Hackathon edition#3

April 2025

Credit Card Fraud Detection: Model Comparison and Evaluation

Team Details :

Manish Tiwari (Leader) (IITMCS_2406327)
Anushka Sarkar (IITMCS_2406255)
Jay Paneliya (IITMCS_24091607)

Project link

https://github.com/JAYpaneliya/credit-card-fraud-detection

Problem Statement :

In real-world applications, fraud detection remains one of the most critical and complex classification challenges due to highly imbalanced datasets and evolving fraud patterns. The objective of this project was to:

- Train and evaluate multiple machine learning classification models
- Compare their performance on key metrics (Precision, Recall, F1-score, etc.)
- Perform hyperparameter tuning to improve model effectiveness
- Handle class imbalance to ensure realistic fraud detection capabilities
- Build an interactive dashboard for comparison, visualization, and exploration

The challenge emphasized not only model performance but also the ability to present results in a clear and insightful manner using modern tools like Streamlit.

Dataset Description :

Source :

The dataset used for this project is the **Credit Card Fraud Detection** dataset made publicly available on Kaggle. It contains real transaction data collected by European cardholders in September 2013. The dataset was downloaded directly from the Kaggle platform.

https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud/data

📊 Dataset Overview

- **Total Records:** 284,807 transactions
- **Fraudulent Transactions:** 492
- **Non-Fraudulent Transactions:** 284,315
- **Imbalance Ratio:** ~0.172% fraud (highly imbalanced)

🧾 Key Features

- Features are result of **PCA anonymization**, named `V1` to `V28`
- Two original features:
  - `Time`: Seconds elapsed between the transaction and the first transaction in the dataset
  - `Amount`: Transaction amount
- **Target Variable**:
  - `Class`: 0 = Non-Fraud, 1 = Fraud

🛠️ Preprocessing Steps:

- **Scaled** `Amount` and `Time` using `StandardScaler`
- **Removed missing values** (none present in this dataset)
- **Applied SMOTE** (Synthetic Minority Oversampling Technique) to balance the dataset before training

🧠 Our Approach

To address the challenge of credit card fraud detection on a highly imbalanced dataset, we followed a structured, multi-step machine learning pipeline:

Step 1: Data Preprocessing

- **Feature Scaling**: The `Amount` and `Time` features were scaled using `StandardScaler` to normalize their ranges.

- **Missing Values**: Dataset was checked for null or missing values; none were found.

- **Class Imbalance Handling**:

    - The dataset is highly imbalanced (~0.17% fraud cases).

    - We applied **SMOTE (Synthetic Minority Over-sampling Technique)** to the training data to synthetically generate minority class samples and improve model learning.

Step 2: Train-Test Split

- An **80/20 split** was used to divide the dataset.

    - 80% for training and SMOTE balancing

    - 20% untouched for testing real-world performance

Step 3: Model Selection

We selected and trained **five popular classification algorithms**:

Model	**Description**
Logistic Regression	A linear baseline classifer
Random Forest	Esemble method using decision trees
SVM	Kernal based non-linear classifier
Gradient Boosting	Boosted decision trees
XGBoost	High performance boosting algorithm

Each model was trained on the **SMOTE- balanced training set**.

Evaluation Strategy :

We evaluated all models on the original, imbalanced test set using:

- **Precision** (focus on reducing false positives)
- **Recall** (focus on catching actual frauds)
- **F1-score** (balance between precision & recall)
- **Confusion Matrix**
- **ROC Curve & AUC Score**

This multi-metric evaluation ensured a balanced view of model strengths, especially under imbalanced conditions.

Hyperparameter Tuning :

We applied **RandomizedSearchCV** on the XGBoost model using:

- F1-score as the optimization metric
- 3-fold cross-validation
- Multiple combinations of learning rate, depth, and estimators

This tuning significantly improved performance and reduced false negatives.

Implementation Details

Our solution was implemented entirely using Python, with a focus on machine learning model development, evaluation, and visualization. The workflow was developed using **Google Colab** and later integrated into a **Streamlit dashboard** for interactive presentation.

⚙️ Technologies and Libraries Used

Tool/Library	Purpose
Pandas	Data loading and manipulation
NumPy	Numerical operations
Scikit-learn	Model training, SMOTE, metrics, tuning
XGBoost	High-performance boosting model
Matplotlib / Seaborn	Visualizations for evaluation
SHAP	Model interpretability (optional)
Streamlit	Interactive dashboard interface
Joblib	Saving and loading trained models

📦 Model Training & Saving :

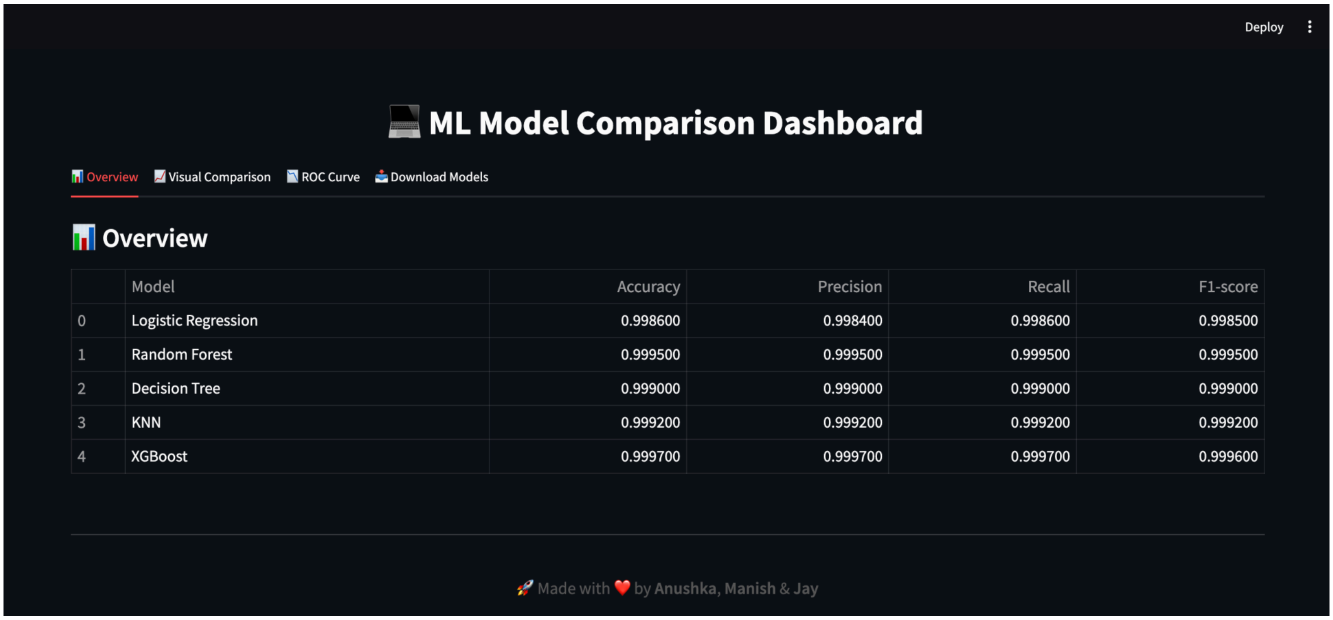
Each model was trained on the SMOTE-balanced dataset and evaluated using the imbalanced test set. Trained models were saved as `.pkl` files using `joblib.dump()` and loaded later into the dashboard.

📈 Dashboard (Streamlit) :

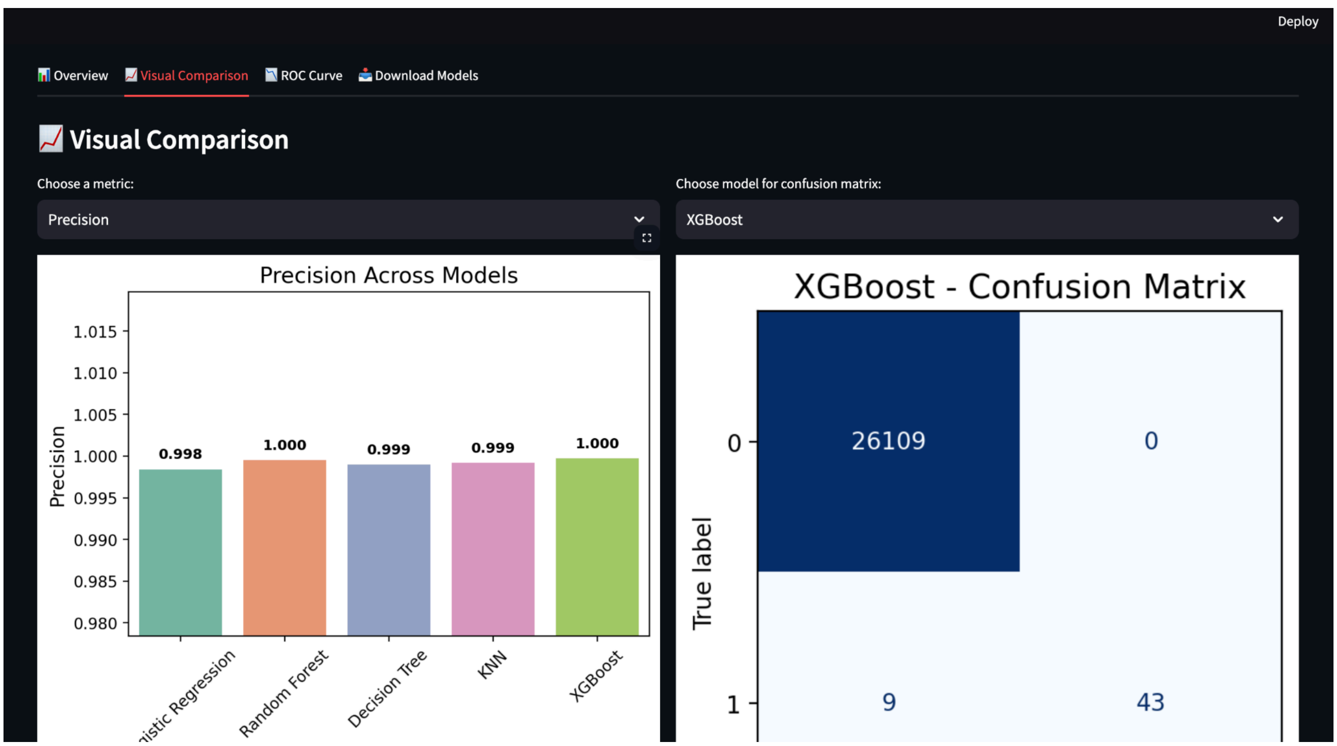
We built a multi-tab Streamlit dashboard with the following features:

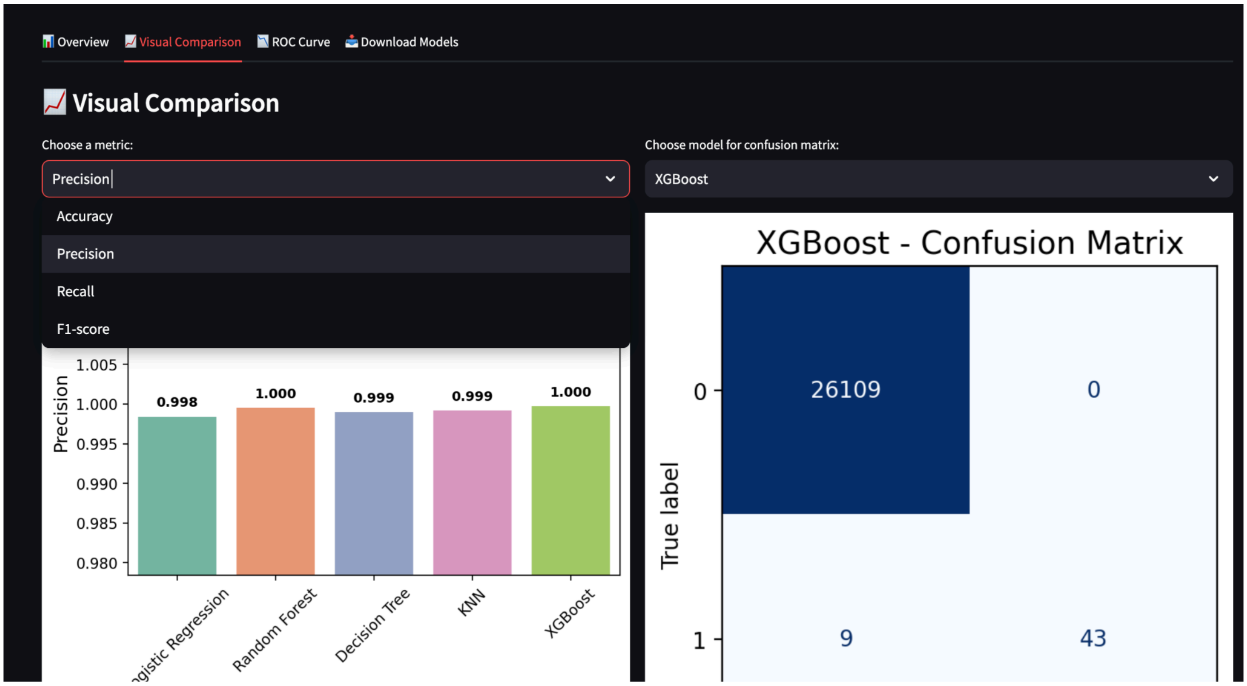
1. **Overview Tab**: Displays model evaluation metrics in a clean table
2. **Visual Comparison Tab**: Allows comparison of precision, recall, and F1-score across models using bar plots; also includes confusion matrix viewer
3. **ROC Curve Tab**: Lets users view and compare AUC-ROC curves model-wise
4. **Download Models Tab**: Provides trained `.pkl` files for download

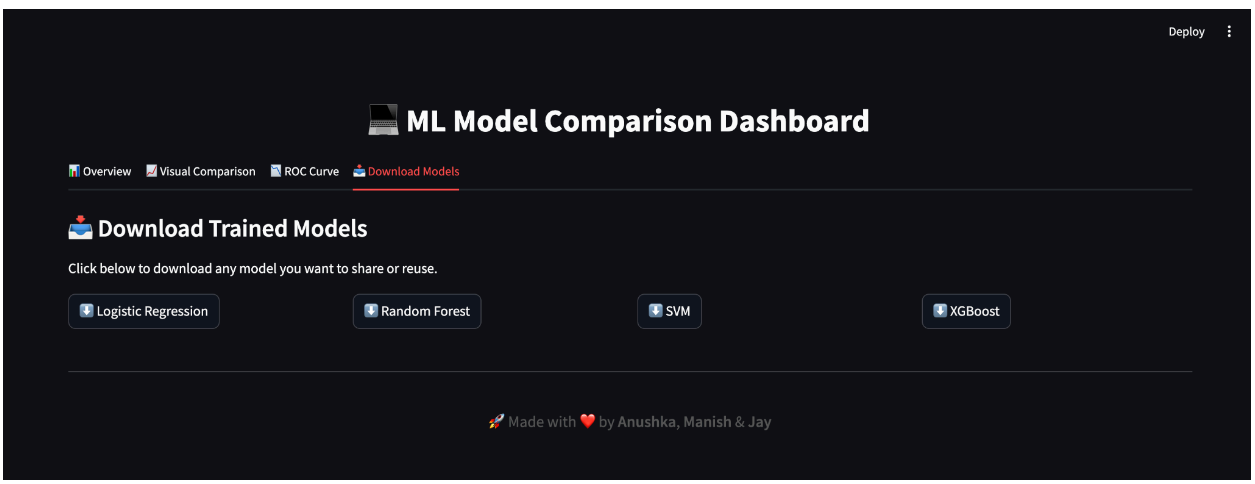
👉 *Screenshots of the dashboard are provided in the next page.*

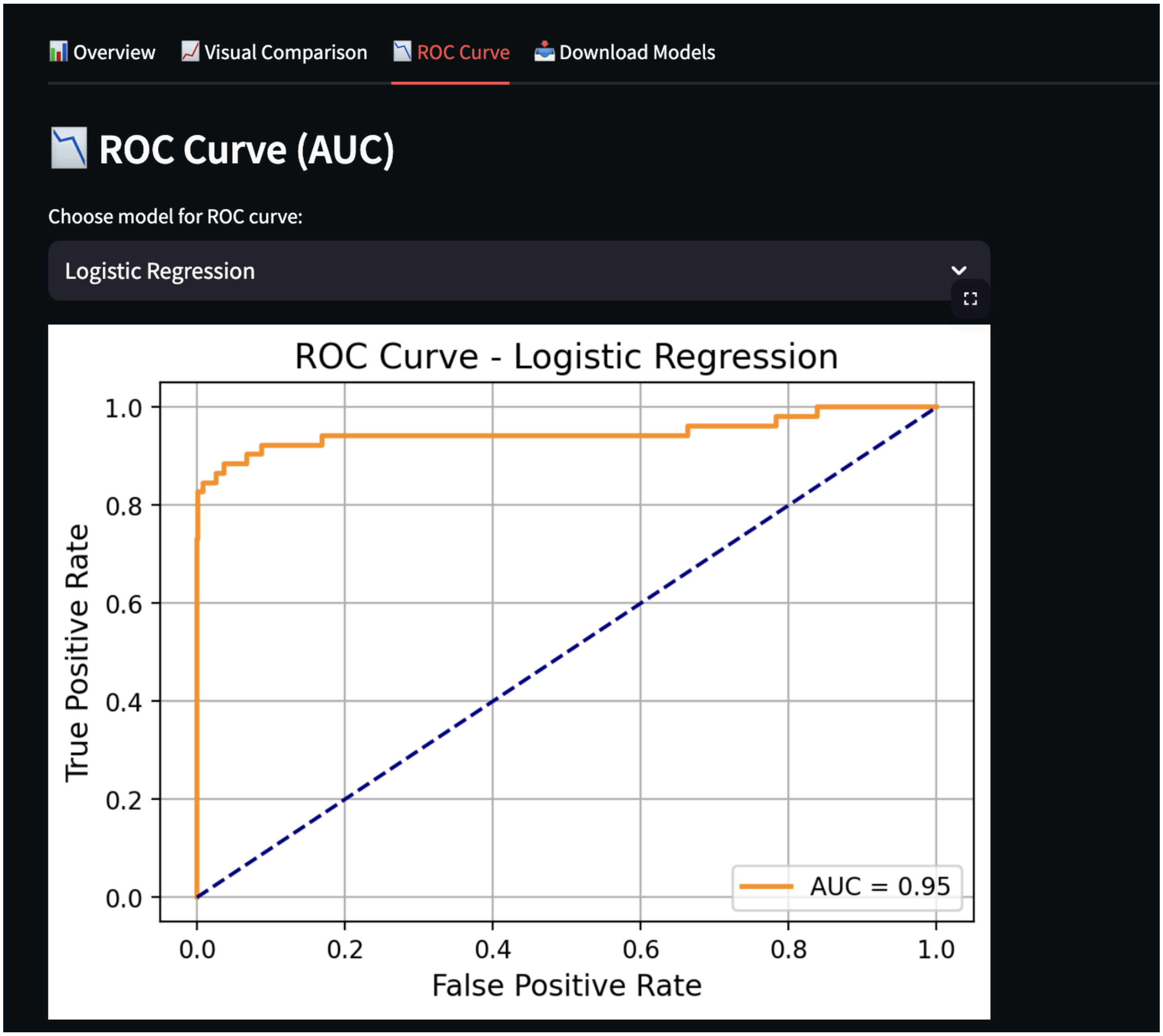


	Model	Accuracy	Precision	Recall	F1-score
0	Logistic Regression	0.998600	0.998400	0.998600	0.998500
1	Random Forest	0.999500	0.999500	0.999500	0.999500
2	Decision Tree	0.999000	0.999000	0.999000	0.999000
3	KNN	0.999200	0.999200	0.999200	0.999200
4	XGBoost	0.999700	0.999700	0.999700	0.999600




Overview
Visual Comparison
ROC Curve
Download Models
Visual Comparison
Choose a metric:
Precision
Accuracy
Precision
Recall
F1-score
Choose model for confusion matrix:
XGBoost
XGBoost - Confusion Matrix
26109
0
9
43
True label
Precision
1.005
1.000
0.995
0.990
0.985
0.980
0.998
1.000
0.999
0.999
1.000
Random Forest
Decision Tree
KNN
XGBoost


Deploy
ML Model Comparison Dashboard
Overview
Visual Comparison
ROC Curve
Download Models
Download Trained Models
Click below to download any model you want to share or reuse.
Logistic Regression
Random Forest
SVM
XGBoost
Made with by Anushka, Manish & Jay


📊 Overview
📈 Visual Comparison
📉 ROC Curve
📥 Download Models
📉 ROC Curve (AUC)
Choose model for ROC curve:
Logistic Regression
ROC Curve - Logistic Regression
True Positive Rate
False Positive Rate
AUC = 0.95
0.0
0.2
0.4
0.6
0.8
1.0

Key Features :

📊 Overview Tab

- Displays a table with metrics like **Precision**, **Recall**, and **F1-score** for each model.
- Styled for clarity and emphasis using font scaling and color.

📈 Visual Comparison Tab

- Users can select a metric (e.g., F1-score) from a dropdown.
- A bar plot shows how models compare.
- A separate dropdown loads the **Confusion Matrix** for any model.

📉 ROC Curve Tab

- Plots **ROC Curve** for a selected model.
- Displays **AUC score** visually for better model comparison.

📥 Download Tab

- Allows downloading of trained models in `.pkl` format.
- Great for portability or integration into other systems.

Key Insights :

Through model experimentation, evaluation, and visualization, several key insights emerged:

1. XGBoost Outperformed Other Models

- After hyperparameter tuning, **XGBoost delivered the best balance between precision and recall**.

- It showed a strong **F1-score**, indicating reliability in detecting fraud while minimizing false alarms.

2. Precision vs Recall Trade-Off

- **Logistic Regression** and **SVM** had high precision but lower recall — they were more conservative in flagging fraud.

- **Random Forest** and **Gradient Boosting** provided better recall but risked more false positives.

- This highlighted the importance of **F1-score** and **ROC-AUC** as balanced evaluation metrics in imbalanced datasets.

3. Importance of Data Resampling (SMOTE)

- Using SMOTE drastically improved recall across all models, especially those that otherwise performed poorly on rare fraud examples.

- Without SMOTE, most models underfit the fraud class due to data imbalance.

4. Dashboard Helped Interpret Results Quickly

- The **bar charts, confusion matrices, and ROC curves** made it easy to compare models visually.

- Stakeholders or judges could quickly identify which model to trust and why.

5. Feature Importance Helped Understand Predictions

- SHAP values (used optionally during model tuning) revealed which features most influenced fraud detection.
- Although features were anonymized (V1–V28), their relative importance offered meaningful interpretability.

Conclusion :

In this project, we successfully developed and evaluated a robust credit card fraud detection system using multiple machine learning models. We addressed the challenges of **class imbalance**, **model tuning**, and **evaluation interpretability** through a systematic pipeline.

Key achievements include:

- ✅ **Training and comparing five classification models** on real-world fraud data
- ✅ **Balancing the data using SMOTE** to handle extreme class imbalance
- ✅ **Hyperparameter tuning** to optimize model performance
- ✅ **Comprehensive evaluation** using F1-score, ROC-AUC, and confusion matrices
- ✅ Building an **interactive Streamlit dashboard** to visualize and compare model performance

The final solution offers a **transparent, explainable, and easily deployable system** for fraud detection in financial datasets.

Future Scope :

While the current solution is robust and production-ready, there are several opportunities to expand and improve this project:

1. Real-Time Prediction & Deployment

- Integrate the trained models into a real-time fraud detection pipeline using **APIs** or **batch prediction endpoints**.
- Serve the dashboard using **Streamlit Cloud**, **Heroku**, or **AWS** for public access.

2. Feature Engineering

- Investigate derived features like **transaction frequency**, **user velocity**, or **time of day**.
- These could enhance fraud detection accuracy with minimal overhead.

3. Advanced Algorithms

- Implement **LightGBM**, **CatBoost**, or **Neural Networks** to explore deeper learning techniques.
- Add **stacking** or **voting ensembles** to combine strengths of multiple models.

4. Precision-Recall Curves & Cost Analysis

- Add **PR curves** to complement ROC, especially in imbalanced settings.
- Conduct **cost-sensitive evaluation**, e.g., how much a false negative costs vs a false positive.

5. Data Privacy & Explainability

- Integrate **LIME/SHAP explanations** into the dashboard so users can interpret individual predictions.

- Consider techniques for **model auditing and bias analysis**, especially in financial applications.

6. User Upload Feature

- Let users upload new transaction CSVs in the dashboard and get predictions with fraud probability and SHAP explanations.

This roadmap would help evolve the current dashboard into a full-fledged ML-powered fraud detection platform.

Video presentation link :

https://drive.google.com/file/d/1wevhGeFYOb8kEVRtqN1gAyoC0K6y1-ZU/view?usp=drive_link