

# ML in Banking

## Contents

- ML in Banking
- Key Performance Metrics in Banking ML
  - 1. Classification Metrics
  - 2. Regression Metrics



## Introduction

The banking sector has always been **data-driven**. From simple ledgers in the 19th century, to **statistical credit scoring in the 1960s**, and finally to **AI-powered predictive systems** in the 21st century.

Today, **Machine Learning (ML)** plays a critical role in:

- Credit scoring
- Fraud detection
- Customer segmentation
- Risk management

## Evolution Timeline of Models in Banking

The development of **predictive models in banking** can be divided into **three main generations**, each shaped by advances in statistics, computing power, and data availability.

Era	Approach	Characteristics & Examples
1960s–1980s	Statistical Models	<div><div>- Linear regression for credit risk.</div><div>- Early <b>credit scoring systems</b> (e.g., <b>FICO Score</b>, introduced 1956).</div><div>- Rule-based underwriting (income-to-debt ratios, collateral value).</div></div>
1990s–2000s	Early ML Models	<div><div>- <b>Logistic Regression</b> widely adopted for loan default prediction.</div><div>- <b>Decision Trees</b> for interpretable rule-based credit approval.</div><div>- <b>Support Vector Machines (SVMs)</b> in fraud detection.</div><div>- Example: <b>SAS Credit Scoring solutions, Fair Isaac models.</b></div></div>
2010s–present	Advanced ML/AI	<div><div>- <b>Random Forests &amp; Gradient Boosting (XGBoost, LightGBM)</b> for credit scoring and churn prediction.</div><div>- <b>Neural Networks</b> for fraud detection (e.g., transaction anomaly detection at Visa &amp; Mastercard).</div><div>- <b>Deep Learning</b> in AML (Anti-Money Laundering) surveillance.</div><div>- Example: <b>Zest AI</b> (credit underwriting), <b>Kabbage</b> (AI-powered small business loans).</div></div>

Each generation introduced **better accuracy, adaptability, and automation**, but also new challenges (e.g., explainability in deep learning).

👉 **Trend:** from simple rule-based systems to **AI-driven predictive platforms** integrating big data, customer behavior, and real-time analytics.

## Traditional vs Predictive ML Models

Feature	Traditional Scoring Models	Predictive ML Models
Data	Limited (few financial ratios, credit score)	Large-scale (transactions, behavior, digital footprints)
Flexibility	Static rules	Dynamic, adapts to new data
Interpretability	High (easy to explain to regulators)	Varies (trees easy, deep learning complex)
Accuracy	Moderate	High (captures complex patterns)
Use Case	Credit approval, basic risk assessment	Fraud detection, churn, segmentation

## Common Model Types in Banking ML

In the banking sector, different predictive models are chosen based on **accuracy**, **interpretability**, and **regulatory requirements**.

### Logistic Regression (1970s–)

- **Description:** A statistical model for binary classification (yes/no outcomes).
- **Pros:** Simple, interpretable, widely accepted by regulators.
- **Cons:** Limited in capturing non-linear patterns.
- **Banking Example:** Credit scoring (probability of default).

## Decision Trees (1980s–)

- **Description:** Rule-based models splitting data by features (if-else logic).
- **Pros:** Easy to visualize and explain → regulator-friendly.
- **Cons:** Prone to overfitting if not pruned.
- **Banking Example:** Loan approval workflows, early fraud detection systems.

## Random Forests (2000s–)

- **Description:** Ensemble of many decision trees (bagging approach).
- **Pros:** Robust, reduces overfitting, handles noisy data well.
- **Cons:** Less interpretable than single trees.
- **Banking Example:** Customer churn prediction, credit risk modeling.

## Gradient Boosting (2010s–)

- **Description:** Ensemble method that builds trees sequentially, focusing on previous errors (e.g., XGBoost, LightGBM, CatBoost).
  - **Pros:** Very high accuracy, works well with imbalanced data.
  - **Cons:** Computationally expensive, harder to explain.
  - **Banking Example:** Fraud detection, anti-money laundering (AML) monitoring.
-

# Comparison Table

Model Type	Pros	Cons	Use Case
<b>Logistic Regression</b>	Simple, interpretable, regulator-friendly	Limited to linear patterns	Credit scoring
<b>Decision Trees</b>	Easy to explain, visualizable	Overfitting risk	Loan approval
<b>Random Forests</b>	Robust, accurate, less variance	Less interpretable	Churn prediction
<b>Gradient Boosting</b>	High accuracy, handles imbalance	Complex, slow training	Fraud detection

👉 Over time, the trend moves from **simplicity & interpretability** → **complexity & accuracy**, raising challenges of **Explainable AI (XAI)** in banking.

In banking, evaluating ML models requires metrics that balance:

- **Accuracy** → are predictions correct?
- **Interpretability** → can regulators and analysts understand them?
- **Financial Impact** → what is the cost of errors?

Metrics are grouped into three categories:

1. **Classification Metrics** → for categorical outcomes (default vs non-default).
2. **Regression Metrics** → for continuous outcomes (loss amount, exposure).
3. **Feature Importance Metrics** → to explain *why* the model made decisions.

When building predictive models in banking, we often classify outcomes such as:

- **Default (loan not repaid)** vs **Non-Default (loan repaid)**
- **Fraud (illegitimate transaction)** vs **Genuine (legitimate transaction)**

## The meaning of *Positive*:

- In machine learning, **Positive** = the event we want to detect or verify.
- Example: if the goal is to detect loan defaults → **Default = Positive**.

- If the goal is to detect fraud → **Fraud = Positive**.

# Confusion Matrix

A confusion matrix shows the relationship between **predictions** and **actual outcomes**:

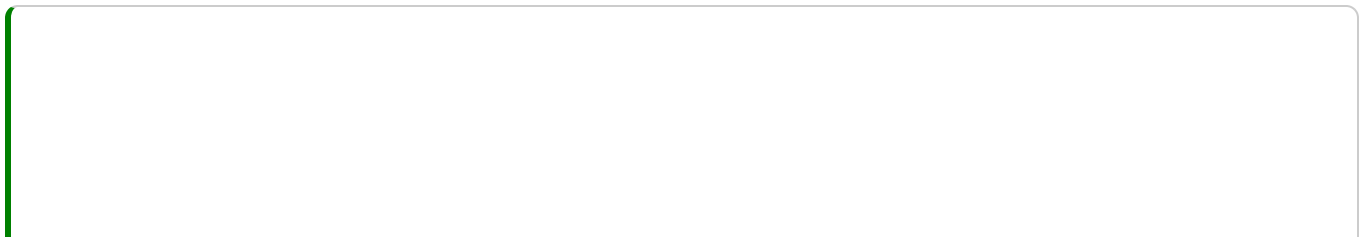
	Predicted Positive	Predicted Negative
Actual Positive	TP (True Positive) → correctly detected default	FN (False Negative ⚠️ costly) → missed default
Actual Negative	FP (False Positive) → good loan wrongly flagged	TN (True Negative) → correctly identified non-default

## Interpretation:

- **True Positive (TP):** The model predicts *default* and the customer actually defaults → ✅ Correct detection.
- **True Negative (TN):** The model predicts *non-default* and the customer actually repays → ✅ Correct rejection.
- **False Positive (FP):** The model predicts *default* but the customer repays → ❌ Lost opportunity.
- **False Negative (FN):** The model predicts *non-default* but the customer defaults → ⚠️ Very costly mistake.

## Visual Representation

Below is a graphical representation of the confusion matrix:



```

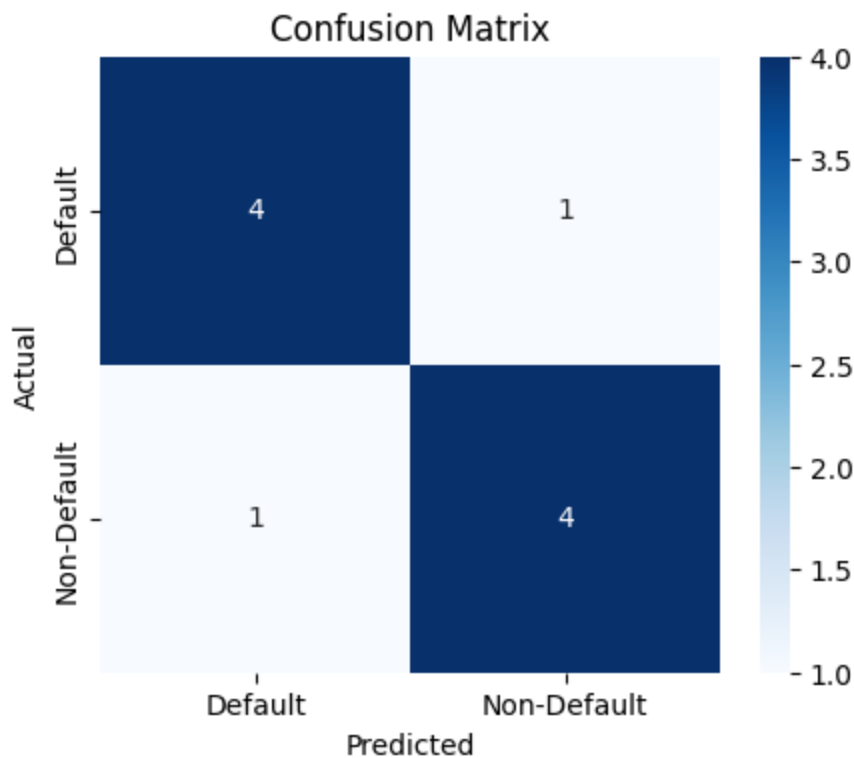
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

y_true = [1, 0, 1, 1, 0, 1, 0, 0, 1, 0] # 1 = Default, 0 = Non-default
y_pred = [1, 0, 1, 0, 0, 1, 0, 1, 1, 0]

# Compute confusion matrix
cm = confusion_matrix(y_true, y_pred)
labels = [["TP", "FN"], ["FP", "TN"]]

# Plot heatmap
plt.figure(figsize=(5,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Default', 'Non-Default'],
            yticklabels=['Default', 'Non-Default'], title="Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

```



## Core Metrics

These metrics evaluate how well a classification model performs.

# Accuracy

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

👉 Proportion of **all correct predictions**. ⚠ Misleading when data is imbalanced (e.g., fraud = only 1% of cases → a model predicting *all non-fraud* has 99% accuracy but is useless).

---

# Precision

$$Precision = \frac{TP}{TP + FP}$$

👉 Out of all **predicted defaulters**, how many were **truly defaulting**?

- Example: If a bank predicts 100 customers as high-risk and 80 truly default, **Precision = 80%**.
- 

# Recall (Sensitivity)

$$Recall = \frac{TP}{TP + FN}$$

👉 Out of all **actual defaulters**, how many did we **catch**?

- Example: If 100 customers actually defaulted, and the model detected 70, then **Recall = 70%**.
- 

# F1 Score

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$



👉 Harmonic mean of Precision & Recall. ✅ Best for fraud detection and credit scoring, where both **false alarms (FP)** and **missed cases (FN)** are costly.

```
from sklearn.metrics import classification_report

print("Classification Report:\n")
print(classification_report(y_true, y_pred))
```

Classification Report:

	precision	recall	f1-score	support
0	0.80	0.80	0.80	5
1	0.80	0.80	0.80	5
accuracy			0.80	10
macro avg	0.80	0.80	0.80	10
weighted avg	0.80	0.80	0.80	10

## Ranking Metrics

Ranking metrics are crucial in **credit scoring and fraud detection**, where the goal is to **rank customers by risk** rather than just classify them.

## AUC (Area Under ROC Curve)

- Measures the probability that the model ranks a **random default higher** than a **random non-default**.
- Range:  $0.5 \leq AUC \leq 1$
- **0.5** → random guessing (no discrimination).
- **1.0** → perfect separation.
- ✅ Standard benchmark in **credit scoring models**.

# GINI Coefficient

- Directly related to AUC:  $GINI = 2 \times AUC - 1$
- Intuition: higher GINI means the model better separates **good vs risky customers**.
- Widely used in **banking regulations and credit risk management**.

## Example in Banking

- If AUC = 0.85 → the model has an **85% chance** to rank a defaulted loan higher than a performing loan.
- Corresponding GINI = 0.70 → considered **strong discriminatory power** in credit risk.

```
from sklearn.metrics import roc_auc_score, roc_curve
import matplotlib.pyplot as plt

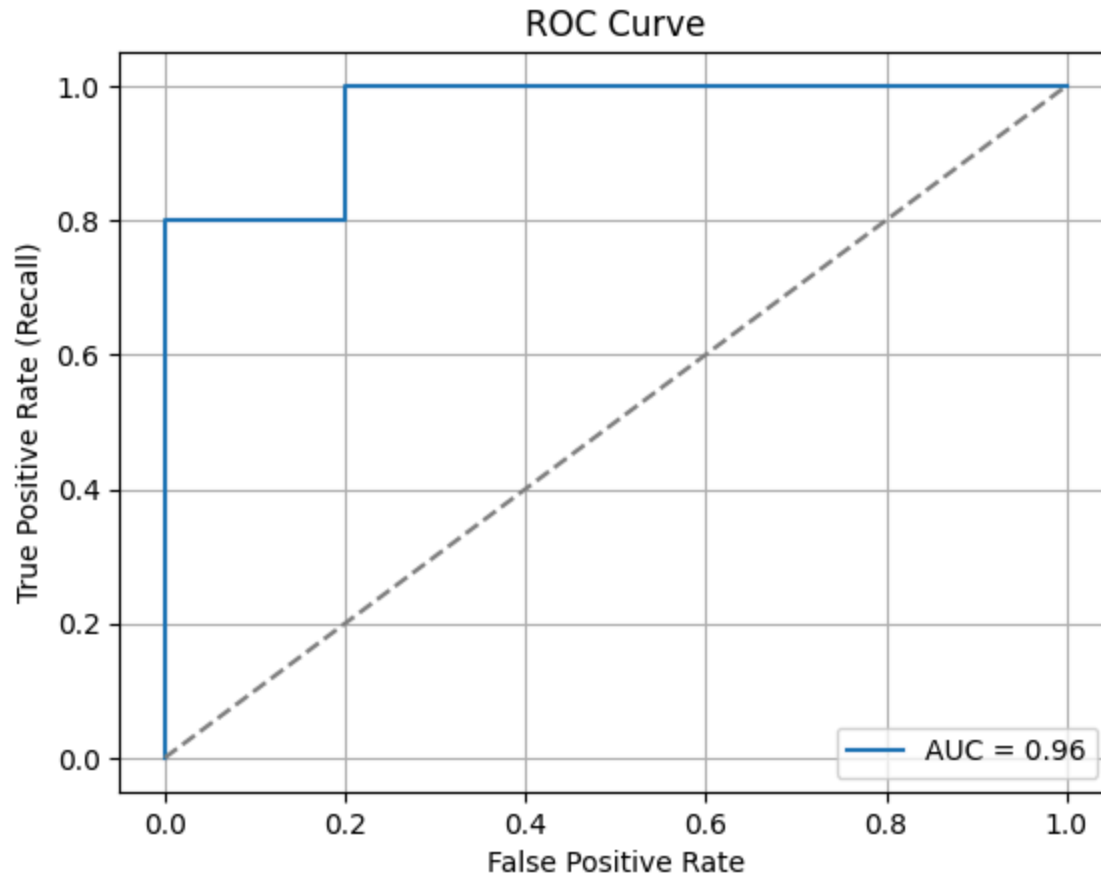
y_prob = [0.9, 0.1, 0.8, 0.4, 0.2, 0.95, 0.3, 0.7, 0.85, 0.05]

# Compute AUC and GINI
auc = roc_auc_score(y_true, y_prob)
gini = 2 * auc - 1

print(f"AUC: {auc:.3f}")
print(f"GINI: {gini:.3f}")
```

AUC: 0.960  
GINI: 0.920

```
# Plot ROC Curve
fpr, tpr, thresholds = roc_curve(y_true, y_prob)
plt.plot(fpr, tpr, label=f'AUC = {auc:.2f}')
plt.plot([0,1], [0,1], linestyle='--', color='gray') # Random baseline
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate (Recall)')
plt.title('ROC Curve')
plt.legend()
plt.grid(True)
plt.show()
```



Regression metrics are used when predicting **continuous values** such as:

- Loss Given Default (LGD)
- Customer Lifetime Value (CLV)
- Exposure at Default (EAD)


## Mean Absolute Error (MAE)

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

- Measures the **average absolute difference** between predicted and actual values.
- ☒ Easy to interpret → e.g., if MAE = 500, predictions are off by \$500 on average.

## Mean Squared Error (MSE)


$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

- Penalizes **larger errors** more heavily (because of squaring).
-  Useful in **risk forecasting**, where big mistakes are costly.
- Example: predicting losses in millions → large deviations get penalized strongly.

## R<sup>2</sup> (Coefficient of Determination)

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

Where:

- $SS_{res} = \sum (y_i - \hat{y}_i)^2 \rightarrow$  residual sum of squares.
- $SS_{tot} = \sum (y_i - \bar{y})^2 \rightarrow$  total variance in data.
- Interprets as: **% of variance explained by the model**.
-  If  $R^2 = 0.85 \rightarrow$  the model explains 85% of the variability in loan losses.

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np
```

```
# Example: Actual vs Predicted loan losses ($)
y_true = np.array([1000, 2000, 3000, 4000, 5000])
y_pred = np.array([1100, 2100, 3100, 3900, 4800])
```

```
mae = mean_absolute_error(y_true, y_pred)
mse = mean_squared_error(y_true, y_pred)
r2 = r2_score(y_true, y_pred)
```

```
print(f"MAE: {mae:.2f}")
print(f"MSE: {mse:.2f}")
print(f"R²: {r2:.3f}")
```

```
MAE: 120.00
MSE: 16000.00
R²: 0.992
```

# When to Use Which Metric?

Scenario	Best Metrics
Loan approval (classification)	AUC, GINI, Confusion Matrix
Fraud detection	Recall, F1 Score ( ⚠️ missing fraud is costly)
Loss forecasting	MAE, MSE, $R^2$
Regulatory audit / Explainability	Feature Importance, Coefficients, Permutation

👉 Always tie metric choice to **financial impact**:

- False Negatives (missed frauds) → direct losses.
- False Positives (rejecting good customers) → lost revenue.
- Large regression errors → mispricing loans and risk.

## Use Cases in Banking

- **Loan Default Prediction:** Estimate the probability of a customer defaulting on a loan.
- **Transaction Classification:** Detect fraudulent or unusual transaction patterns.
- **Customer Segmentation:** Group clients for targeted marketing, cross-selling, or risk-based pricing.
- **Credit Risk Modeling:** Assess overall portfolio health.
- **Churn Prediction:** Identify customers at risk of leaving the bank.

## Example: Loan Default Prediction

We will:

1. Load a real banking dataset ( `loan_predication.csv` ).
2. Train **Logistic Regression, Decision Tree, Random Forest**.
3. Evaluate using **AUC, GINI, Confusion Matrix, F1 Score, and ROC Curve**.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score, roc_curve, confusion_matrix, f1_score
from sklearn.impute import SimpleImputer

# Load dataset
df = pd.read_csv("datasets/loan_predication/loan_predication.csv")

# Preview dataset
df.head()
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantInr
0	LP001002	Male	No	0	Graduate	No	
1	LP001003	Male	Yes	1	Graduate	No	
2	LP001005	Male	Yes	0	Graduate	Yes	
3	LP001006	Male	Yes	0	Not Graduate	No	
4	LP001008	Male	No	0	Graduate	No	

```

# Check missing values in dataset
print("Missing values per column:\n")
print(df.isnull().sum())

# Scatter visualization of missing values
plt.figure(figsize=(10,6))

# Convert dataframe to a mask (1 = missing, 0 = not missing)
missing_mask = df.isnull()

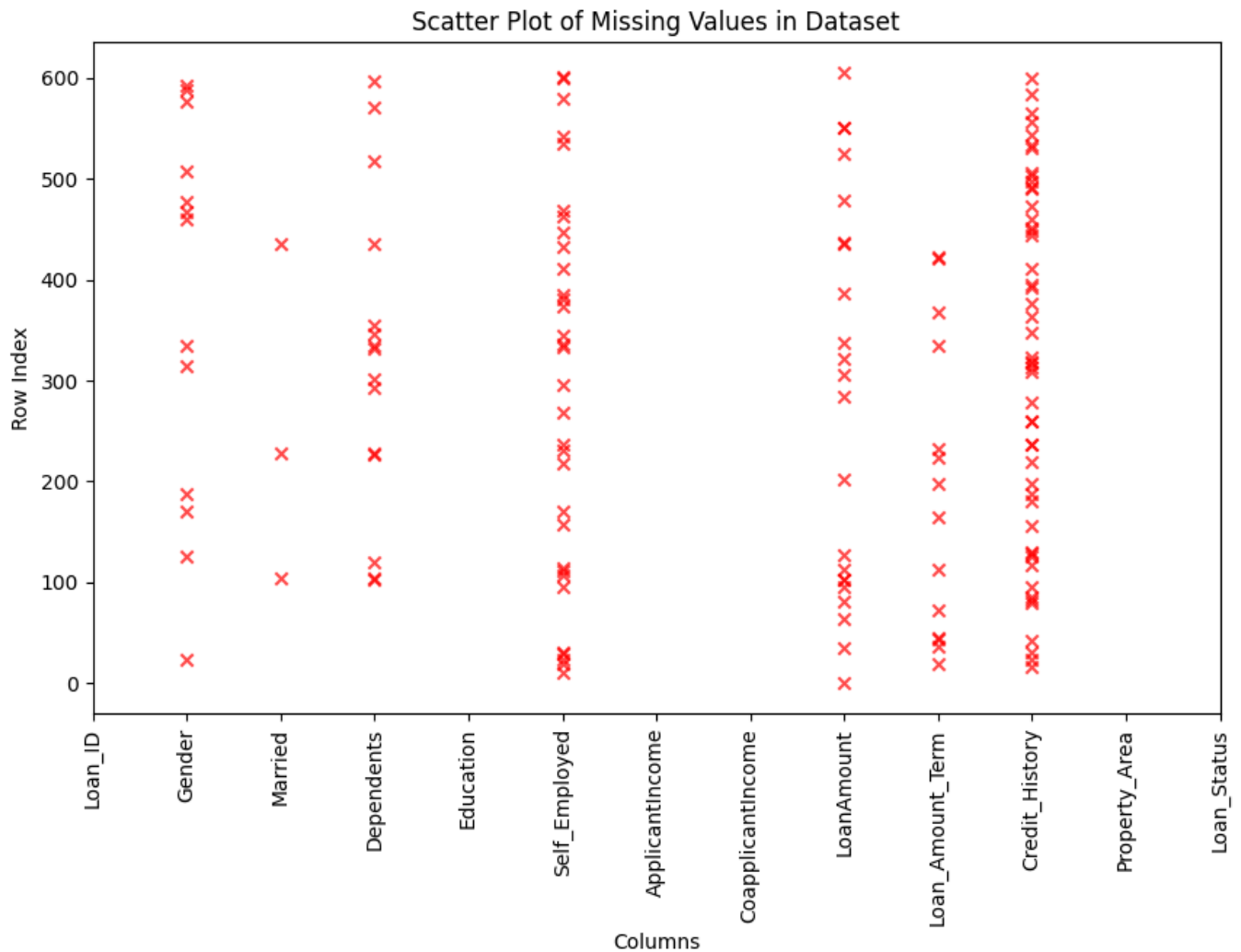
# Loop through columns and plot missing points
for i, col in enumerate(df.columns):
    # Get the indices of missing values for this column
    missing_indices = missing_mask[missing_mask[col]].index
    plt.scatter([i]*len(missing_indices), missing_indices, marker='x', color='red', a

plt.xticks(range(len(df.columns)), df.columns, rotation=90)
plt.ylabel("Row Index")
plt.xlabel("Columns")
plt.title("Scatter Plot of Missing Values in Dataset")
plt.show()

```

Missing values per column:

Loan_ID	0
Gender	13
Married	3
Dependents	15
Education	0
Self_Employed	32
ApplicantIncome	0
CoapplicantIncome	0
LoanAmount	22
Loan_Amount_Term	14
Credit_History	50
Property_Area	0
Loan_Status	0
dtype: int64	



```
# Separate target before encoding
y = df['Loan_Status'].map({'Y': 1, 'N': 0}) # Encode target: Y=1, N=0
X = df.drop(columns=['Loan_Status'])

# Preprocessing: Encode categorical features
X = pd.get_dummies(X, drop_first=True)

# Handle missing values (fill with median for numeric, mode for categorical)
X = X.fillna(X.median(numeric_only=True))
X = X.fillna(X.mode().iloc[0])

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
X_train.shape, X_test.shape
```

```
((429, 627), (185, 627))
```



```

from sklearn.metrics import classification_report

def evaluate_model(model, X_train, X_test, y_train, y_test, model_name):
    model.fit(X_train, y_train)
    preds = model.predict(X_test)
    probs = model.predict_proba(X_test)[ :,1]

    # Metrics
    auc = roc_auc_score(y_test, probs)
    gini = 2 * auc - 1
    cm = confusion_matrix(y_test, preds)

    print(f"==== {model_name} =====")
    print(f"AUC: {auc:.3f}")
    print(f"GINI: {gini:.3f}")
    print("\nClassification Report:")
    print(classification_report(y_test, preds, digits=3))
    print("Confusion Matrix:\n", cm)

    # Confusion Matrix Plot
    plt.figure(figsize=(5,4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=['Predicted Non-Default', 'Predicted Default'],
                yticklabels=['Actual Non-Default', 'Actual Default'])
    plt.title(f"Confusion Matrix - {model_name}")
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()

    # ROC Curve Plot
    fpr, tpr, _ = roc_curve(y_test, probs)
    plt.figure(figsize=(6,5))
    plt.plot(fpr, tpr, label=f'{model_name} (AUC={auc:.3f})')
    plt.plot([0,1],[0,1], '--', color='gray')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f"ROC Curve - {model_name}")
    plt.legend(loc='lower right')
    plt.show()

    return

# Logistic Regression
evaluate_model(LogisticRegression(max_iter=10000, solver='saga', class_weight='balance

```

==== Logistic Regression =====

AUC: 0.490

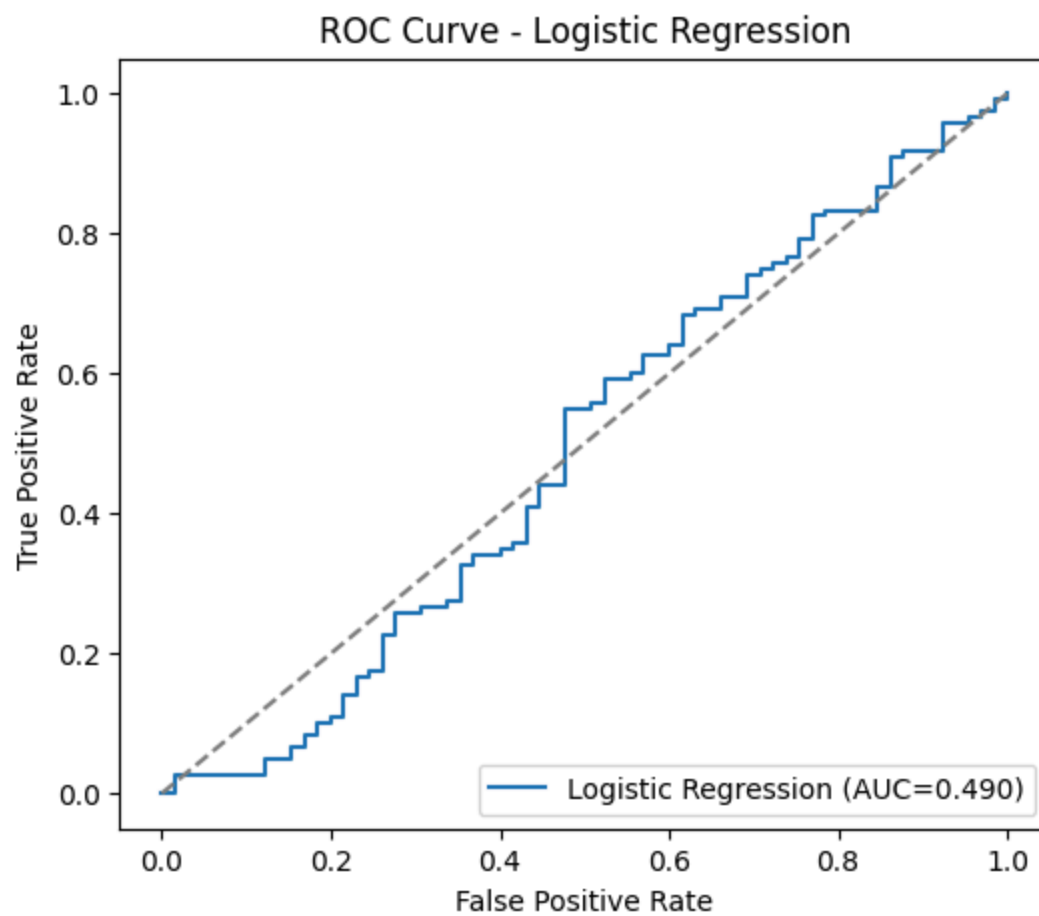
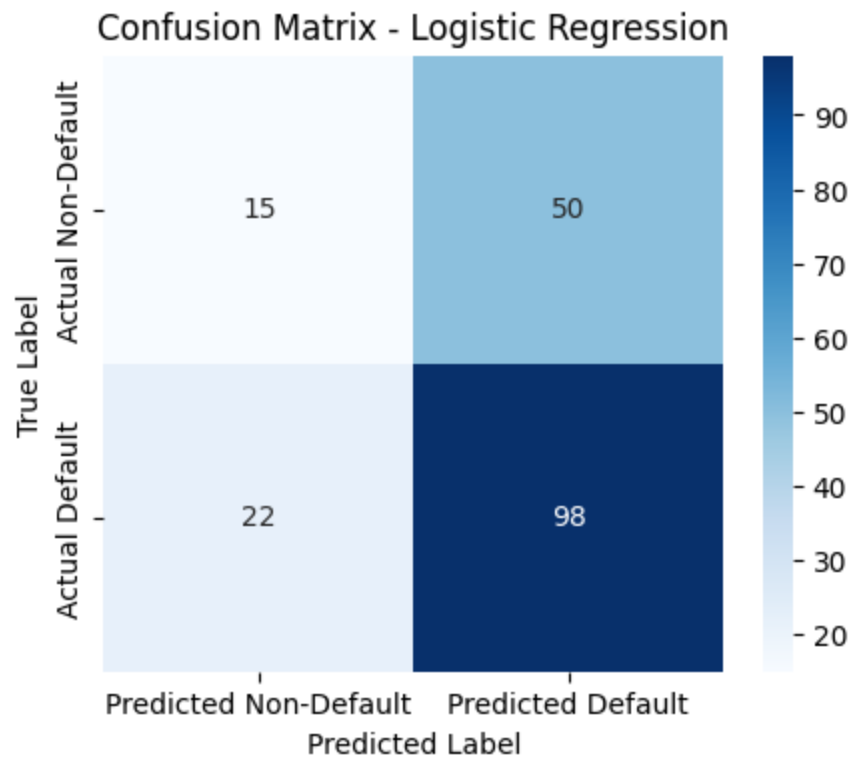
GINI: -0.021

Classification Report:

	precision	recall	f1-score	support
0	0.405	0.231	0.294	65
1	0.662	0.817	0.731	120
accuracy			0.611	185
macro avg	0.534	0.524	0.513	185
weighted avg	0.572	0.611	0.578	185

Confusion Matrix:

```
[[15 50]
 [22 98]]
```



```
# Decision Tree
evaluate_model(DecisionTreeClassifier(max_depth=5, random_state=42), X_train, X_test,
```

===== Decision Tree =====

AUC: 0.735

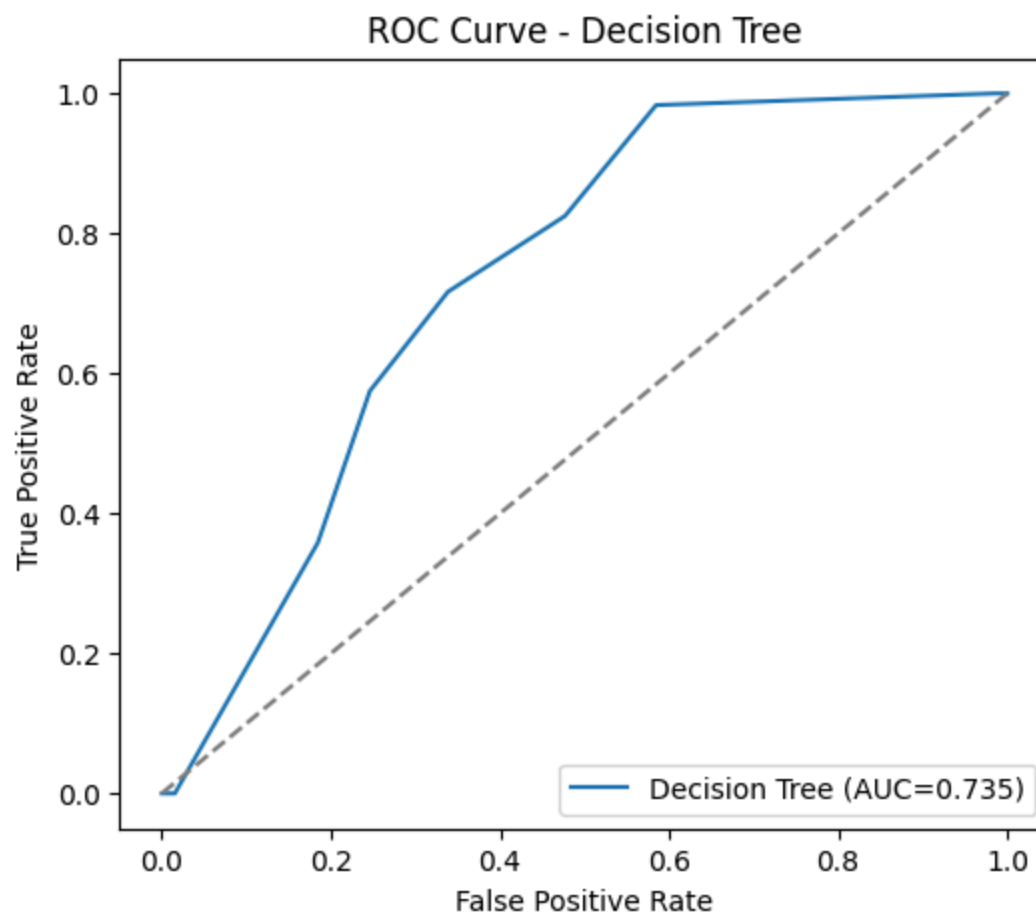
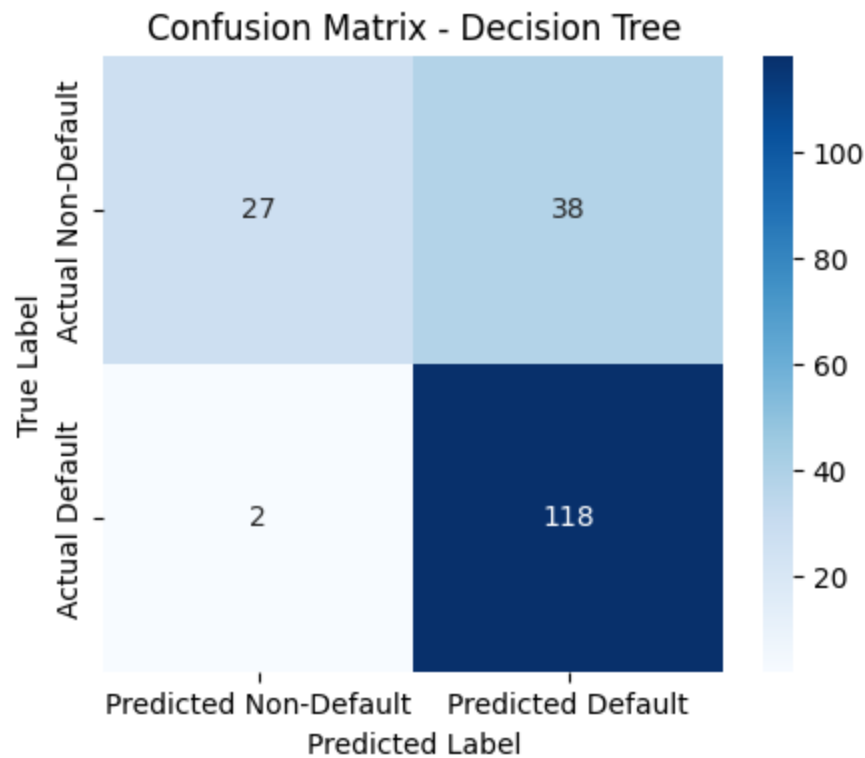
GINI: 0.470

Classification Report:

	precision	recall	f1-score	support
0	0.931	0.415	0.574	65
1	0.756	0.983	0.855	120
accuracy			0.784	185
macro avg	0.844	0.699	0.715	185
weighted avg	0.818	0.784	0.756	185

Confusion Matrix:

```
[[ 27  38]
 [   2 118]]
```



```
# Random Forest  
evaluate_model(RandomForestClassifier(n_estimators=100, random_state=42), X_train, X_test)
```

==== Random Forest ====

AUC: 0.758

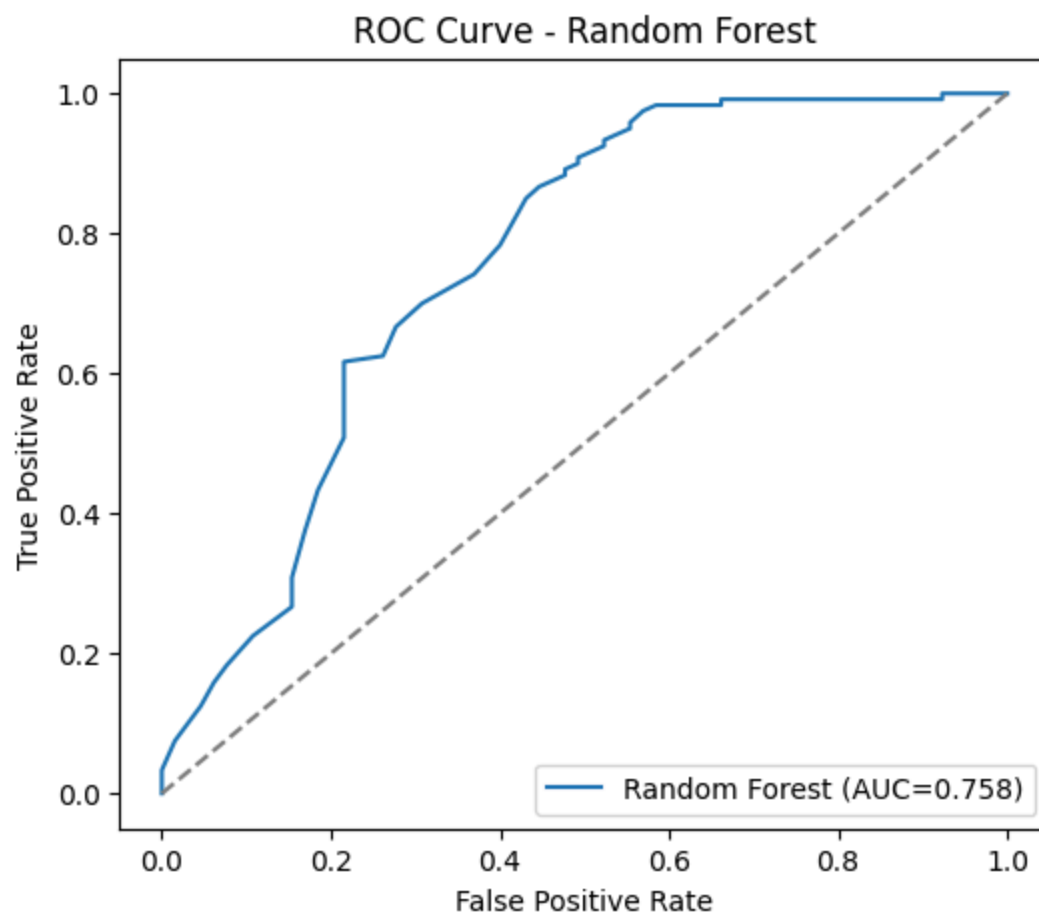
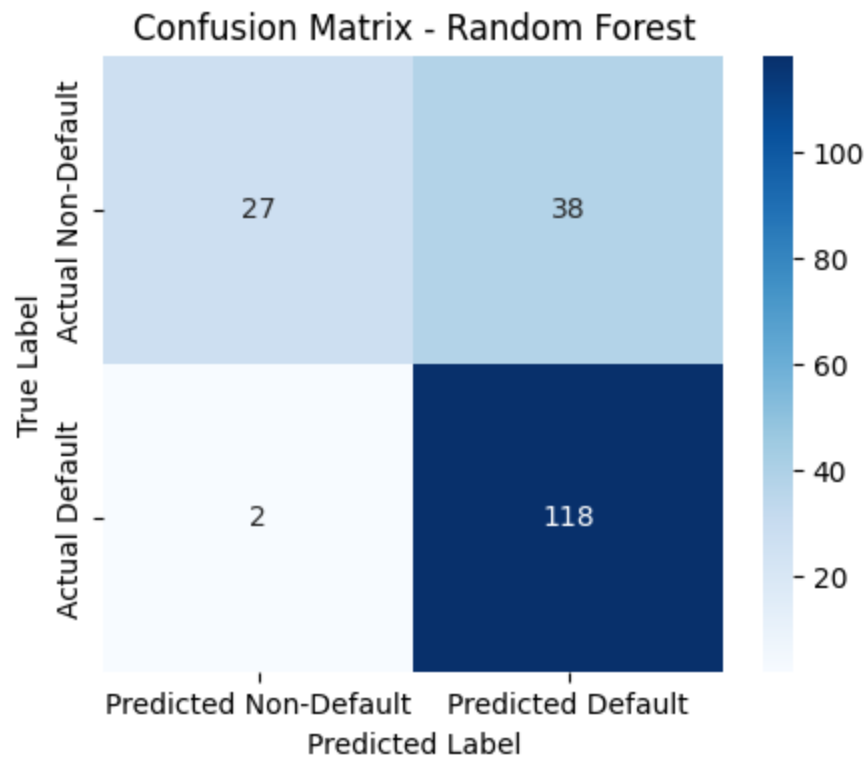
GINI: 0.516

Classification Report:

	precision	recall	f1-score	support
0	0.931	0.415	0.574	65
1	0.756	0.983	0.855	120
accuracy			0.784	185
macro avg	0.844	0.699	0.715	185
weighted avg	0.818	0.784	0.756	185

Confusion Matrix:

```
[[ 27  38]
 [   2 118]]
```



# Interpretation of Results

## Model Comparison

- **Logistic Regression** → baseline, interpretable, good for regulatory reporting.
- **Decision Tree** → captures non-linear relationships, easy to visualize but may overfit.
- **Random Forest** → ensemble of trees, more robust and usually best performance.

## Additional Banking Metrics

- **AUC (Area Under Curve)**: Measures ability of the model to rank defaulters higher than non-defaulters.
- **GINI Coefficient**:  $GINI = 2 \times AUC - 1$  Widely used in credit risk → higher GINI means stronger discriminatory power.
- **Accuracy Ratio (AR)**:  $AR = GINI \times 100\%$  Expresses the model's improvement over random guessing in percentage terms.

```
# Train Random Forest and keep the model instance
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Example: Compute AUC, GINI, and AR for Random Forest
rf_probs = rf_model.predict_proba(X_test)[: ,1]
auc = roc_auc_score(y_test, rf_probs)
gini = 2 * auc - 1
ar = gini * 100

print("==== Random Forest Evaluation =====")
print(f"AUC: {auc:.3f}")
print(f"GINI: {gini:.3f}")
print(f"Accuracy Ratio (AR): {ar:.2f}%")
```

```
==== Random Forest Evaluation =====
AUC: 0.758
GINI: 0.516
Accuracy Ratio (AR): 51.56%
```