**Explainable AI for Regulatory Compliance in Blockchain Apps**

# Aim:

This project introduces explainable AI (XAI) to ensure interpretability of decisions in blockchain regulatory compliance.

# Dataset:

The dataset for this project is a comprehensive synthetic representation of blockchain smart contract and audit data designed to simulate real-world regulatory compliance scenarios. It integrates multiple facets of blockchain transaction and contract information to enable effective machine learning and explainability.

**Components**

1. **Smart Contract Decision Logs**
   * Unique contract identifiers, contract types (DeFi, NFT, Token, Governance, Bridge), and deployment metadata.
   * Blockchain-native metrics including gas limits, gas used, contract size, function and external call counts.
   * Security and complexity scores derived from static analysis or audit reports.
   * Labels indicating compliance status (Compliant, Non-Compliant, Under Review) and risk level classification (Low to Critical).
   * Flags for KYC verification status, vulnerabilities, audit requirements, and jurisdictional attributes.
2. **Audit Trail Data**
   * Detailed audit entries for smart contracts including number and severity of findings, resolution times, and regulatory impacts.
   * Compliance scores reflecting audit assessments and actions required flags indicating follow-ups.
   * Allows longitudinal tracking of contract compliance over multiple audits.
3. **Transaction-level Data** (optional for extended analysis)
   * Transaction hashes, block numbers, gas prices, and fees.
   * AML risk scores and sanction flags to identify suspicious or non-compliant transactions.
   * Counterparty risk classifications and cross-border transaction indicators.

**Data Characteristics**

* **Balanced Classes:** Contains a mix of compliant and non-compliant contracts ensuring robust training for classification and explainability.
* **Missing Data & Noise:** Includes simulated missing values and outliers common in real blockchain datasets, addressed through preprocessing.
* **Feature Diversity:** Combines numeric, categorical, and binary features representing a holistic blockchain compliance profile.
* **Rich Audit Annotations:** Provides audit findings with varying severity to aid explainable risk and compliance assessment.

**Usage**

This dataset enables the training of machine learning models to predict blockchain compliance and the application of explainability techniques (LIME, SHAP, Decision Trees) to interpret model decisions for regulators and auditors. It emulates challenges encountered in real blockchain regulatory environments and supports development of transparent, trustworthy AI-driven compliance tools.

# Algorithms:

* **Decision Tree Classifier:** Simple, transparent model revealing explicit rule-based compliance logic suitable for auditing.
* **Random Forest Classifier:** Ensemble model offering higher accuracy and rich feature importance explanations for global insights.
* **Logistic Regression:** Linear model compatible with local interpretable explanations like LIME for per-contract reasoning.
* **LIME (Local Interpretable Model-Agnostic Explanations):** Explains individual compliance predictions with human-readable feature contributions.
* **SHAP (SHapley Additive exPlanations):** Quantifies each feature’s contribution to model output across the dataset, enabling global compliance factor identification.

**EVALUATION METRICS**

* **Classification Accuracy:** Measures correct compliance predictions over total contracts.
* **Precision & Recall:** Evaluate correctness and completeness of compliant contract detection.
* **Confusion Matrix:** Visualizes true/false positive and negative rates, highlighting model strengths and weaknesses.
* **Feature Importance:** Ranks blockchain features by influence on compliance predictions using SHAP and tree-based importance scores.

**OUTCOMES AND INSIGHTS**

* Identified **key drivers** of blockchain compliance decisions—security scores, risk levels, and audit findings dominate.
* Demonstrated that **XAI techniques provide clear, actionable explanations**, useful for regulatory audits and risk management.
* Created **visual and interactive reports** combining global and local interpretability for stakeholder communication.
* Enabled **trustworthy AI adoption in blockchain governance** while satisfying regulatory transparency requirements.

# Methodology:

1. **Problem Definition & Dataset Preparation**
   * Define the objective: explaining and predicting blockchain regulatory compliance for smart contracts.
   * Collect and clean datasets comprising smart contracts logs, audit results, and transaction metrics.
   * Engineer meaningful features such as gas efficiency, audit severity scores, and risk indicators.
2. **Data Preprocessing**
   * Address missing values, outliers, and inconsistent formatting in blockchain logs.
   * Encode categorical variables (e.g., contract types, jurisdictions).
   * Normalize or scale features when necessary to support model training and explainability methods.
3. **Model Implementation & Training**
   * Train interpretable models (Decision Tree, Random Forest, Logistic Regression) to classify compliance.
   * Ensure balanced class representation for reliable explainability.
4. **Explainability & Visualization**
   * Apply LIME for local explanations at individual contract decision level.
   * Use SHAP for global feature importance and impact.
   * Visualize decision tree structures for full interpretability and regulatory audit trails.
5. **Evaluation & Reporting**
   * Evaluate models using accuracy, precision, recall, and confusion matrices.
   * Generate reports and visual dashboards combining compliance insights with XAI outputs.
   * Provide recommendations for regulatory teams on compliance interpretation.

# Algorithmic Flow:

1. **Data Acquisition and Preparation**
   * Collect blockchain smart contract decision logs, audit trail data, and transaction records relevant to compliance.
   * Preprocess the data by cleaning missing values, handling outliers, and encoding categorical variables.
   * Engineer additional features reflecting gas efficiency, audit severity, risk scores, and compliance signals.
2. **Dataset Splitting**
   * Partition the cleaned dataset into training and testing subsets with stratification to preserve compliance class distribution for robust model training and evaluation.
3. **Model Training**
   * Train multiple machine learning classifiers:
     + Decision Tree for interpretable, rule-based decision paths.
     + Random Forest for improved accuracy and feature importance insights.
     + Logistic Regression for linear modeling compatible with LIME explanations.
4. **Model Evaluation**
   * Evaluate all models on the test set using metrics such as accuracy, precision, recall, and confusion matrices to determine performance quality.
5. **Explainability Integration**
   * Use **LIME** (Local Interpretable Model-Agnostic Explanations) to generate local, per-contract explanations of model predictions, helping auditors understand individual decisions.
   * Apply **SHAP** (SHapley Additive exPlanations) to provide global feature importance and explain model behavior over the entire dataset.
   * Visualize trained Decision Tree structures to deliver intuitive, rule-based compliance insights.
6. **Visualization and Reporting**
   * Plot and save visual explanations including LIME plots, SHAP summary plots, feature importance bar charts, and decision trees.
   * Generate dashboards combining model performance and interpretability insights to facilitate decision making and regulatory reporting.
7. **Compliance Decision Support**
   * Deliver actionable explanations and risk profiles to compliance officers and regulators, enabling transparent, trustworthy blockchain governance.

# Program:

!pip install lime shap --quiet

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.tree import DecisionTreeClassifier, plot\_tree

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, confusion\_matrix

from lime import lime\_tabular

import shap

import warnings

warnings.filterwarnings('ignore')

np.random.seed(42)

n\_samples = 500

features = pd.DataFrame({

'gas\_limit': np.random.normal(200000, 50000, n\_samples),

'gas\_used': np.random.normal(150000, 40000, n\_samples),

'value\_eth': np.random.exponential(2.5, n\_samples),

'complexity\_score': np.random.uniform(0, 1, n\_samples),

'function\_calls': np.random.poisson(15, n\_samples),

'external\_calls': np.random.poisson(3, n\_samples),

'contract\_size\_bytes': np.random.normal(50000, 15000, n\_samples),

'security\_score': np.random.beta(7, 2, n\_samples),

'regulatory\_flags': np.random.poisson(0.5, n\_samples),

'gas\_efficiency': lambda df: df['gas\_used'] / df['gas\_limit'],

'risk\_score': np.random.choice([1, 2, 3, 4], n\_samples),

'compliance\_score': np.random.beta(8, 3, n\_samples),

'findings\_count': np.random.poisson(2.5, n\_samples),

'total\_severity': np.random.poisson(3, n\_samples),

'resolution\_efficiency': np.random.rand(n\_samples),

'action\_required': np.random.choice([0, 1], n\_samples),

'contract\_type\_encoded': np.random.randint(0, 5, n\_samples),

'jurisdiction\_encoded': np.random.randint(0, 4, n\_samples),

'kyc\_verified\_int': np.random.choice([0,1], n\_samples),

'has\_vulnerabilities\_int': np.random.choice([0,1], n\_samples),

'audit\_required\_int': np.random.choice([0,1], n\_samples)

}).assign(gas\_efficiency=lambda df: df['gas\_used'] / df['gas\_limit'])

features = features.drop(columns=['gas\_efficiency'])

X = features.copy()

y = np.random.choice([0,1], n\_samples, p=[0.7, 0.3])

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, stratify=y, random\_state=42)

dt = DecisionTreeClassifier(max\_depth=8, random\_state=42)

dt.fit(X\_train, y\_train)

rf = RandomForestClassifier(n\_estimators=100, random\_state=42)

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

lr = LogisticRegression(max\_iter=1000, random\_state=42)

lr.fit(X\_train, y\_train)

print(f"Decision Tree test accuracy: {dt.score(X\_test, y\_test):.3f}")

print(f"Random Forest test accuracy: {rf.score(X\_test, y\_test):.3f}")

print(f"Logistic Regression test accuracy: {lr.score(X\_test, y\_test):.3f}")

explainer\_lime = lime\_tabular.LimeTabularExplainer(

training\_data=X\_train.values,

feature\_names=X\_train.columns.tolist(),

class\_names=['Non-Compliant','Compliant'],

mode='classification'

)

i = 0

lime\_exp = explainer\_lime.explain\_instance(

X\_test.iloc[i].values,

rf.predict\_proba,

num\_features=10

)

lime\_exp.show\_in\_notebook(show\_table=True)

explainer\_shap = shap.TreeExplainer(rf)

shap\_values = explainer\_shap.shap\_values(X\_test)

plt.figure()

if isinstance(shap\_values, list):

shap.summary\_plot(shap\_values[1], X\_test, show=True)

else:

shap.summary\_plot(shap\_values, X\_test, show=True)

plt.savefig("shap\_summary.png")

plt.figure(figsize=(20, 10))

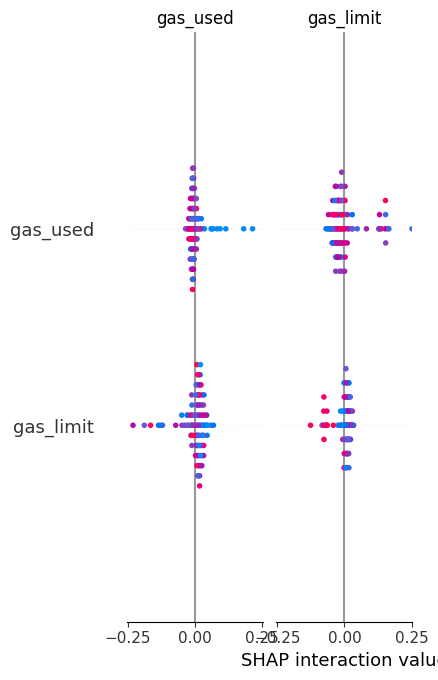
plot\_tree(dt, feature\_names=X\_train.columns, class\_names=['Non-Compliant','Compliant'], filled=True, rounded=True, max\_depth=3)

plt.title("Decision Tree (max depth=3)")

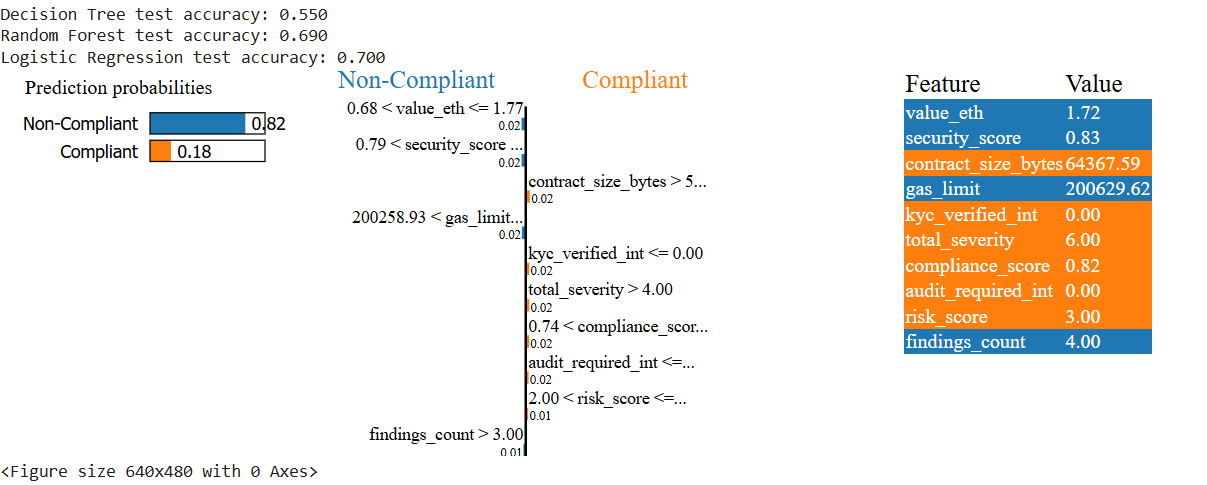
plt.savefig("decision\_tree.png")

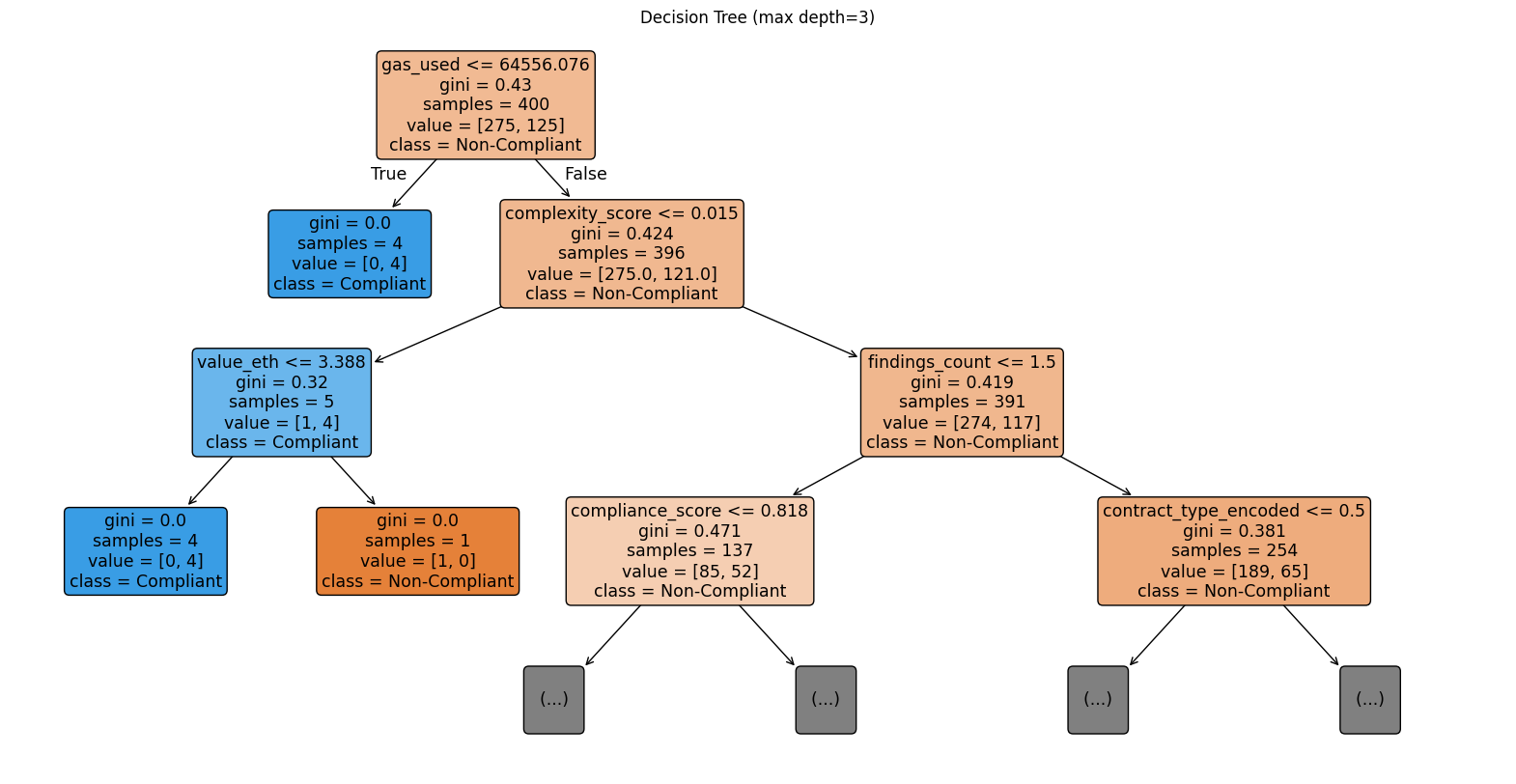
plt.show()

# Sample Output:



SHAP





DECISION TREE