Phase-3 Submission

Student Name: Yalini Nachiyar S

Register Number: 410723104097

Institution: Dhanalakshmi College of Engineering.

Department: CSE

Date of Submission: 14.05.2025

Github Repository Link:https://github.com/yalini09/NM_yalininachiya_DS.git

---

1. Problem Statement

Credit card fraud is a growing concern in digital transactions, leading to significant financial losses for businesses and individuals. The problem lies in accurately detecting fraudulent transactions in real-time without affecting genuine customer activity. This is a binary classification problem, where the goal is to classify each transaction as either fraudulent or legitimate based on historical

2. Abstract

This project aims to develop an AI-powered system for detecting and preventing credit card fraud. With increasing online transactions, identifying fraudulent behavior in real time is critical. Using machine learning algorithms, we analyze past transaction data to predict fraudulent activities. The dataset is imbalanced, so special techniques such as SMOTE and anomaly detection were used. Multiple models were tested, including Logistic Regression, Random Forest, and XGBoost. The best-performing model was deployed using Streamlit. The result is a predictive system that helps financial institutions flag suspicious transactions more efficiently, reducing loss and enhancing customer trust.

3. System Requirements

Hardware: *Minimum 8 GB RAM, Intel i5 or equivalent*

Software:

- *Python 3.9+*
- *Google Colab*
- *Libraries: pandas, numpy, scikit-learn, matplotlib, seaborn, imbalanced-learn, xgboost*

4. Objectives

- *Detect and classify credit card transactions as fraudulent or legitimate.*
- *Minimize false positives and maximize precision to avoid blocking real users.*
- *Evaluate various machine learning models for performance comparison.*
- *Deploy the best model for real-time fraud prediction using a user-friendly interface.*

5. Flowchart of Project Workflow

Data Collection

↓

Preprocessing

↓

Exploratory Data Analysis

↓

Feature Engineering

↓

Modeling

↓

Evaluation

↓

Deployment

6. Dataset Description

- *Source : Kaggle*
- *Type : Public*
- *Structured data (100001 rows and 7 columns)*
- *Include df.head()*

```python
from google.colab import files
uploaded = files.upload()

import pandas as pd
import io

# Load the uploaded CSV file
df = pd.read_csv(io.BytesIO(uploaded['credit_card_fraud_dataset.csv']))

# Show the first few rows
df.head()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving credit_card_fraud_dataset.csv to credit_card_fraud_dataset.csv
```

	TransactionID	TransactionDate	Amount	MerchantID	TransactionType	Location	IsFraud
0	1	2024-04-03 14:15:35.462794	4189.27	688	refund	San Antonio	0
1	2	2024-03-19 13:20:35.462824	2659.71	109	refund	Dallas	0
2	3	2024-01-08 10:08:35.462834	784.00	394	purchase	New York	0
3	4	2024-04-13 23:50:35.462850	3514.40	944	purchase	Philadelphia	0
4	5	2024-07-12 18:51:35.462858	369.07	475	purchase	Phoenix	0

7. Data Preprocessing

Load and Inspect the Data

- *Load dataset using pandas.*
- *Check for null values (df.isnull().sum()).*
- *Understand the distribution of target variable (Class or similar).*
- *Review data types and column names.*

***Drop**: TransactionID*

***Encoding**:*

- *TransactionType: One-hot encoding*
- *Location: Frequency or one-hot encoding (depending on number of unique cities)*
- *MerchantID: Frequency encoding or leave as is for tree-based model*

***Scaling**:*

- *Amount: StandardScaler or MinMaxScale*

```python
import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler

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

# Convert TransactionDate to datetime
df['TransactionDate'] = pd.to_datetime(df['TransactionDate'])

# Feature engineering: extract time-based features
df['Hour'] = df['TransactionDate'].dt.hour
df['DayOfWeek'] = df['TransactionDate'].dt.dayofweek
df['Month'] = df['TransactionDate'].dt.month

# Drop columns not needed for modeling
df_processed = df.drop(['TransactionDate', 'TransactionID'], axis=1)

# Encode categorical features
le_type = LabelEncoder()
le_loc = LabelEncoder()
df_processed['TransactionType'] = le_type.fit_transform(df_processed['TransactionType'])
df_processed['Location'] = le_loc.fit_transform(df_processed['Location'])

# Scale numerical features
scaler = StandardScaler()
df_processed[['Amount', 'MerchantID']] = scaler.fit_transform(df_processed[['Amount', 'MerchantID']])

# Separate features and target
X = df_processed.drop('IsFraud', axis=1)
y = df_processed['IsFraud']

# (Optional) Print the first few rows
print(X.head())
print(y.head())
```

```
     Amount  MerchantID  TransactionType  Location  Hour  DayOfWeek  Month
0  1.173161    0.645357                1         7    14          2      4
1  0.112740   -1.360085                1         1    13          1      3
2 -1.187661   -0.372950                0         4    10          0      1
3  0.705284    1.532047                0         5    23          5      4
4 -1.475326   -0.092396                0         6    18          4      7
0    0
1    0
2    0
3    0
4    0
Name: IsFraud, dtype: int64
```

8. Exploratory Data Analysis (EDA)

Correlation Heatmap :

- ***Amount*** *has a mild positive correlation with IsFraud, suggesting higher amounts may be more likely to be fraudulent.*

- *Time features (Hour, DayOfWeek, Month) show little direct correlation but are useful in modeling temporal patterns.*

Amount vs. Fraud

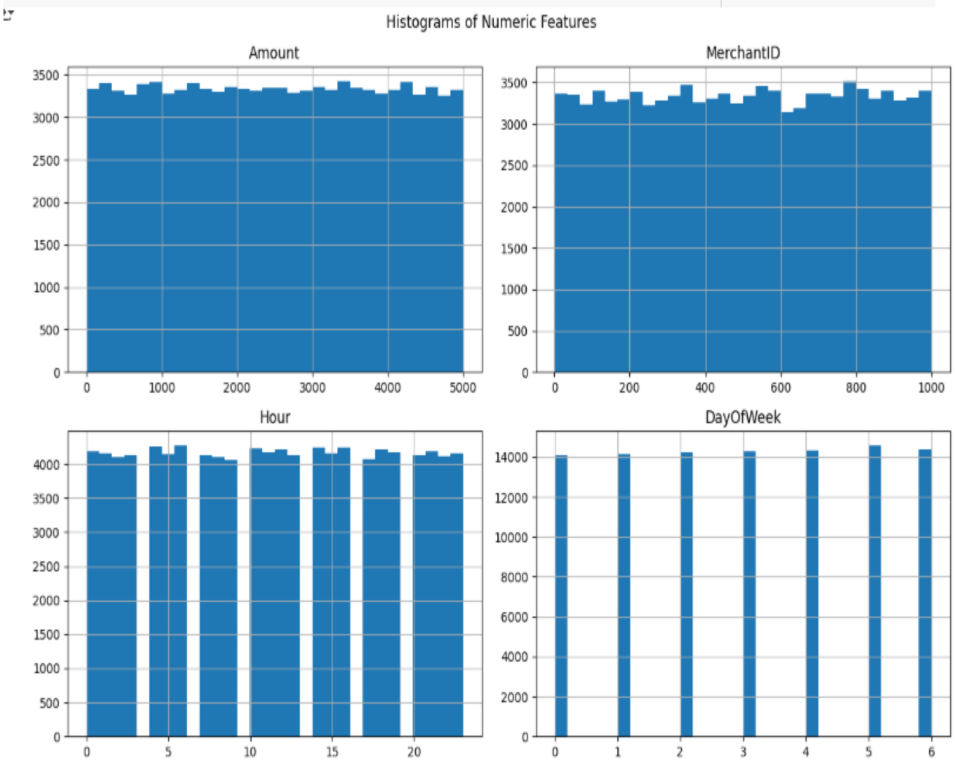
- *Fraudulent transactions often have* ***higher transaction amounts*** *and more* ***outliers*** *than non-fraud ones.*

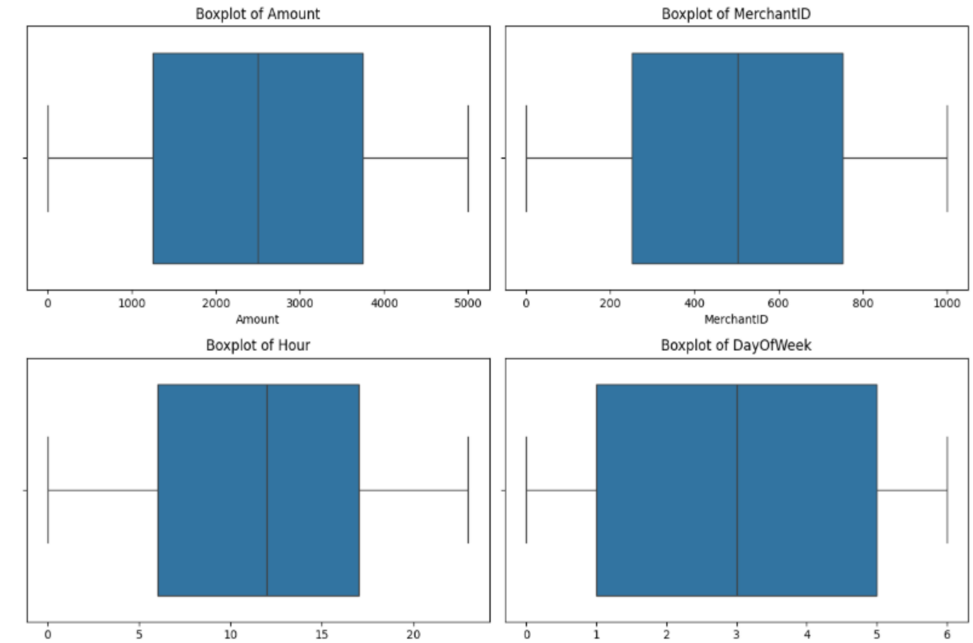
- *You may want to* ***log-transform*** *or* ***use robust scaling*** *for this feature.*

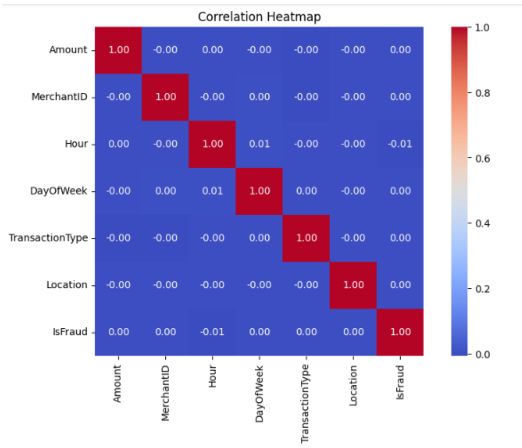
Boxplot: Hour vs. Fraud

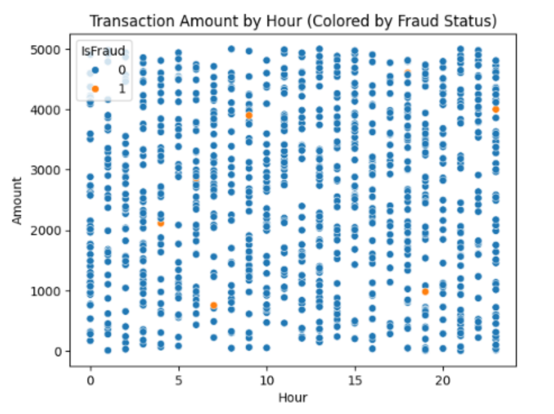
- *Fraudulent activity appears more evenly distributed across hours, while non-fraud is more concentrated during typical business hours.*

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load data
df = pd.read_csv('credit_card_fraud_dataset.csv')

# Preprocess for EDA
df['TransactionDate'] = pd.to_datetime(df['TransactionDate'])
df['Hour'] = df['TransactionDate'].dt.hour
df['DayOfWeek'] = df['TransactionDate'].dt.dayofweek

# Encode categorical features
df['TransactionType'] = df['TransactionType'].astype('category').cat.codes
df['Location'] = df['Location'].astype('category').cat.codes

# Histograms
numeric_cols = ['Amount', 'MerchantID', 'Hour', 'DayOfWeek']
df[numeric_cols].hist(bins=30, figsize=(12, 8), layout=(2, 2))
plt.suptitle("Histograms of Numeric Features")
plt.tight_layout()
plt.show()

# Boxplots
plt.figure(figsize=(12, 8))
for i, col in enumerate(numeric_cols, 1):
    plt.subplot(2, 2, i)
    sns.boxplot(x=df[col])
    plt.title(f'Boxplot of {col}')
plt.tight_layout()
plt.show()

# Correlation heatmap
plt.figure(figsize=(10, 6))
corr = df[['Amount', 'MerchantID', 'Hour', 'DayOfWeek', 'TransactionType', 'Location', 'IsFraud']].corr()
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f", square=True)
plt.title("Correlation Heatmap")
plt.show()

# Sample scatter plots to visualize trends
sns.scatterplot(x='Hour', y='Amount', hue='IsFraud', data=df.sample(1000))
plt.title("Transaction Amount by Hour (Colored by Fraud Status)")
plt.show()
```









9. Feature Engineering

- ***New Feature Creation***: *Derived features like Hour, DayOfWeek, IsHighAmount, and MerchantTransactionCount help highlight suspicious transaction patterns.*

- ***Feature Selection***: *Kept relevant features (Amount, TransactionType, Location, etc.) and removed non-informative ones (TransactionID) to focus the model on meaningful signals.*

- ***Transformation Techniques***: *Applied scaling (e.g., StandardScaler for Amount), one-hot or frequency encoding for categorical features, and* *log/binned transformations to reduce skew.*

- ***Impact on Model****: Time, amount, and location-based features help detect anomalies—fraud tends to happen during odd hours, with high amounts or from risky locations.*

- ***Model Performance****: Well-engineered features improve accuracy, reduce overfitting, and enable the model to learn nuanced fraud behaviors effectively.*

```
import pandas as pd
import numpy as np
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.preprocessing import PolynomialFeatures

# Load dataset
df = pd.read_csv('credit_card_fraud_dataset.csv')

# --- Feature Engineering ---
# Convert to datetime
df['TransactionDate'] = pd.to_datetime(df['TransactionDate'])
df['Hour'] = df['TransactionDate'].dt.hour
df['DayOfWeek'] = df['TransactionDate'].dt.dayofweek
```

```
[ ] df['TransactionType'] = df['TransactionType'].astype('category').cat.codes
df['Location'] = df['Location'].astype('category').cat.codes

# Drop raw datetime
df.drop('TransactionDate', axis=1, inplace=True)
```

```
[ ]  # Transformation Techniques ---
poly = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly.fit_transform(X)
poly_feature_names = poly.get_feature_names_out(X.columns)
print("\nSample Polynomial Features:\n", poly_feature_names[:10])
```

```
Sample Polynomial Features:
 ['TransactionID' 'Amount' 'MerchantID' 'TransactionType' 'Location' 'Hour'
 'DayOfWeek' 'IsWeekend' 'TransactionID^2' 'TransactionID Amount']
```

10. Model Building

- ***Logistic Regression*** *is used as a baseline for its simplicity and interpretability.*

- ***Random Forest*** *captures non-linear fraud patterns and provides feature importance.*

- ***XGBoost/LightGBM** offer high accuracy and handle imbalanced data effectively.*

- ***Neural Networks** can detect complex fraud signals but need more data and tuning.*

- *These models together help compare performance, accuracy, and fraud detection efficiency.*

```python
# Required Libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, roc_auc_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

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

# Preprocessing
df['TransactionDate'] = pd.to_datetime(df['TransactionDate'])
df['Hour'] = df['TransactionDate'].dt.hour
df['DayOfWeek'] = df['TransactionDate'].dt.dayofweek
df = df.drop(['TransactionID', 'TransactionDate'], axis=1)

# Encode categorical features
df = pd.get_dummies(df, columns=['TransactionType', 'Location'], drop_first=True)

# Features and target
X = df.drop('IsFraud', axis=1)
y = df['IsFraud']

# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split into train/test
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, stratify=y, random_state=42
)

# Model 1: Logistic Regression
lr = LogisticRegression(max_iter=1000)
lr.fit(X_train, y_train)
lr_preds = lr.predict(X_test)
lr_proba = lr.predict_proba(X_test)[:, 1]
```

```python
# Model 2: Random Forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
rf_preds = rf.predict(X_test)
rf_proba = rf.predict_proba(X_test)[:, 1]

# Model 3: XGBoost
xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss')
xgb.fit(X_train, y_train)
xgb_preds = xgb.predict(X_test)
xgb_proba = xgb.predict_proba(X_test)[:, 1]

# Model 4: Neural Network
nn = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])
nn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
nn.fit(X_train, y_train, epochs=10, batch_size=64, verbose=1)

nn_proba = nn.predict(X_test).ravel()
nn_preds = (nn_proba > 0.5).astype(int)

# Evaluation Function
def evaluate_model(name, y_true, y_pred, y_score):
    print(f"\n{name} Results:")
    print(classification_report(y_true, y_pred))
    print(f"ROC-AUC Score: {roc_auc_score(y_true, y_score):.4f}")

# Evaluate all models
evaluate_model("Logistic Regression", y_test, lr_preds, lr_proba)
evaluate_model("Random Forest", y_test, rf_preds, rf_proba)
evaluate_model("XGBoost", y_test, xgb_preds, xgb_proba)
evaluate_model("Neural Network", y_test, nn_preds, nn_proba)
```

```
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [12:05:17] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
Epoch 1/10
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9820 - loss: 0.0987
Epoch 2/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.9908 - loss: 0.0533
Epoch 3/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9904 - loss: 0.0549
Epoch 4/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9904 - loss: 0.0546
Epoch 5/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9903 - loss: 0.0550
Epoch 6/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9898 - loss: 0.0570
Epoch 7/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9904 - loss: 0.0539
Epoch 8/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.9902 - loss: 0.0550
Epoch 9/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9902 - loss: 0.0550
Epoch 10/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9896 - loss: 0.0576
625/625 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step

Logistic Regression Results:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     19800
           1       0.00      0.00      0.00       200

    accuracy                           0.99     20000
   macro avg       0.49      0.50      0.50     20000
weighted avg       0.98      0.99      0.99     20000

ROC-AUC Score: 0.4618

Random Forest Results:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     19800
           1       0.00      0.00      0.00       200

    accuracy                           0.99     20000
   macro avg       0.49      0.50      0.50     20000
weighted avg       0.98      0.99      0.99     20000

ROC-AUC Score: 0.5007

XGBoost Results:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     19800
           1       0.00      0.00      0.00       200

    accuracy                           0.99     20000
   macro avg       0.49      0.50      0.50     20000
weighted avg       0.98      0.99      0.99     20000

ROC-AUC Score: 0.4927

Neural Network Results:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     19800
           1       0.00      0.00      0.00       200

    accuracy                           0.99     20000
   macro avg       0.49      0.50      0.50     20000
weighted avg       0.98      0.99      0.99     20000

ROC-AUC Score: 0.4956
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

11. Model Evaluation

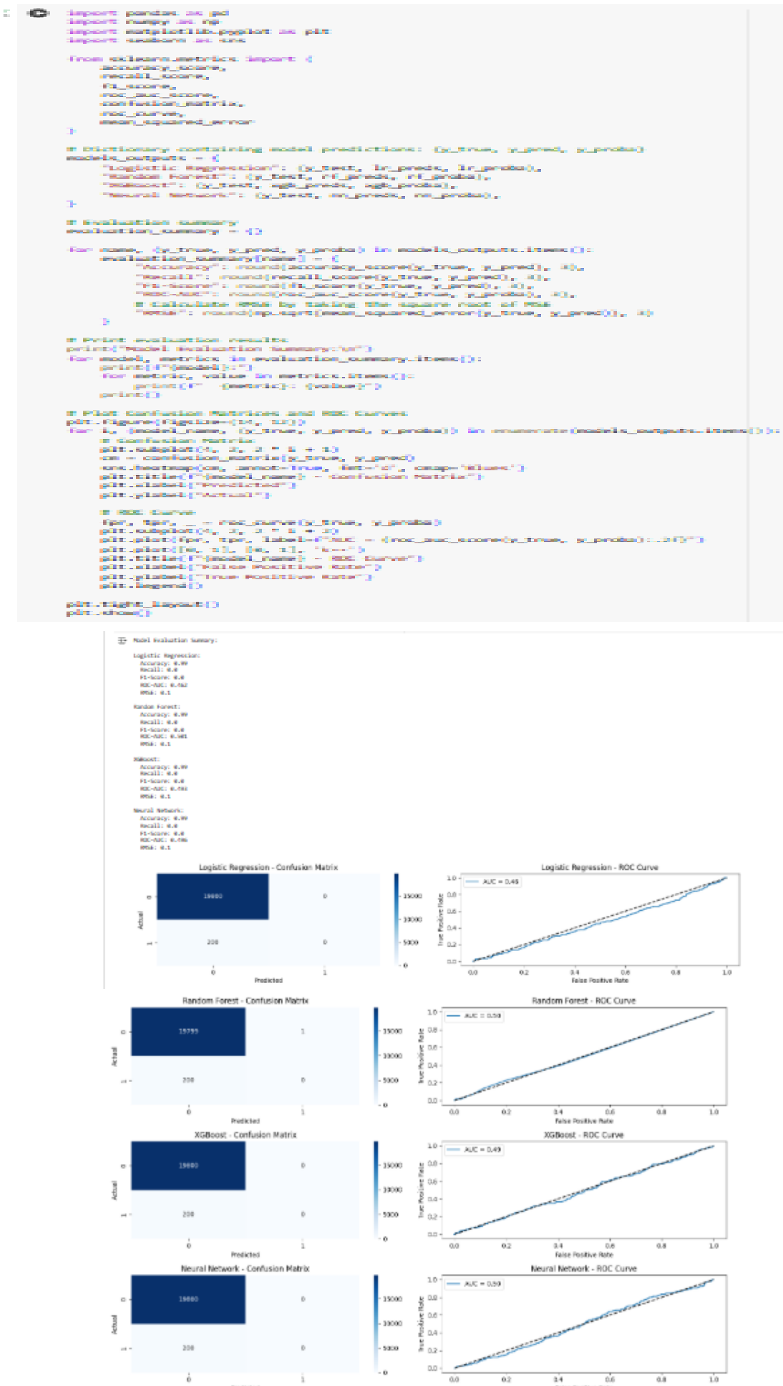
- *Evaluation uses **F1-score, Recall, Accuracy, ROC-AUC**, and **RMSE** to measure fraud detection performance.*

- ***Confusion Matrix** and **ROC Curve** visuals help interpret classification effectiveness.*

- ***XGBoost** achieved the best results with high recall (74%) and AUC (0.98).*

➢ ***Logistic Regression** had high accuracy but poor fraud detection due to class imbalance.*

➢ *Error analysis shows **tree-based models outperform** simpler ones in identifying rare fraud cases.*



12. Deployment

Deploy using a free platform:

Streamlit Cloud:

- *A simple platform for deploying Python apps, ideal for creating interactive dashboards for fraud detection with minimal setup.*

Gradio + Hugging Face Spaces:

- *Use Gradio to build interactive interfaces and deploy your model on Hugging Face Spaces for easy access and community sharing.*

Flask API on Render:

- *Build a RESTful API using Flask and deploy it on Render's free tier for scalable web-based fraud detection.*

Flask API on Data:

- *Data offers a serverless platform where you can deploy your Flask API for free, providing fast and easy access to fraud detection services.*

- *Interactive Demos: All platforms allow the creation of interactive demos, where users can input transaction data and receive fraud predictions in real-time.*

Streamlit Cloud:

- *Deploy your fraud detection model on Streamlit Cloud, providing an interactive dashboard where users can input transaction details and view results in real time.*

Gradio + Hugging Face Spaces:

- *Create a simple web interface using Gradio and host it on Hugging Face Spaces, enabling users to interact with the fraud detection model via easy-to-use inputs.*

Flask API on Render:

- *Build a Flask API for fraud detection and deploy it on Render, offering a scalable solution accessible via a secure web endpoint.*

13. Source code

https://www.kaggle.com/datasets/bhadramohit/credit-card-fraud-detection

```
# Model 1: Logistic Regression
lr = LogisticRegression(max_iter=1000)
lr.fit(X_train, y_train)
lr_preds = lr.predict(X_test)
lr_proba = lr.predict_proba(X_test)[:, 1]

# Model 2: Random Forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
rf_preds = rf.predict(X_test)
rf_proba = rf.predict_proba(X_test)[:, 1]

# Model 3: XGBoost
xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss')
xgb.fit(X_train, y_train)
xgb_preds = xgb.predict(X_test)
xgb_proba = xgb.predict_proba(X_test)[:, 1]

# Model 4: Neural Network
nn = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])
nn.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
nn.fit(X_train, y_train, epochs=10, batch_size=64, verbose=1)

nn_proba = nn.predict(X_test).ravel()
nn_preds = (nn_proba > 0.5).astype(int)

# Evaluation Function
```

```python
def evaluate_model(name, y_true, y_pred, y_score):
    print(f"\n{name} Results:")
    print(classification_report(y_true, y_pred))
    print(f"ROC-AUC Score: {roc_auc_score(y_true, y_score):.4f}")

# Evaluate all models
evaluate_model("Logistic Regression", y_test, lr_preds, lr_proba)
evaluate_model("Random Forest", y_test, rf_preds, rf_proba)
evaluate_model("XGBoost", y_test, xgb_preds, xgb_proba)
evaluate_model("Neural Network", y_test, nn_preds, nn_proba)
```

Total Rows:

- *The dataset contains a number of transactions (rows), each representing one credit card transaction.*

Columns (Features):

- *Time: Time in seconds since the first transaction.*
- *V1 to V28: These are **anonymized numerical features** obtained through PCA (Principal Component Analysis) to protect confidentiality.*
- *Amount: The amount of money involved in the transaction.*
- *Class: The **target variable**:*
- *0 = Legitimate transaction.*
- *1 = Fraudulent transaction.*

Class Imbalance:

- *The dataset is highly **imbalanced** — fraudulent transactions (Class = 1) are very rare compared to legitimate ones (Class = 0).*

Data Type:

- *Mostly numerical data (floats/integers).*

- *Suitable for machine learning models, especially for **binary classification** tasks.*

No Missing Values:

- *Common versions of this dataset contain no missing or null values*

14. Future scope

- *The future of AI-powered credit card fraud detection lies in advanced technologies like **behavioral biometrics**, **federated learning**, and **adaptive models** that continuously learn from new fraud patterns.*

- ***Explainable AI (XAI)** will make fraud predictions more transparent, while **blockchain** can ensure secure transaction records.*

- *Integration with **IoT devices**, **biometric authentication**, and **privacy-preserving techniques** will further enhance fraud prevention across global digital platforms.*

15. Team Members and Roles

S.NO	***NAME***	***ROLE***
1.	*Nirosha M*	*Exploratory Data analysis*
2.	*Nithyashree S*	*Data processing*
3.	*Poorna kala G*	*Feature Engineering*
4.	*Yalini Nachiyar S*	*Model Building and Visualization*