_test, y_pred, y_proba)
```

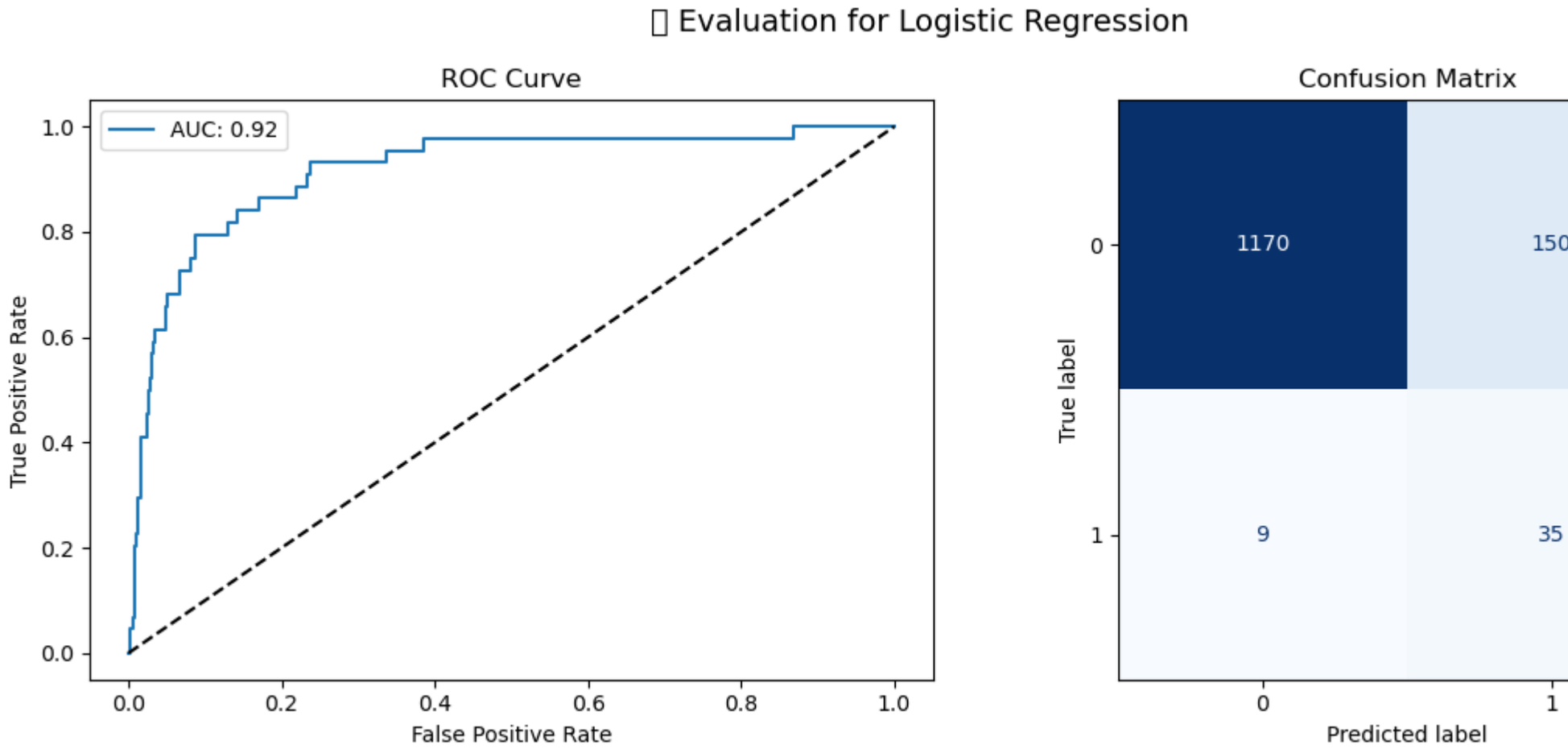
```
Training Random Forest...

=== Random Forest ===
              precision    recall  f1-score   support

           0     0.9877    0.9742    0.9809      1320
           1     0.4516    0.6364    0.5283        44

    accuracy                         0.9633      1364
   macro avg     0.7197    0.8053    0.7546      1364
weighted avg     0.9704    0.9633    0.9663      1364
```



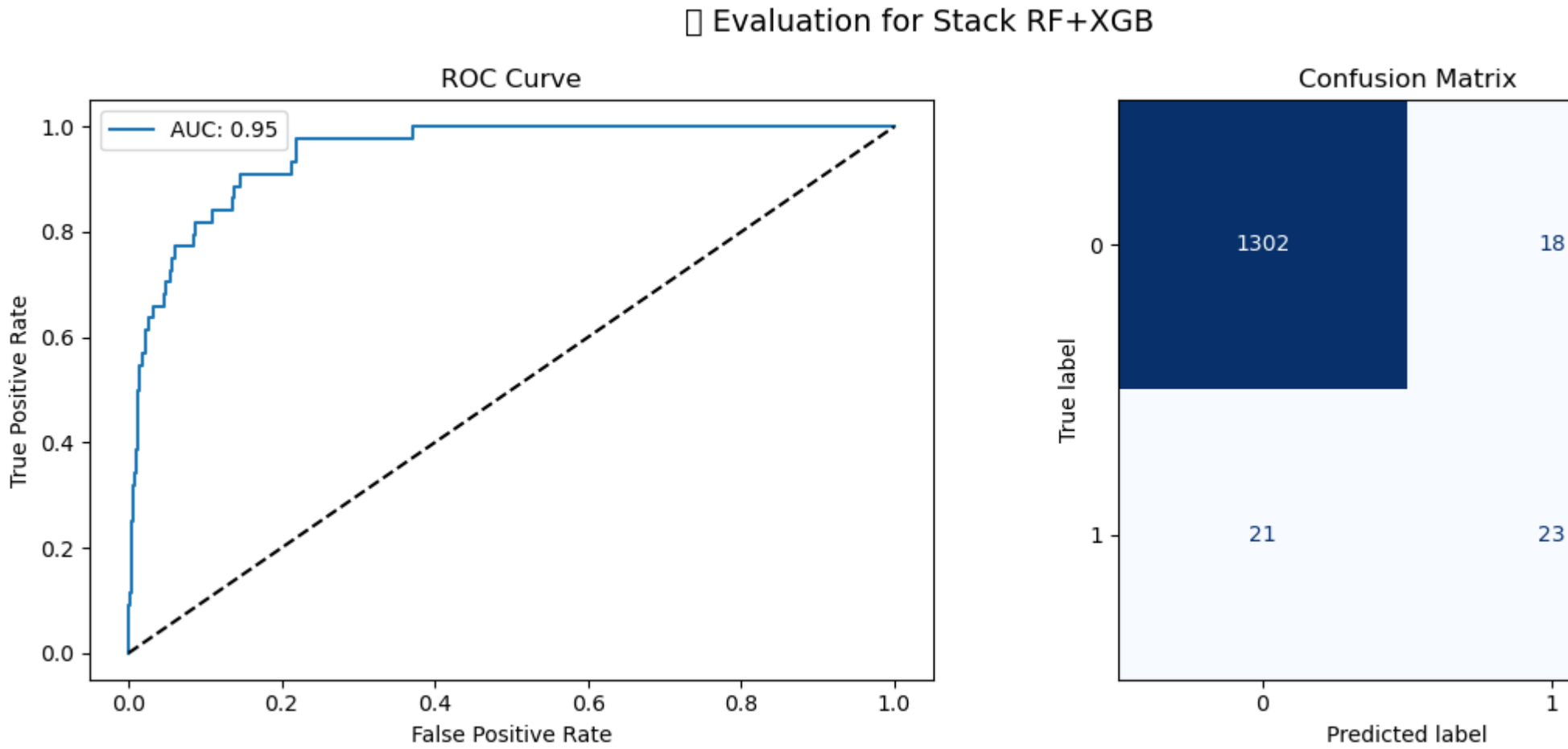

```
Training XGBoost...

=== XGBoost ===
              precision    recall  f1-score   support

           0     0.9855    0.9773    0.9814      1320
           1     0.4545    0.5682    0.5051        44

    accuracy                         0.9641      1364
   macro avg     0.7200    0.7727    0.7432      1364
weighted avg     0.9684    0.9641    0.9660      1364
```



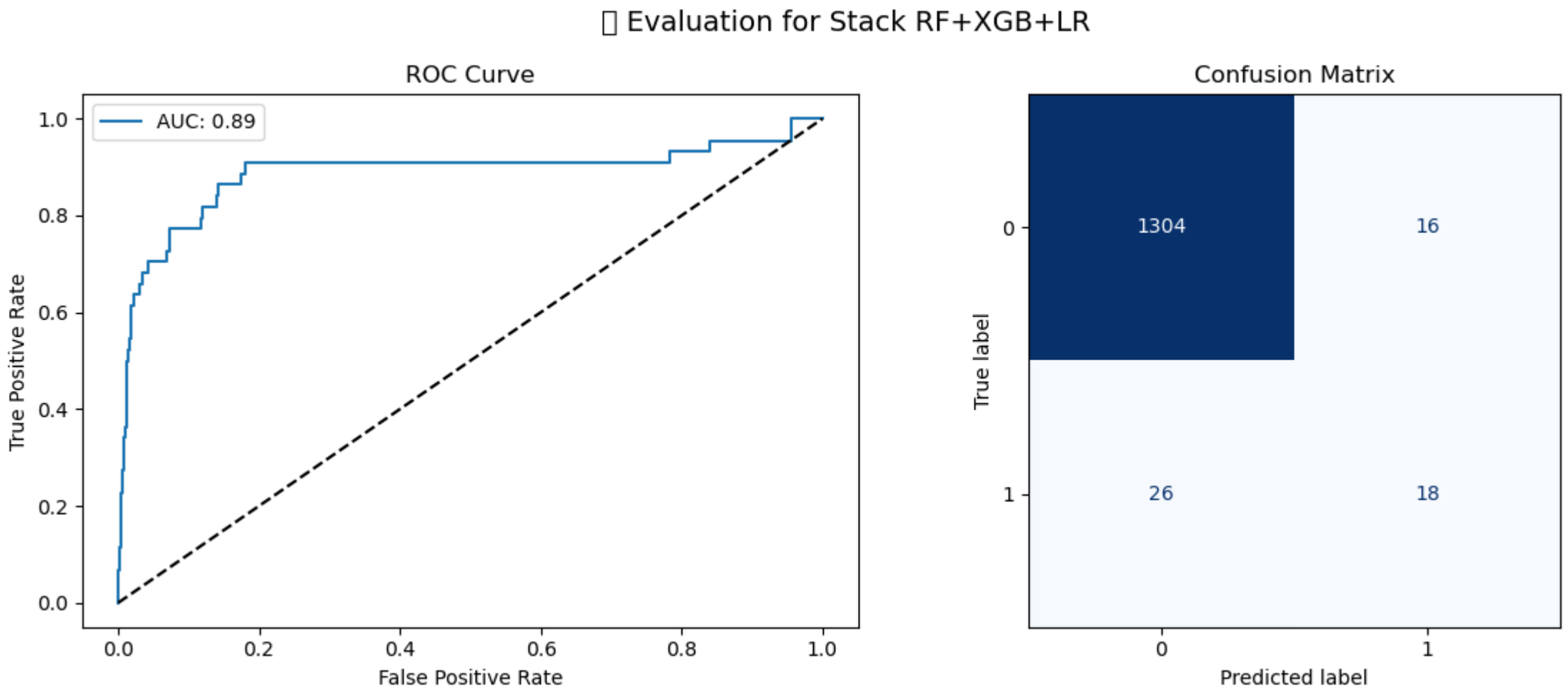

```
Training Logistic Regression...

=== Logistic Regression ===
              precision    recall  f1-score   support

           0     0.9924    0.8864    0.9364      1320
           1     0.1892    0.7955    0.3057        44

    accuracy                         0.8834      1364
   macro avg     0.5908    0.8409    0.6210      1364
weighted avg     0.9665    0.8834    0.9160      1364
```



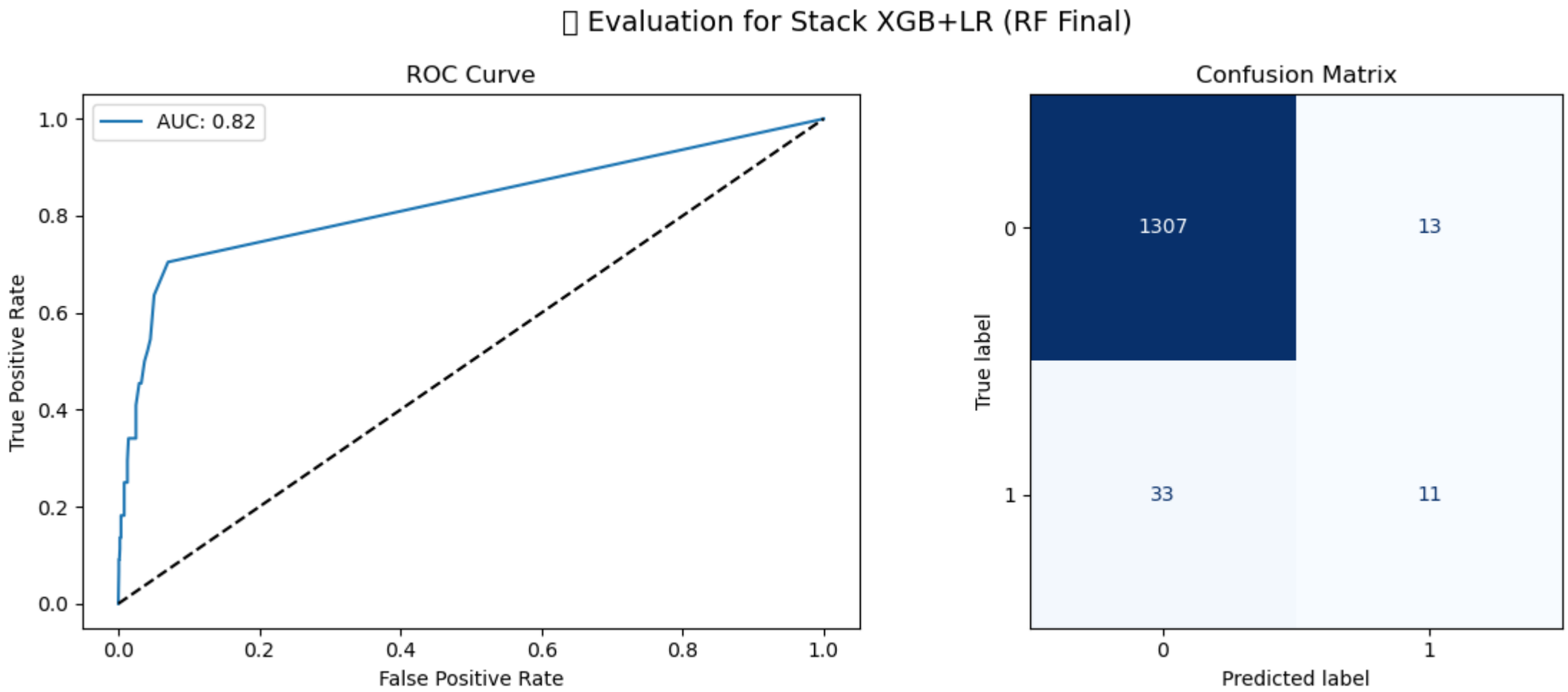

```
Training Stack RF+XGB...

=== Stack RF+XGB ===
              precision    recall  f1-score   support

           0     0.9841    0.9864    0.9852      1320
           1     0.5610    0.5227    0.5412        44

    accuracy                         0.9714      1364
   macro avg     0.7726    0.7545    0.7632      1364
weighted avg     0.9705    0.9714    0.9709      1364
```



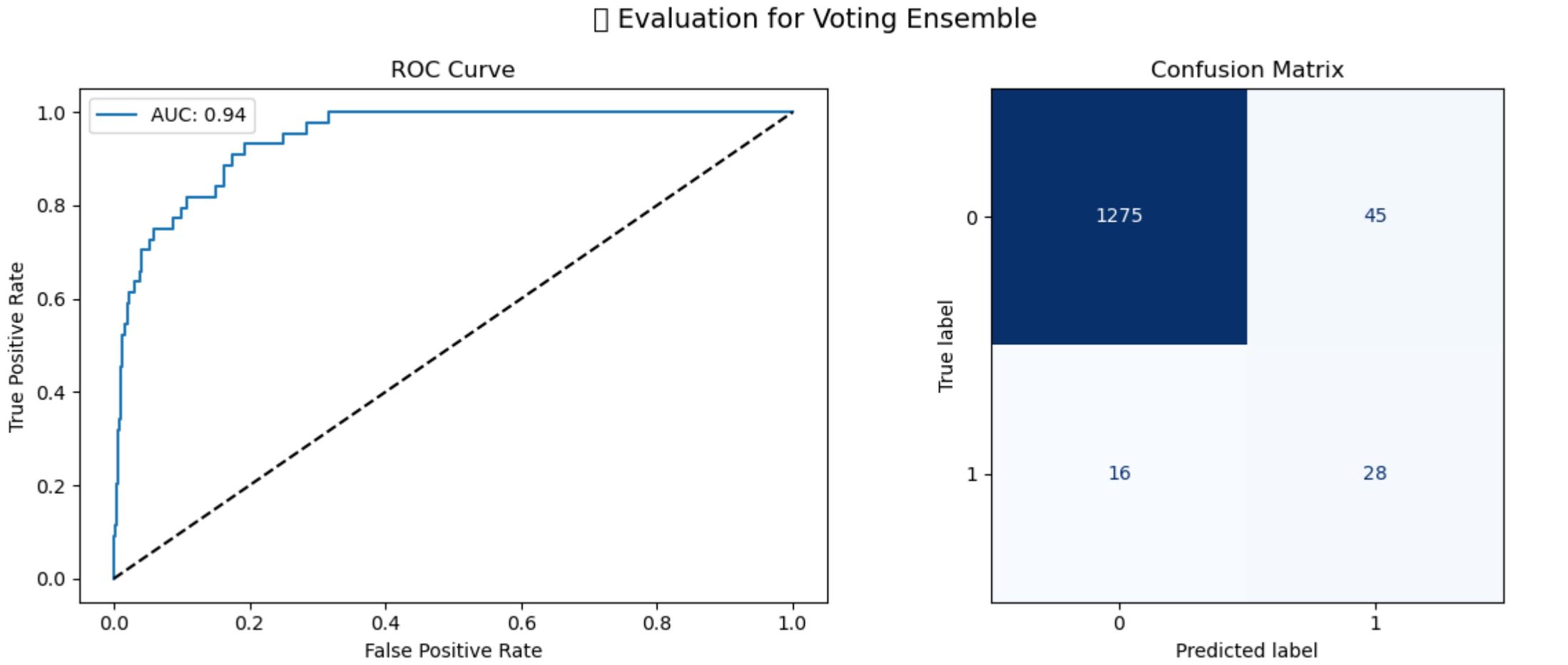

```
Training Stack RF+XGB+LR...

=== Stack RF+XGB+LR ===
              precision    recall  f1-score   support

           0     0.9805    0.9879    0.9842      1320
           1     0.5294    0.4091    0.4615        44

    accuracy                         0.9692      1364
   macro avg     0.7549    0.6985    0.7228      1364
weighted avg     0.9659    0.9692    0.9673      1364
```




```
Training Stack XGB+LR (RF Final)...

=== Stack XGB+LR (RF Final) ===
              precision    recall  f1-score   support

           0     0.9754    0.9902    0.9827      1320
           1     0.4583    0.2500    0.3235        44

    accuracy                         0.9663      1364
   macro avg     0.7169    0.6201    0.6531      1364
weighted avg     0.9587    0.9663    0.9614      1364
```




```
Training Voting Ensemble...

=== Voting Ensemble ===
              precision    recall  f1-score   support

           0     0.9876    0.9659    0.9766      1320
           1     0.3836    0.6364    0.4786        44

    accuracy                         0.9553      1364
   macro avg     0.6856    0.8011    0.7276      1364
weighted avg     0.9681    0.9553    0.9606      1364
```



In [46]:
```python
# Created a DataFrame from the stored model evaluation results
results_df = pd.DataFrame(
    results,
    columns=["Model", "F1 Score", "ROC AUC"]
)

# Sortted models by F1 Score in descending order
results_df = results_df.sort_values(by="F1 Score", ascending=False)

# Printed a header for clarity
print("\n Final Model Comparison:\n")

# Displayed the final comparison table
print(results_df)
```

```
🏁 Final Model Comparison:

                    Model  F1 Score   ROC AUC
3             Stack RF+XGB  0.541176  0.948192
0            Random Forest  0.528302  0.944155
1                  XGBoost  0.505051  0.937534
6          Voting Ensemble  0.478632  0.943905
4          Stack RF+XGB+LR  0.461538  0.886863
5  Stack XGB+LR (RF Final)  0.323529  0.824862
2      Logistic Regression  0.305677  0.915926
```

Final Model Comparison Results

This table summarises the performance of all evaluated models based on **F1 Score** and **ROC AUC**, calculated on the test dataset.

Models are ranked in descending order of **F1 Score**, which is prioritised due to the imbalanced nature of the bankruptcy classification problem and its focus on minority-class detection.

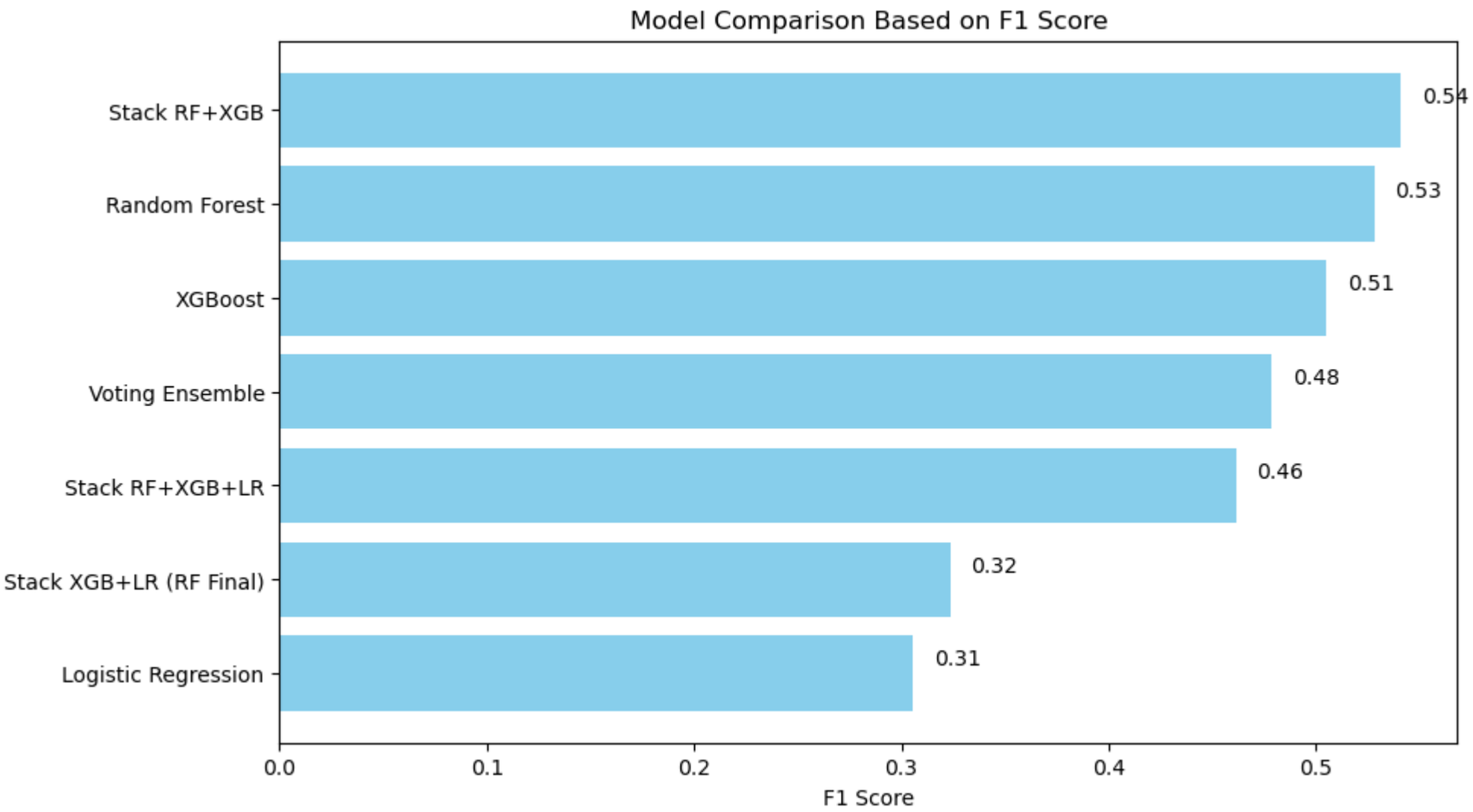
Key observations from the table:

- The **Stack RF+XGB** ensemble achieves the highest F1 Score (≈ 0.54) and the highest ROC AUC (≈ 0.95), indicating the strongest overall balance between precision and recall for bankrupt firms.
- **Random Forest** and **XGBoost** perform competitively as standalone models, with strong ROC AUC values and relatively high F1 Scores.
- The **Voting Ensemble** provides moderate performance, improving upon some individual models but not surpassing the best stacking configuration.
- **Logistic Regression** and the **Stack XGB+LR (RF Final)** model exhibit lower F1 Scores, indicating weaker minority-class classification performance despite acceptable ROC AUC values.

Overall, ensemble-based approaches—particularly stacking—demonstrate improved classification performance compared to individual baseline models.

In [47]:
```
# Created a figure for the horizontal bar chart
plt.figure(figsize=(10, 6))

# Plotted horizontal bars for F1 Scores of each model
bars = plt.barh(
    results_df["Model"],
    results_df["F1 Score"],
    color='skyblue'
)

# Labeled the x-axis and set the plot title
plt.xlabel("F1 Score")
plt.title("Model Comparison Based on F1 Score")

# Inverted y-axis so the best-performing model appears at the top
plt.gca().invert_yaxis()
```

```
# Annotated each bar with its corresponding F1 Score value
for bar in bars:
    plt.text(
        bar.get_width() + 0.01,      # Position text slightly to the right of the bar
        bar.get_y() + 0.25,          # Vertically align text with the bar
        f"{bar.get_width():.2f}",    # Format F1 score to two decimal places
        va='center'
    )

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



Visual Comparison of Model Performance

This cell visualises the performance of all evaluated models using a horizontal bar chart based on their F1 Scores. The chart enables an intuitive comparison of model effectiveness, particularly for minority-class classification.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: