**PHASE 3 REPORT :**

**INTRODUCTION:**

Credit card fraud detection is a critical application of machine learning and data analysis aimed at identifying and preventing unauthorized or fraudulent activities in credit card transactions. As technology has advanced, so too have the methods and techniques used to detect fraudulent activities, which can save financial institutions and individuals from significant financial losses and maintain trust in the financial system.

**DATA SET :**

**1.Import Libraries**: Start by importing the necessary Python libraries.

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.preprocessing import StandardScaler

1. **Load the Dataset**: Download the credit card transaction dataset and load it into a Pandas DataFrame.
2. You can use the **read\_csv** function to load a CSV file.

# Replace 'your\_dataset.csv' with the actual path to your dataset data = pd.re

ad\_csv('your\_dataset.csv')

1. **Explore the Data**: Before preprocessing, take a quick look at your data to understand its structure and check for any missing values.

# Display the first few rows of the dataset print(data.head())

# Check for missing values print(data.isnull().sum())

# Get summary statistics print(data.describe())

**4.Preprocess the Data**: Preprocessing is a crucial step in fraud detection to ensure that the data is ready for analysis. Common preprocessing steps include:

5. **Handling Imbalanced Data**: Check the balance between fraud and non-fraud transactions. If imbalanced, consider oversampling, undersampling, or using advanced techniques like Synthetic Minority Over-sampling Technique (SMOTE).

6.. **Feature Scaling**: Scale the features to have a mean of 0 and a standard deviation of 1. This is important for algorithms like logistic regression or k-nearest neighbors.

# Separate features and target variable

X = data.drop('Class', axis=1)

y = data['Class']

# Split the dataset into training and testing sets

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

# Standardize the data scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train) X\_test = scaler.transform(X\_test)

**7.Explore the Preprocessed Data**: After preprocessing, explore the data again to ensure everything looks good.

# Check the class balance print(y\_train.value\_counts())

# Visualize the data (e.g., his

tograms, scatter plots)

# You can use libraries like Matplotlib and Seaborn for this.

**8.Save Preprocessed Data**: If you want to save the preprocessed data for future use, you can use Pandas' **to\_csv** function.

X\_train\_df = pd.DataFrame(X\_train, columns=X.columns) X\_test\_df = pd.DataFrame(X\_test, columns=X.columns) X\_train\_df.to\_csv('preprocessed\_train\_data.csv', index=False) X\_test\_df.to\_csv('preprocessed\_test\_data.csv', index=False)

CODE :

# Import necessary libraries

import pandas as pd

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

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report, roc\_auc\_score

# Step 1: Load the dataset

# Replace 'your\_dataset.csv' with the actual path to your dataset

data = pd.read\_csv('your\_dataset.csv')

# Step 2: Data Preprocessing

X = data.drop('Class', axis=1)

y = data['Class']

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

# Step 3: Model Selection and Training

model = RandomForestClassifier()

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

# Step 4: Model Evaluation

y\_pred = model.predict(X\_test)

print(classification\_report(y\_test, y\_pred))

roc\_auc = roc\_auc\_score(y\_test, y\_pred)

print("ROC AUC:", roc\_auc)

CODE EXPLANATION :

# Import necessary libraries import pandas as pd from sklearn.model\_selection import train\_test\_split from sklearn.ensemble import RandomForestClassifier from sklearn.metrics import classification\_report, roc\_auc\_score

In this section, we import the required libraries:

* **pandas** is used for data manipulation and handling DataFrames.
* **train\_test\_split** from **sklearn.model\_selection** is used to split the dataset into a training set and a test set.
* **RandomForestClassifier** from **sklearn.ensemble** is a machine learning model that will be used for fraud detection.
* **classification\_report** and **roc\_auc\_score** from **sklearn.metrics** are used to evaluate the model's performance.

# Step 1: Load the dataset data = pd.read\_csv('your\_dataset.csv')

Here, we load the credit card transaction dataset from a CSV file into a Pandas DataFrame. Replace **'your\_dataset.csv'** with the actual path to your dataset.

# Step 2: Data Preprocessing X = data.drop('Class', axis=1) y = data['Class'] X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

In this section, we perform data preprocessing:

* We separate the dataset into the feature matrix **X** (containing all columns except 'Class') and the target variable **y** (which is the 'Class' column, indicating whether a transaction is fraudulent or not).
* We split the data into a training set (**X\_train** and **y\_train**) and a test set (**X\_test** and **y\_test**) using **train\_test\_split**. The test set is 20% of the data, and we set a random seed (**random\_state**) for reproducibility.

# Step 3: Model Selection and Training model = RandomForestClassifier() model.fit(X\_train, y\_train)

In this section, we select a machine learning model for fraud detection. We chose a **RandomForestClassifier**, which is an ensemble learning method based on decision trees. We then fit (train) the model using the training data.

# Step 4: Model Evaluation y\_pred = model.predict(X\_test) print(classification\_report(y\_test, y\_pred)) roc\_auc = roc\_auc\_score(y\_test, y\_pred) print("ROC AUC:", roc\_auc)

In this final part, we evaluate the performance of the trained model:

* We make predictions on the test set using **model.predict(X\_test)**.
* We print a classification report, which provides metrics such as precision, recall, F1-score, and support for both classes (fraudulent and non-fraudulent).
* We calculate the ROC AUC score, which is a measure of how well the model distinguishes between the two classes. A higher ROC AUC indicates better performance.

CONCLUSION :

The provided Python code outlines a basic framework for building a credit card fraud detection model using a Random Forest Classifier. Here's a conclusion summarizing the key points:

1. **Data Preprocessing**: The code starts by loading the credit card transaction dataset from a CSV file and splitting it into feature variables (**X**) and the target variable (**y**). Data preprocessing is a critical step in machine learning, including dealing with data imbalance and scaling features.
2. **Model Selection and Training**: A Random Forest Classifier, a popular ensemble learning algorithm, is chosen as the machine learning model. The model is trained using the training dataset (**X\_train** and **y\_train**).
3. **Model Evaluation**: The code evaluates the model's performance on the test dataset (**X\_test** and **y\_test**). It prints a classification report, which provides insights into the model's precision, recall, F1-score, and support for both classes (fraudulent and non-fraudulent). Additionally, it calculates the ROC AUC score, which measures the model's ability to distinguish between the two classes.
4. **Limitations**: The code presented is a simplified example and does not include advanced techniques often used in credit card fraud detection, such as feature engineering, hyperparameter tuning, handling of imbalanced data, and real-time monitoring.
5. **Deployment and Continuous Monitoring**: In a real-world scenario, deploying the model for real-time or batch processing of credit card transactions and continuous monitoring are critical steps for maintaining the effectiveness of the fraud detection system.
6. **Security and Privacy**: Handling credit card transaction data involves sensitive information. Security and privacy considerations are essential, and models should be developed and deployed following best practices to protect this information.

In practice, credit card fraud detection is a complex and evolving field, requiring advanced techniques, large datasets, and continuous updates to stay ahead of fraudulent activities. The code presented here serves as a foundational framework, and real-world applications would involve more sophisticated approaches and stringent security measures.

The provided Python code outlines a basic framework for building a credit card fraud detection model using a Random Forest Classifier. Here's a conclusion summarizing the key points: