from google.colab import files

uploaded = files.upload()

import pandas as pd

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

from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import IsolationForest

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

import matplotlib.pyplot as plt

import seaborn as sns

# Load dataset

df = pd.read\_csv('creditcard.csv')

# Display first few rows

df.head()

# Shape of the dataset

print("Shape:", df.shape)

# Column names

print("Columns:", df.columns.tolist())

# Data types and non-null values

df.info()

# Summary statistics for numeric features

df.describe()

# Check for missing values

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

# Check for duplicates

print("Duplicate rows:", df.duplicated().sum())

target = 'Class'

features = df.columns.drop(target)

print("Features:", features)

# Identify categorical columns

categorical\_cols = df.select\_dtypes(include=['object']).columns

print("Categorical Columns:", categorical\_cols.tolist())

from sklearn.preprocessing import StandardScaler

# Create a StandardScaler object

scaler = StandardScaler()

# Scale the features (excluding the target variable 'Class')

X\_scaled = scaler.fit\_transform(df\_encoded.drop('Class', axis=1))

# Separate the target variable 'Class'

y = df\_encoded['Class']

# Assuming 'categorical\_cols' contains the names of your categorical columns

df\_encoded = pd.get\_dummies(df, columns=categorical\_cols, drop\_first=True)

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

from sklearn.linear\_model import LogisticRegression # Changed to LogisticRegression

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score # Changed metrics

# Split data

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

# Train model

model = LogisticRegression() # Changed to LogisticRegression

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

# Predict

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

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

# Calculate metrics

accuracy = accuracy\_score(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred)

recall = recall\_score(y\_test, y\_pred)

f1 = f1\_score(y\_test, y\_pred)

# Print metrics

print("Accuracy:", accuracy)

print("Precision:", precision)

print("Recall:", recall)

print("F1-score:", f1)

# Sample input (replace values with valid transaction data)

new\_transaction = {

'Time': 12345, # Replace with a transaction time

'V1': -1.23, # Replace with the value of feature V1

'V2': 2.34, # Replace with the value of feature V2

# ... (include all other relevant features V3 to V28)

# Include all features from V3 to V28 here with their respective values

'V3': 0, # Example - Replace with actual value

'V4': 0, # Example - Replace with actual value

# ... (continue for V5 to V28)

'Amount': 120.50 # Replace with the transaction amount

}

# Create a DataFrame for the new transaction

new\_transaction\_df = pd.DataFrame([new\_transaction])

# One-hot encode categorical features if necessary (if you have any in your dataset)

# ... (apply the same encoding used during training)

# Ensure new\_transaction\_df has the same columns as df\_encoded (used for training)

# 1. Get missing columns:

missing\_cols = set(df\_encoded.drop('Class', axis=1).columns) - set(new\_transaction\_df.columns)

# 2. Add missing columns to new\_transaction\_df and fill with 0:

for col in missing\_cols:

new\_transaction\_df[col] = 0

# 3. Remove extra columns (if any) to match df\_encoded's columns:

extra\_cols = set(new\_transaction\_df.columns) - set(df\_encoded.drop('Class', axis=1).columns)

new\_transaction\_df = new\_transaction\_df.drop(columns=list(extra\_cols))

# Reorder columns to match the order during training:

new\_transaction\_df = new\_transaction\_df[df\_encoded.drop('Class', axis=1).columns]

# Scale the features using the same StandardScaler used during training

new\_transaction\_scaled = scaler.transform(new\_transaction\_df)

# Make a prediction using the trained model

prediction = model.predict(new\_transaction\_scaled)

# Print the prediction

print("Prediction:", prediction[0]) # 0 for non-fraudulent, 1 for fraudulent

# ... (previous code for data preparation and scaling) ...

# Make a prediction using the trained model

prediction = model.predict(new\_transaction\_scaled)

# Print the prediction with formatting

print("💳 Transaction Prediction:", "Fraudulent" if prediction[0] == 1 else "Non-Fraudulent")

pip install --upgrade gradio

import pandas as pd

import gradio as gr

from sklearn.preprocessing import StandardScaler

# Assuming 'model' is your trained model and 'scaler' is your fitted StandardScaler

def predict\_grade(school, sex, age, address, famsize, Pstatus, Medu, Fedu,

Mjob, Fjob, reason, guardian, traveltime, studytime,

failures, schoolsup, famsup, paid, activities, nursery,

higher, internet, romantic, famrel, freetime, goout,

Dalc, Walc, health, absences, G1, G2):

# Create input dictionary

input\_data = {

'school': school, 'sex': sex, 'age': int(age), 'address': address, 'famsize': famsize,

'Pstatus': Pstatus, 'Medu': int(Medu), 'Fedu': int(Fedu), 'Mjob': Mjob, 'Fjob': Fjob,

'reason': reason, 'guardian': guardian, 'traveltime': int(traveltime), 'studytime': int(studytime),

'failures': int(failures), 'schoolsup': schoolsup, 'famsup': famsup, 'paid': paid,

'activities': activities, 'nursery': nursery, 'higher': higher, 'internet': internet,

'romantic': romantic, 'famrel': int(famrel), 'freetime': int(freetime), 'goout': int(goout),

'Dalc': int(Dalc), 'Walc': int(Walc), 'health': int(health), 'absences': int(absences),

'G1': int(G1), 'G2': int(G2)

}

# Create DataFrame

input\_df = pd.DataFrame([input\_data])

# Combine and encode

df\_temp = pd.concat([df.drop('G3', axis=1), input\_df], ignore\_index=True)

df\_temp\_encoded = pd.get\_dummies(df\_temp, drop\_first=True)

df\_temp\_encoded = df\_temp\_encoded.reindex(columns=df\_encoded.drop('G3', axis=1).columns, fill\_value=0)

# Scale and predict

scaled\_input = scaler.transform(df\_temp\_encoded.tail(1))

prediction = model.predict(scaled\_input)

return round(prediction[0], 2)

#launch app

gr.Interface(fn=predict\_grade, inputs=inputs, outputs="text", title="Student Grade Prediction").launch()