BNB_BANK_001

In [3]:

```
import pandas as pd
import numpy as np
import warnings
import joblib

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score, accuracy_score
from sklearn.preprocessing import StandardScaler, OneHotEncoder, FunctionTransformer #LabelEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.feature_extraction.text import TfidfVectorizer
#from xgboost import XGBClassifier

from utils import flatten_array
from custom_models import XGBWithLE
from preprocessing_predict import preprocess
#from sklearn.base import BaseEstimator, ClassifierMixin

# Suppress warnings for cleaner output
warnings.filterwarnings('ignore')
```

In [4]:

```
# --- Load and prepare data ---
print("\n📂 Loading and preparing data...")
df = pd.read_csv('transactions_with_labels_cleaned.csv')

# Ensure correct data types
df['Description'] = df['Description'].astype(str)
df['Amount'] = pd.to_numeric(df['Amount'], errors='coerce')

# Drop rows missing required target or numeric values
df = df.dropna(subset=['Label', 'Amount'])
```

```
📂 Loading and preparing data...
```

In [9]:

```
df.head(1)
```

Out[9]:

	Description	Credit or Debit	Amount	Label	day_of_week_numeric	month_numeric
0	POS DEB 1102 12/29/23 58047332 NNT BURLINGTON ...	Debit	13.7	Expense: General business expenses	4	12

In [6]:

```python
# --- Separate the target variable ---
y = df['Label']
X = df[['Description', 'Credit or Debit', 'Amount', 'day_of_week_numeric', 'month_numeric']]

# --- Train/test split ---
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# === Transformers ===
# --- Preprocessing Pipelines ---
text_transformer = Pipeline([
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
    ('flatten', FunctionTransformer(flatten_array, validate=False)),
    ('tfidf', TfidfVectorizer(max_features=100, lowercase=True, stop_words='english'))
])

numeric_transformer = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

preprocessor = ColumnTransformer(
    transformers=[
        ('text', text_transformer, ['Description']),
        ('num', numeric_transformer, ['Amount', 'day_of_week_numeric', 'month_numeric']),
        ('cat', categorical_transformer, ['Credit or Debit'])
    ]
)

# --- Models to evaluate ---
models = {
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42, class_weight='balanced'),
    'Logistic Regression': LogisticRegression(random_state=42, max_iter=1000, class_weight='balanced'),
    'XGBoost': XGBWithLE(random_state=42, eval_metric='mlogloss', scale_pos_weight=1)
}

# --- Training & Evaluation ---
results = []

for name, model in models.items():
    print(f"\n🚀 Training {name}...")
    try:
        pipeline = Pipeline([
            ('preprocessor', preprocessor),
            ('classifier', model)
        ])

        pipeline.fit(X_train, y_train)
        y_pred = pipeline.predict(X_test)

        f1 = f1_score(y_test, y_pred, average='weighted')
        acc = accuracy_score(y_test, y_pred)
        results.append({
            'Model': name,
            'F1 Score': f1,
            'Accuracy': acc,
            'Pipeline': pipeline
        })

        print(f"✅ {name} trained successfully – F1 Score: {f1:.3f}, Accuracy: {acc:.3f}")

    except Exception as e:
        print(f"❌ Error training {name}: {e}")

# --- Display Final Results ---
if results:
    print("\n📊 === Final Model Comparison ===")
    results_df = pd.DataFrame(results).drop(columns=['Pipeline']).sort_values(by='F1 Score', ascending=False)
    print(results_df.to_string(index=False))

    best_model_info = max(results, key=lambda x: x['F1 Score'])
    print(f"\n🏆 Best Model: {best_model_info['Model']} (F1 Score: {best_model_info['F1 Score']:.3f})")
    print(f"🏆 Best Model: {best_model_info['Model']} (Accuracy: {best_model_info['Accuracy']:.3f})")
    best_pipeline = best_model_info['Pipeline']
else:
    print("\n⚠️ No models were successfully trained.")
```

```
🚀 Training Random Forest...
✅ Random Forest trained successfully – F1 Score: 0.668, Accuracy: 0.708

🚀 Training Logistic Regression...
✅ Logistic Regression trained successfully – F1 Score: 0.604, Accuracy: 0.562

🚀 Training XGBoost...
✅ XGBoost trained successfully – F1 Score: 0.763, Accuracy: 0.787

📊 === Final Model Comparison ===
              Model  F1 Score  Accuracy
            XGBoost  0.763198  0.786517
      Random Forest  0.667518  0.707865
Logistic Regression  0.604333  0.561798

🏆 Best Model: XGBoost (F1 Score: 0.763)
🏆 Best Model: XGBoost (Accuracy: 0.787)
```

In [7]:

```
joblib.dump(best_pipeline, "best_model_pipeline.pkl")
print(f"✅ Saved best model ({best_model_info['Model']}) to best_model_pipeline.pkl")
```

```
✅ Saved best model (XGBoost) to best_model_pipeline.pkl
```

In [ ]:

In [ ]: