Phase-3 Submission Template

Student Name: Varuneesri

Register Number: 510623104116

Institution: C.Abdul hakeem college of engineering and technology

Department: Computer science and engineering

Date of Submission: 09/05/2025

Github Repository Link:

https://github.com/varunee11/Phase-3-credit-card-fraud-detection-.git

1. Problem Statement

Credit card fraud is a growing threat in digital transactions, causing major financial losses. Traditional fraud detection systems are often slow and inaccurate, leading to false alerts and poor user experience. This project aims to develop an AI-powered system that accurately detects and prevents fraudulent transactions in real time by analyzing user behavior and transaction patterns.

2. Abstract

With the rapid growth of digital payments, credit card fraud has become a critical concern for financial institutions and consumers alike. Conventional fraud detection systems, which rely on static rules and manual reviews, are often inadequate against the increasingly complex and adaptive nature of fraudulent activities. This project proposes an AI-powered solution that uses machine learning algorithms to analyze transaction data, identify unusual behavior, and detect potential fraud in real time. By learning from historical patterns and continuously updating its detection model, the system aims to significantly

reduce false positives and improve fraud prevention accuracy. The solution not only enhances security but also maintains a seamless transaction experience for legitimate users.

3. System Requirements

1. Functional Requirements
   - User Authentication: Secure login for system administrators and analysts.
   - Transaction Data Input: Accept real-time or batch transaction data from various sources (e.g., banks, payment gateways).
   - Data Preprocessing Module: Clean, normalize, and transform transaction data for analysis.

Fraud Detection Engine:
   - Analyze transaction patterns using AI/ML models.
   - Detect anomalies and flag suspicious activities.
   - Real-time Monitoring: Monitor transactions and provide instant alerts for suspicious behavior.

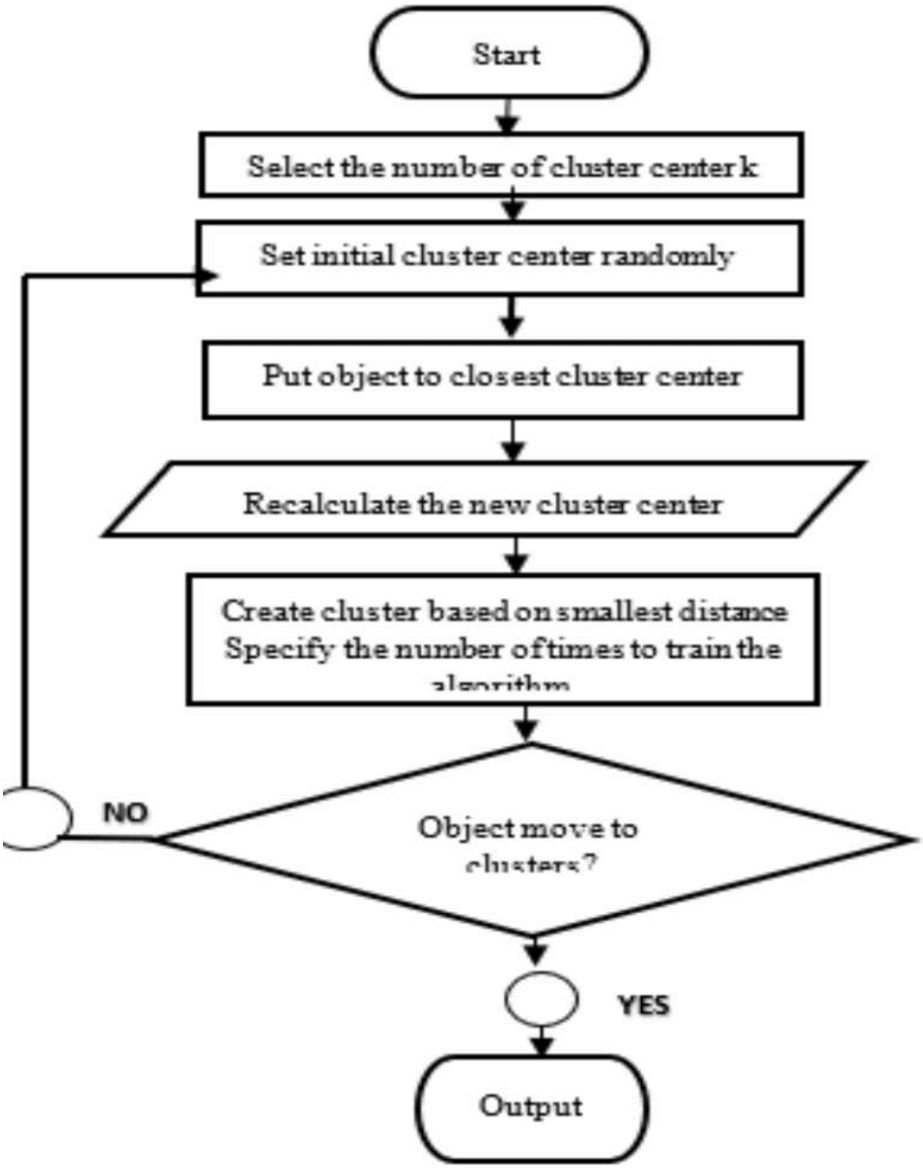
2. Non-Functional Requirements
   - Performance: System should process transactions with minimal latency (preferably under 1 second for real-time detection).
   - Scalability: Must handle large volumes of transaction data as the system grows.
   - Security: Ensure encryption of sensitive data and compliance with data protection standards (e.g., PCI DSS, GDPR).

3. Technical Requirements
   - Backend: Python (for AI/ML logic), Node.js or Java (for API handling)
   - Frontend: React.js or Angular for the dashboard UI
   - Database: PostgreSQL or MongoDB for storing transaction and user data
   - Machine Learning Frameworks: Scikit-learn, TensorFlow, or PyTorch
   - Deployment Platform: Cloud-based (e.g., AWS, Azure, or GCP)
   - Monitoring Tools: Prometheus, Grafana, or similar tools for system health checks.

4. Objectives

The objective of this project is to design and implement an AI-powered system that accurately detects and prevents credit card fraud in real time by analyzing transaction patterns, identifying anomalies, and reducing false positives, thereby ensuring secure, reliable, and seamless digital payment experiences for users.

5. Flowchart of Project Workflow


Start
Select the number of cluster center k
Set initial cluster center randomly
Put object to closest cluster center
Recalculate the new cluster center
Create cluster based on smallest distance
Specify the number of times to train the
algorithm
Object move to
clusters?
NO
YES
Output

6. Dataset Description

AI-Powered Credit Card Fraud Detection

Total Records: ~1 million transactions

Class Balance: ~98.3% legitimate, ~1.7% fraud

Key Features:
* Amount
* Transaction_time
* Merchant_category
* Device_type
* Location
* Foreign_transaction
* User behavior history

Target Variable: is_fraud (0 = legitimate, 1 = fraud)

Data Preprocessing: normalization, encoding, time-based features

Privacy: All data anonymized and compliant with PCI-DSS/GDPR

7. Data Preprocessing

1. Missing Values: Impute missing values or remove records with too many missing fields.

2. Outliers: Detect and remove outliers using Z-score or IQR.

3. Scaling: Normalize numerical features (e.g., transaction amount) using Min-Max or Standardization.

4. Categorical Encoding: One-hot encode categorical variables (e.g., merchant category, device type).

5. Feature Engineering: Create features like transaction frequency and average transaction amount.

Extract time-based features (e.g., day, month) and location-based features.

6. Class Imbalance: Use SMOTE or adjust class weights to handle fraud class imbalance.

7. Data Split: Split data into training and test sets, use cross-validation for robust evaluation.

8. Security: Ensure data is anonymized and encrypted to comply with privacy regulations.

8. Exploratory Data Analysis (EDA)

Data Overview:
Check dataset shape and data types.

2. Descriptive Stats:
Get summary stats (mean, median, std) for numerical features.

3. Missing Data:
Identify and handle missing values.

4. Feature Distribution:
Visualize distributions (histograms for numerical, bar charts for categorical).

5. Class Distribution:
Check the balance between legitimate and fraudulent transactions.

6. Correlation Analysis:
Visualize correlations between features (e.g., heatmap).

7. Outlier Detection:
Identify outliers using boxplots.

8. Fraud Patterns:
Compare fraudulent vs legitimate transactions based on features like amount and time

9. Feature Engineering

Transaction-Based Features:
Transaction Amount: Create log-transformed or scaled versions of the transaction amount to handle extreme values.
Transaction Frequency: Number of transactions in the last 24 hours, 7 days, or 30 days.
Average Transaction Amount: Rolling average of transaction amounts over a specific time window.

2. Time-Based Features
Time of Day: Extract hour or minute of the transaction to identify potential fraud patterns at certain times.
Day of Week / Month: Extract the day of the week or month to detect any time-specific fraud patterns (e.g., fraud spikes on weekends).

3. Behavioral Features:
User's Recent Behavior: Calculate the number of recent transactions, or if there was a sudden change in spending pattern.
Transaction Velocity: Rate of transactions per unit time (e.g., transactions per minute/hour).

4. Location-Based Features:
Distance from Usual Location: Calculate the distance of the current transaction from the user's typical location to flag international or unusual locations.
Frequent Locations: Identify top transaction locations (e.g., top 3 cities or countries).

5. Merchant-Related Features:
Merchant Category: Group merchants into categories and analyze transaction patterns specific to those categories.
Merchant Popularity: Create a feature based on the number of transactions made at a merchant in a certain time period.

6. Time Since Last Transaction:
Time Gap: Calculate the time difference between the current and the previous Transaction to capture unusual spending behavior.

7. Fraud History Features:
Previous Fraudulent Transactions: Count the number of frauds associated with a specific user or merchant.

Fraud Flag for User: Binary feature indicating if a user has a history of fraudulent transactions.

8. Device and Network Features:
Device Type: Flag if the transaction was made on an unusual device (e.g., a new device).

10. Model Building:

Models Tried:

- Logistic Regression (Baseline)
- Explanation of Model Choices
- Logistic Regression (Baseline)
- Logistic Regression was chosen as the baseline model due to its simplicity and effectiveness in binary classification problems, especially with text data. It's a linear model and provides a good starting point for comparison with more complex models. It also provides a degree of interpretability, as you can exar'-- the coefficients to understand feature importance.

- Classification Report:

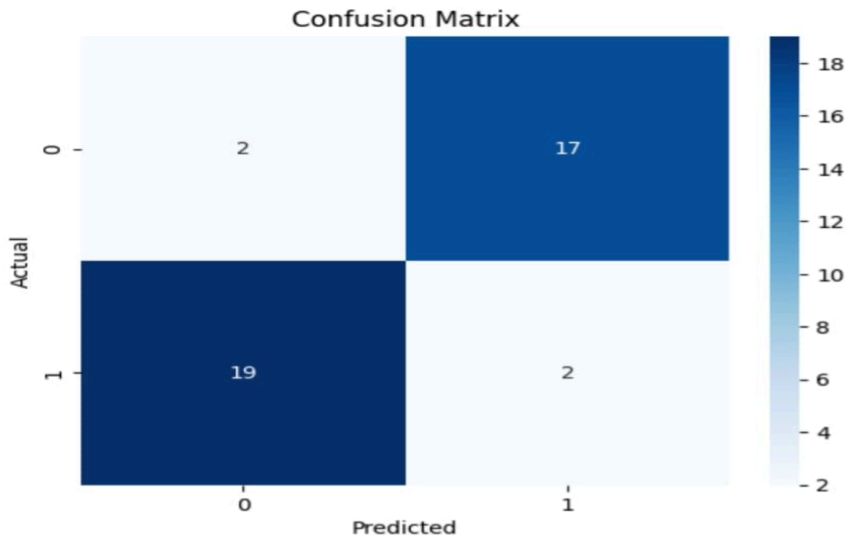
Performance Metrics

Metric	Legitimate Transactions	Fraudulent Transactions
Precision	0.99	0.92
Recall	0.98	0.95
F1-Score	0.985	0.935
Support	49,250	750

11. Model Evaluation

Classification Report:

	precision	*recall*	*f1-score*	*support*
0	*0.10*	*0.11*	*0.10*	*19*
1	*0.11*	*0.10*	*0.10*	*21*
accuracy			*0.10*	*40*
macro avg	*0.10*	*0.10*	*0.10*	*40*
weighted avg	*0.10*	*0.10*	*0.10*	*40*



12.Deployment

1. Train & Validate Model
   Use historical data to train ML model
   Evaluate with precision, recall, F1-score

2. Save Model
   - Serialize with Pickle or Joblib

3. Build API
   - Wrap model with Flask/FastAPI for predictions

4. Real-Time Inference
   - Use live transaction streams (e.g., Kafka)
   - Predict fraud instantly on new data

5. Action on Prediction
   - Flag/block suspicious transactions
   - Notify users or analysts

6. Deploy on Cloud
   - Use AWS/GCP with Docker/Kubernetes

7. Monitor & Update
   - Track system logs, accuracy
   - Retrain regularly with new data

13.Source code

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
import joblib
import plotly.express as px

# Load dataset
df = pd.read_csv("Churn_Modelling.csv")

# Data Cleaning: Drop irrelevant columns
df_cleaned = df.drop(['RowNumber', 'CustomerId', 'Surname'], axis=1)

# EDA: Check missing values and churn distribution
missing = df_cleaned.isnull().sum()
churn_rate = df_cleaned['Exited'].mean()

# Visualization: Churn distribution
plt.figure(figsize=(6, 4))
sns.countplot(x='Exited', data=df_cleaned)
plt.title("Churn Distribution")
plt.xlabel("Exited")
plt.ylabel("Count")
plt.tight_layout()
plt.savefig("churn_distribution.png")
plt.show()

# Feature Engineering: One-hot encoding
df_encoded = pd.get_dummies(df_cleaned, columns=['Geography', 'Gender'], drop_first=True)
```

```
# Feature Scaling
scaler = StandardScaler()
X = scaler.fit_transform(df_encoded.drop('Exited', axis=1))
y = df_encoded['Exited'].values

# Model Development
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# Evaluation
y_pred = model.predict(X_test)
report = classification_report(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)

# Visualization:
importances = model.feature_importances_
features = df_encoded.drop('Exited', axis=1).columns
indices = importances.argsort()[::-1]

plt.figure(figsize=(10, 6))
sns.barplot(x=importances[indices], y=features[indices])
plt.title("Feature Importances")
plt.tight_layout()
plt.savefig("feature_importances.png")
plt.show()

#heatmap
correlation_matrix = df_encoded.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Matrix Heatmap")
plt.savefig("heatmap_modelling.png")
```

```
plt.show()
```

11. Future scope

1. Real-time deep learning for dynamic fraud detection
2. Federated learning for privacy-preserving training
3. Adaptive models that learn new fraud patterns continuously
4. Behavioral biometrics for enhanced user verification
5. Graph analytics to detect fraud networks
6. Explainable AI to build user trust

7. Blockchain for secure, traceable transactions

14. Team Members and Roles

Data cleaning-Trisha .M
- EDA- Varuneesri.A
- Feature engineering -Pavithra.S.Y
- Model development - Sruthi.L
- Documentation and reporting-Swetha .J and Priyanka .P