**Fraud Detection Project Report**

**1. Introduction**

Financial fraud, especially in the context of credit card transactions, is a persistent challenge that leads to substantial monetary losses for institutions and individuals. The primary objective of this project is to develop a robust machine learning solution capable of detecting fraudulent credit card transactions by analyzing patterns and anomalies in transaction data.

**2. Dataset Overview**

* **Dataset Source:** Kaggle's Credit Card Fraud Detection dataset
* **Total Records:** 284,807 transaction instances
* **Features:** 30 anonymized numerical features derived from principal component analysis (PCA), alongside two additional features: Time and Amount
* **Target Variable:** Class
  + 0: Legitimate transaction
  + 1: Fraudulent transaction

**2.1 Key Characteristics**

* **Imbalance:** The dataset is highly imbalanced with fraudulent cases constituting only about 0.17% of the total transactions.
* **Data Security:** Feature anonymization ensures data privacy and compliance with data protection standards.

**3. Tools and Technologies**

* **Programming Language:** Python 3
* **Data Handling:** Pandas, NumPy
* **Machine Learning:** Scikit-learn
* **Imbalance Handling:** Imbalanced-learn (SMOTE)
* **Model Serialization:** Joblib
* **API Development:** FastAPI
* **Server:** Uvicorn

**4. Methodology**

**4.1 Data Preprocessing**

* **Loading:** The dataset is imported using Pandas.
* **Feature Separation:** Independent features are separated from the target class.
* **Normalization:** StandardScaler is applied to standardize the feature set, ensuring all variables contribute equally to the model.

**4.2 Addressing Class Imbalance**

* **Technique:** SMOTE (Synthetic Minority Over-sampling Technique) generates synthetic samples for the minority class to balance the dataset.
* This step is crucial to improve the model's sensitivity to fraudulent transactions.

**4.3 Model Development**

* **Algorithm:** Random Forest Classifier with 100 decision trees.
* **Data Split:** The dataset is divided into 70% training and 30% testing subsets.
* **Training:** The model is trained on the balanced and scaled dataset.

**4.4 Evaluation Metrics**

* **Confusion Matrix:** Evaluates true positives, true negatives, false positives, and false negatives.
* **Classification Report:** Provides precision, recall, F1-score, and support for each class.
* **ROC-AUC Score:** Measures the ability of the model to distinguish between classes.

**4.5 Model Saving**

* The trained model and the scaler are serialized using Joblib, allowing for reuse without retraining.

**5. API Development**

* **Framework:** FastAPI is employed to expose the machine learning model as a RESTful API.
* **Endpoint:**
  + **POST /predict:** Accepts transaction data as a JSON payload and returns a prediction indicating the likelihood of fraud along with the probability score.
* **Workflow:**
  1. Accept feature inputs.
  2. Standardize inputs using the saved scaler.
  3. Generate predictions using the trained Random Forest model.
  4. Return results in JSON format.

**6. Project Execution Guide**

**6.1 Prerequisites**

* Python 3.x
* Install required packages:

pip install pandas numpy scikit-learn imbalanced-learn joblib fastapi uvicorn

**6.2 Running the Model**

* Execute the training script to generate the model and scaler files.
* Run the API server:

uvicorn <script\_name>:app --reload

* Access the API locally at http://localhost:8000/predict

**6.3 Sample API Request**

{

"features": [feature\_1, feature\_2, ..., feature\_30]

}

**6.4 Sample API Response**

{

"fraud\_prediction": 0,

"fraud\_probability": 0.02

}

**7. Deployment Recommendations**

* **Local Hosting:** Suitable for development and initial testing.
* **Dockerization:** Recommended for environment consistency and portability.
* **Cloud Deployment:** Platforms like AWS EC2, GCP Compute Engine, or Azure App Services for scalable and secure deployment.

**8. Potential Improvements**

* Implement real-time data streaming for immediate fraud detection.
* Experiment with advanced algorithms such as Gradient Boosting, XGBoost, or deep learning techniques.
* Incorporate explainability frameworks like SHAP for model interpretability.
* Set up monitoring and alert systems for fraud prediction activities.

**9. GitHub link**

https://github.com/Satyavk098/ML\_project/blob/main/Fraud\_Detection\_project.ipynb

**10. Conclusion**

This project demonstrates a complete pipeline from data preprocessing and model training to API deployment for credit card fraud detection. By addressing class imbalance and creating an accessible API, the system is well-prepared for integration into financial applications where fraud mitigation is critical.