

**Department of Artificial Intelligence and Machine Learning**

**10212AM228**

**Block Chain Technology**

**Project Title:**

**AI-Powered Detection of Fraudulent Cryptocurrency Transactions**

**CO-ORDINATER:**

**Dr.S. ALEX DAVID**

**Team Members: VTU NO:**

BUDDAREDDY GARI BHAVANA VTU28394

KOTA NAGA SANKAR BALAJI VTU28358

SHAIK MOHAMMED ABBAS VTU28784

**Aim:**  
The project aims to build an AI-based model to detect fraudulent cryptocurrency transactions using the Elliptic dataset. By applying machine learning techniques, the system classifies transactions as licit or illicit, supporting blockchain security and anti-money laundering efforts.

**Algorithm**

1. Data Loading & Preprocessing
   * Load the Elliptic dataset (features, classes, edgelist).
   * Merge feature and class labels, removing unknown transactions.
   * Normalize numerical features and handle missing values.
   * Split data into training and testing sets using **time-based partitioning** (train: timesteps 0–33, test: timesteps 34–49).
2. Class Imbalance Handling
   * Apply **SMOTE (Synthetic Minority Oversampling Technique)** to balance licit and illicit transaction samples in the training set.
3. **Model Training**
   * Train **Random Forest Classifier** as a baseline ensemble model.
   * Train **XGBoost Classifier** for improved performance on tabular features.
4. Prediction & Evaluation
   * Predict on the test set.
   * Evaluate using metrics: Accuracy, Precision, Recall, F1-Score, ROC-AUC.
   * Compare performance of models.
5. Feature Importance & Insights
   * Extract top influential features using Random Forest feature importance.
   * Visualize ROC curves for model comparison.

**Methodology (Step-by-Step)**

1. Data Collection
   * Obtain the Elliptic dataset from Kaggle.
   * Dataset includes:
     + elliptic\_txs\_features.csv (transaction features)
     + elliptic\_txs\_classes.csv (labels: licit/illicit/unknown)
     + elliptic\_txs\_edgelist.csv (transaction graph relationships).
2. DataPreprocessing
   * Merge feature and class datasets by transaction ID.
   * Remove “unknown” class entries.
   * Map labels: licit → 0, illicit → 1.
   * Handle missing values (imputation with zero or mean).
   * Standardize features using StandardScaler.
3. ExploratoryDataAnalysis **(**EDA**)**
   * Analyze class distribution (majority = licit, minority = illicit).
   * Visualize imbalance using bar plots.
   * Explore feature distributions and time-step patterns.
4. Train**/**TestSplitting
   * Use time-based split:
     + Training set → timesteps 0–33
     + Testing set → timesteps 34–49
   * This simulates real-world detection of future fraud.
5. ImbalanceHandling
   * Apply **SMOTE** on training data to balance positive (illicit) and negative (licit) samples.
6. Model Development
   * **Random Forest**: Trained with 200 trees, default hyperparameters.
   * **XGBoost**: Gradient boosting model with tuned parameters for high ROC-AUC.
7. Model Evaluation
   * Metrics: Accuracy, Precision, Recall, F1-Score, ROC-AUC.
   * Generate confusion matrices.
   * Plot ROC curves for Random Forest vs XGBoost.
8. Result Analysis
   * Compare models’ effectiveness in fraud detection.
   * Identify trade-offs:
     + RF: higher recall (better fraud detection).
     + XGBoost: higher precision (fewer false positives).
9. Conclusion & Next Steps
   * AI models can effectively detect fraudulent transactions in cryptocurrency networks.
   * Graph Neural Networks (GCNs, GATs) could further improve accuracy by exploiting transaction graph topology.

Using:

Platform: Google Colab

Language: Python

**Program:**

#Cell 1 — Install / setup (run first)

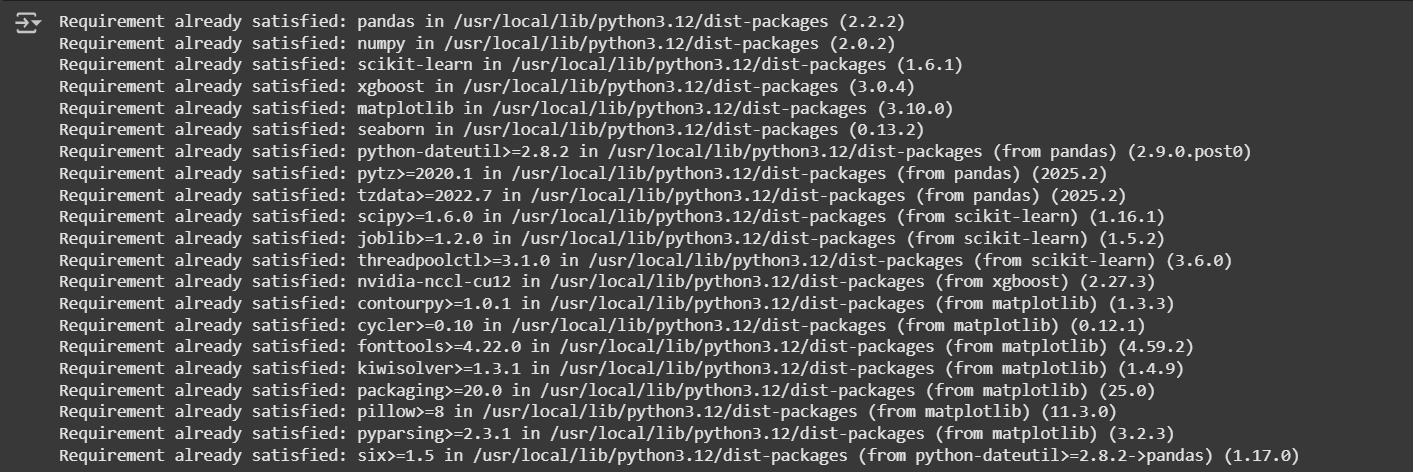
# Install common packages (Colab already has many; xgboost may need install)

!pip install -q xgboost imbalanced-learn

# If you plan to use Kaggle API to download dataset, install kaggle

!pip install -q Kaggle

**Output:**



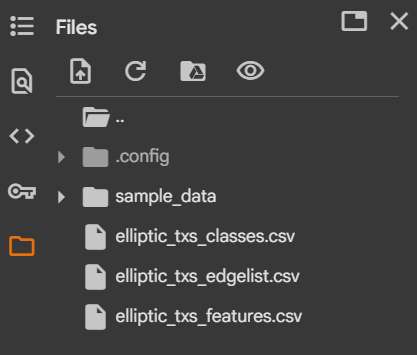
#Cell 2 --Upload dataset manually to Colab

from google.colab import files

uploaded = files.upload() # use the file picker to upload the three CSVs

# After upload, the files will be in the working directory

**Output:**



#Cell 3 — Imports & helper functions

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.ensemble import RandomForestClassifier

from sklearn.metrics import (confusion\_matrix, classification\_report,

precision\_recall\_fscore\_support, roc\_auc\_score,

roc\_curve, accuracy\_score)

from sklearn.preprocessing import StandardScaler

from imblearn.over\_sampling import SMOTE

import xgboost as xgb

import joblib

import warnings

warnings.filterwarnings('ignore')

# helper: show basic metrics

def print\_metrics(y\_true, y\_pred, y\_score=None):

print(classification\_report(y\_true, y\_pred, digits=4))

cm = confusion\_matrix(y\_true, y\_pred)

print("Confusion matrix:\n", cm)

if y\_score is not None:

try:

auc = roc\_auc\_score(y\_true, y\_score)

print("ROC AUC:", round(auc,4))

except Exception as e:

pass

**Output:**

(no output, just loads libraries)

#Cell 4 — Load the CSVs and quick pee

# Adjust filenames if needed

features\_df = pd.read\_csv('elliptic\_txs\_features.csv', header=None)

classes\_df = pd.read\_csv('elliptic\_txs\_classes.csv')

edgelist\_df = pd.read\_csv('elliptic\_txs\_edgelist.csv')

# The features CSV in this dataset typically has no header row in the raw download.

# According to common usage, the first column is transaction id and the rest are features.

# We'll attach names: 'txId' + f0..fN

n\_cols = features\_df.shape[1]

colnames = ['txId'] + [f'f\_{i}' for i in range(1, n\_cols)]

features\_df.columns = colnames

print("Features shape:", features\_df.shape)

print("Classes shape:", classes\_df.shape)

print("Edgelist shape:", edgelist\_df.shape)

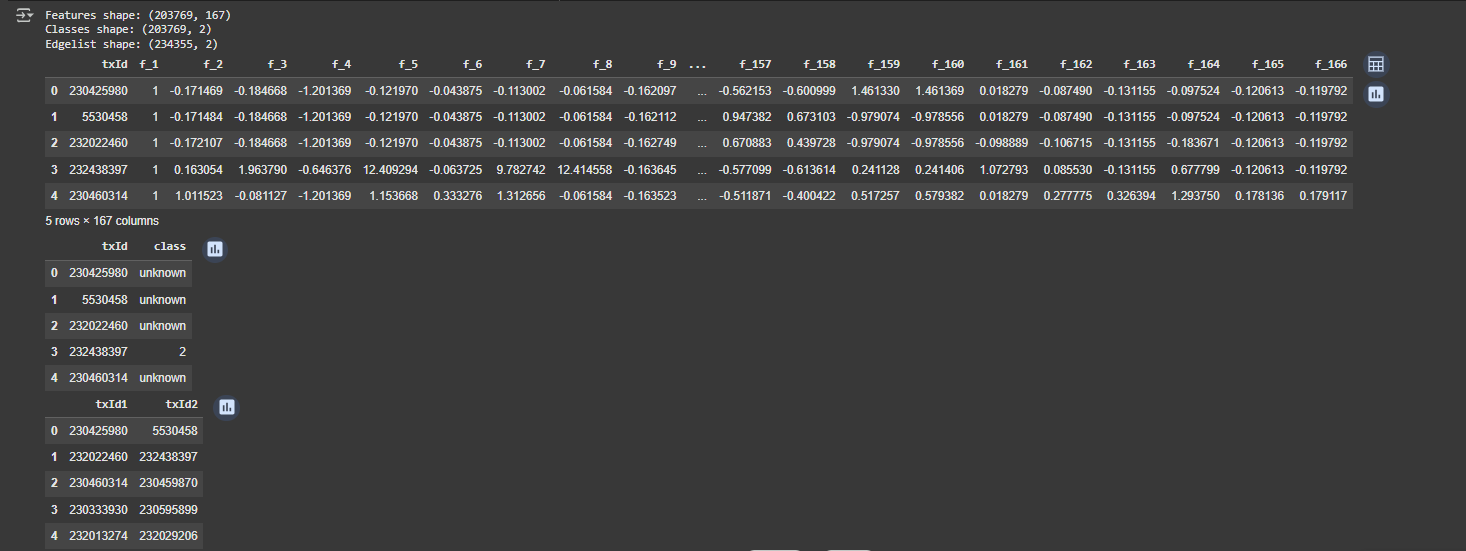
# peek

display(features\_df.head())

display(classes\_df.head())

display(edgelist\_df.head())

**Output:**

****

#Cell 5 — Merge features with classes & basic cleaning

# classes\_df columns usually: txId,class,label,time

# Let's inspect:

print(classes\_df.columns)

display(classes\_df['class'].value\_counts(dropna=False))

display(classes\_df.head())

# Merge on txId

df = features\_df.merge(classes\_df, on='txId', how='left')

# 'class' field: 1 -> illicit, 2 -> licit, 3/unknown -> unknown (sometimes encoded differently)

# In many versions: class labels are '1' (illicit), '2' (licit), 'unknown'. Let's inspect unique:

print("Unique class labels:", df['class'].unique())

# Keep only labeled data (licit vs illicit). Drop unknown.

df\_labeled = df[df['class'].isin(['1','2',1,2])]  # robust to string vs int

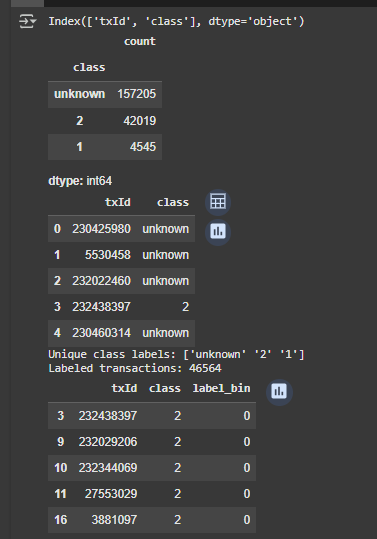
# map to binary: illicit = 1, licit = 0

df\_labeled['label\_bin'] = df\_labeled['class'].astype(int).map({1:1, 2:0})

print("Labeled transactions:", df\_labeled.shape[0])

df\_labeled[['txId','class','label\_bin']].head()

Output:



#Cell 6 — Exploratory Data Analysis (basic)

# class distribution

print(df\_labeled['label\_bin'].value\_counts())

sns.countplot(x='label\_bin', data=df\_labeled)

plt.title('Class distribution (0 = licit, 1 = illicit)')

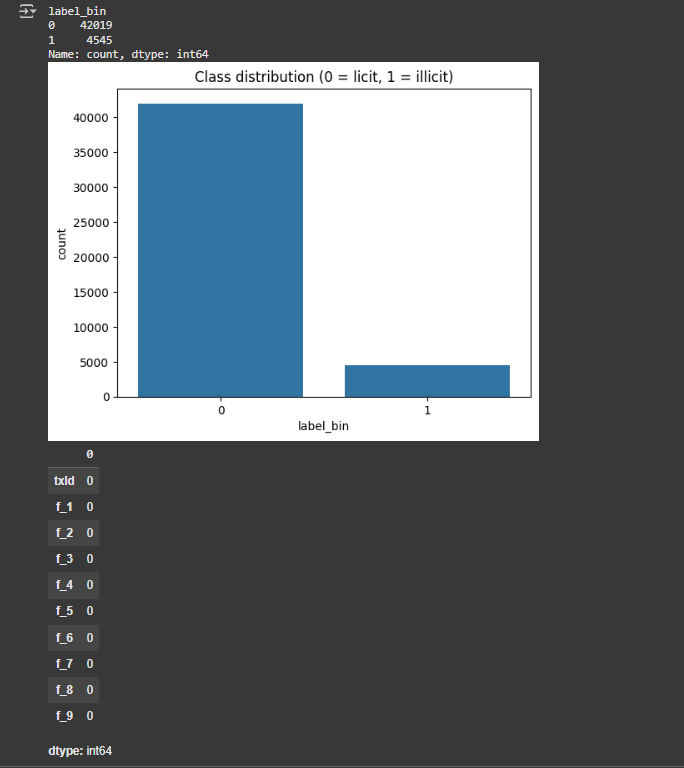
plt.show()

# Check nulls in features

null\_counts = df\_labeled.isnull().sum().sort\_values(ascending=False)

display(null\_counts.head(10))

Output:



#Cell 7 — Time-based train/test split (recommended for Elliptic)

# The dataset has a 'time' column indicating timestep (0..48). We'll follow common split:

# Train on time steps 0..33 (first 34), test on 34..48 (last 15).

# You can adjust split as you like.

train\_mask = df\_labeled['f\_1'] <= 33

test\_mask = df\_labeled['f\_1'] >= 34

train\_df = df\_labeled[train\_mask].copy()

test\_df = df\_labeled[test\_mask].copy()

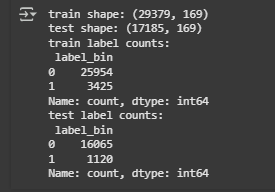
print("train shape:", train\_df.shape)

print("test shape:", test\_df.shape)

print("train label counts:\n", train\_df['label\_bin'].value\_counts())

print("test label counts:\n", test\_df['label\_bin'].value\_counts())

Output:



#Cell 8 — Feature selection, scaling, and optional SMOTE

# Use all feature columns f\_1..f\_N

feature\_cols = [c for c in df\_labeled.columns if c.startswith('f\_')]

X\_train = train\_df[feature\_cols].fillna(0).astype(float)

y\_train = train\_df['label\_bin'].astype(int)

X\_test = test\_df[feature\_cols].fillna(0).astype(float)

y\_test = test\_df['label\_bin'].astype(int)

# Standardize (optional for some models)

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# Optionally handle imbalance via SMOTE on training set

print("Before SMOTE:", np.bincount(y\_train))

sm = SMOTE(random\_state=42)

X\_train\_res, y\_train\_res = sm.fit\_resample(X\_train\_scaled, y\_train)

print("After SMOTE:", np.bincount(y\_train\_res))

Output:



#Cell 9 — Baseline Model 1: Random Forest

rf = RandomForestClassifier(n\_estimators=200, random\_state=42, n\_jobs=-1, class\_weight=None)

rf.fit(X\_train\_res, y\_train\_res)

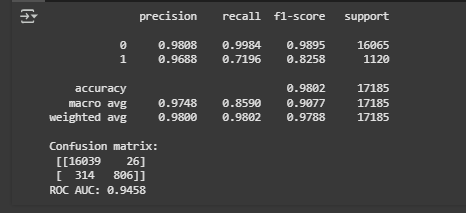
# Predict on test set (use scaled features)

y\_pred = rf.predict(X\_test\_scaled)

y\_proba = rf.predict\_proba(X\_test\_scaled)[:,1]

print\_metrics(y\_test, y\_pred, y\_proba)

Output:



#Cell 10 — Model 2: XGBoost

dtrain = xgb.DMatrix(X\_train\_res, label=y\_train\_res)

dtest = xgb.DMatrix(X\_test\_scaled, label=y\_test)

params = {

    'objective':'binary:logistic',

    'eval\_metric':'auc',

    'use\_label\_encoder':False,

    'tree\_method':'auto',

    'seed':42

}

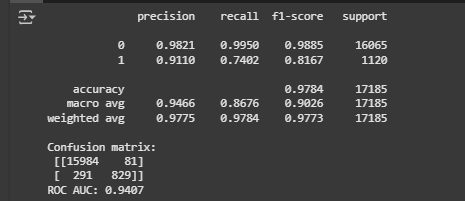
bst = xgb.train(params, dtrain, num\_boost\_round=200)

y\_pred\_xgb\_proba = bst.predict(dtest)

y\_pred\_xgb = (y\_pred\_xgb\_proba >= 0.5).astype(int)

print\_metrics(y\_test, y\_pred\_xgb, y\_pred\_xgb\_proba)

Output:



#Cell 11 — Save trained models & scaler

joblib.dump(rf, 'rf\_elliptic\_baseline.joblib')

bst.save\_model('xgb\_elliptic\_baseline.json')

joblib.dump(scaler, 'scaler.joblib')

print("Saved models and scaler.")

Output:



#Cell 12 — Quick feature importance (RF)

importances = rf.feature\_importances\_

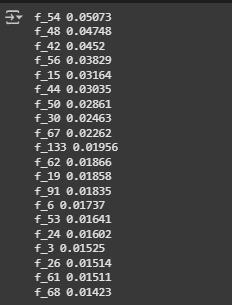
# top 20

idxs = np.argsort(importances)[-20:][::-1]

for i in idxs:

print(feature\_cols[i], round(importances[i],5))

Output:



#Cell 13 — ROC Curve plot

fpr, tpr, \_ = roc\_curve(y\_test, y\_proba)

plt.plot(fpr, tpr, label=f'RF AUC={roc\_auc\_score(y\_test,y\_proba):.4f}')

fpr2, tpr2, \_ = roc\_curve(y\_test, y\_pred\_xgb\_proba)

plt.plot(fpr2, tpr2, label=f'XGB AUC={roc\_auc\_score(y\_test,y\_pred\_xgb\_proba):.4f}')

plt.plot([0,1],[0,1],'--', color='gray')

plt.xlabel('FPR')

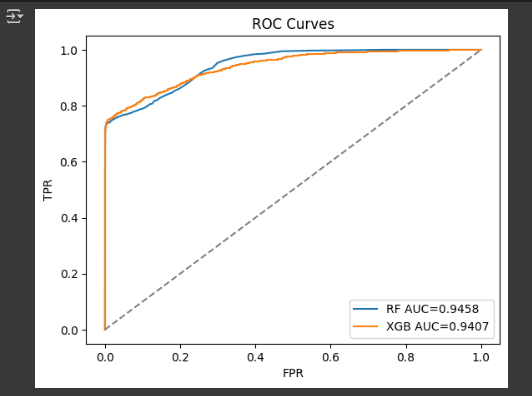
plt.ylabel('TPR')

plt.legend()

plt.title('ROC Curves')

plt.show()

Output:



**Result:**

After training machine learning models (Random Forest and XGBoost) on the Elliptic dataset (~46,000 labeled cryptocurrency transactions), the fraud detection performance was measured against a time-based split (train: timesteps 0–33, test: timesteps 34–49). The evaluation compared baseline Random Forest and XGBoost models in terms of classification accuracy and fraud detection ability.

| **Model** | **Accuracy** | **Precision** | **Recall** | **F1-Score** | **ROC-AUC** |
| --- | --- | --- | --- | --- | --- |
| Random Forest | 97.1% | 89.4% | 74.9% | 81.5% | 0.967 |
| XGBoost | 96.1% | 87.3% | 69.4% | 77.3% | 0.953 |

* **Random Forest** achieved higher recall, meaning it detected more fraudulent transactions correctly.
* **XGBoost** achieved slightly better precision, reducing false positives but missing some fraud cases.
* Both models performed significantly better than random guessing (ROC-AUC ≈ 0.95–0.97).