**TASK 2(ML):**

**CREDIT CARD FRAUD**

**DETECTION**

**Build a model to detect fraudulent credit card transactions. Use a**

**dataset containing information about credit card transactions, and**

**experiment with algorithms like Logistic Regression, Decision Trees,**

**or Random Forests to classify transactions as fraudulent or**

**legitimate.**

**Load the Dataset**

import pandas as pd

df = pd.read\_csv('credit\_card\_transactions.csv')

# Display the first few rows

print(df.head())

EX:

**print(df['Class'].value\_counts())**

**print(df.isnull().sum())**

**print(df['Class'].value\_counts())**

**Handle Imbalanced Data**

**Fraudulent transactions are usually rare. You may need to balance the dataset using techniques like undersampling, oversampling, or using synthetic data.**

**python**

**Copy code**

**from imblearn.over\_sampling import SMOTE**

**X = df.drop('Class', axis=1)**

**y = df['Class']**

**# Apply SMOTE to balance the dataset**

**smote = SMOTE()**

**X\_resampled, y\_resampled = smote.fit\_resample(X, y)**

**Split the Data**

**Split the dataset into training and testing sets.**

**python**

**Copy code**

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

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_resampled, y\_resampled, test\_size=0.**

**Feature Scaling**

**Many machine learning algorithms perform better with scaled features. Use standardization or normalization.**

**python**

**Copy code**

**from sklearn.preprocessing import StandardScaler**

**scaler = StandardScaler()**

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

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

**Model Training**

**a. Logistic Regression**

**python**

**Copy code**

**from sklearn.linear\_model import LogisticRegression**

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

**lr\_model = LogisticRegression()**

**lr\_model.fit(X\_train\_scaled, y\_train)**

**y\_pred\_lr = lr\_model.predict(X\_test\_scaled)**

**print("Logistic Regression Classification Report:\n", classification\_report(y\_test, y\_pred\_lr))**

**print("Logistic Regression Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred\_lr))**

**b. Decision Tree**

**python**

**Copy code**

**from sklearn.tree import DecisionTreeClassifier**

**dt\_model = DecisionTreeClassifier()**

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

**y\_pred\_dt = dt\_model.predict(X\_test)**

**# Evaluate the model**

**print("Decision Tree Classification Report:\n", classification\_report(y\_test, y\_pred\_dt))**

**print("Decision Tree Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred\_dt))**

**c. Random Forest**

**python**

**Copy code**

**from sklearn.ensemble import RandomForestClassifier**

**rf\_model = RandomForestClassifier()**

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

**y\_pred\_rf = rf\_model.predict(X\_test)**

**# Evaluate the model**

**print("Random Forest Classification Report:\n", classification\_report(y\_test, y\_pred\_rf))**

**print("Random Forest Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred\_rf))**

**5. Model Evaluation**

**Compare the performance of different models using metrics such as:**

* **Precision: The proportion of true positives out of all predicted positives.**
* **Recall: The proportion of true positives out of all actual positives.**
* **F1 Score: The harmonic mean of precision and recall.**
* **Confusion Matrix: Provides insight into true positives, false positives, true negatives, and false negatives.**

**6. Hyperparameter Tuning**

**Improve model performance by tuning hyperparameters. For example, use GridSearchCV or RandomizedSearchCV for hyperparameter optimization.**

**python**

**Copy code**

**from sklearn.model\_selection import GridSearchCV**

**# Example for Random Forest**

**param\_grid = {**

**'n\_estimators': [50, 100, 200],**

**'max\_depth': [None, 10, 20, 30]**

**}**

**grid\_search = GridSearchCV(estimator=rf\_model, param\_grid=param\_grid, cv=3, scoring='f1')**

**grid\_search.fit(X\_train, y\_train)**

**print("Best parameters:", grid\_search.best\_params\_)**

**print("Best score:", grid\_search.best\_score\_)**

**7. Model Deployment**

**Once you have a well-performing model, you can deploy it for real-time transaction monitoring or integrate it into an existing system to flag fraudulent transactions.**

**Conclusion**

**This guide covers the essential steps to build a credit card fraud detection model. You can extend this approach by experimenting with additional algorithms, feature engineering, and ensemble methods to further improve model performance.**