```
# 1. Import necessary 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
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix,
roc_auc_score
from imblearn.under_sampling import RandomUnderSampler
# 2. Load the dataset
df = pd.read_csv('credit_card_fraud_dataset.csv')
print("Data loaded successfully.")
print(df.head()
df.drop(['Transaction_ID', 'Customer_ID', 'Timestamp', 'Device_ID'],
axis=1, errors='ignore')
# 4. Encode categorical features
categorical_cols = ['Merchant_ID', 'Merchant_Category',
'Transaction_Type',
                    'Location_City', 'Location_Country', 'Channel',
                    'Is_3DS_Authenticated', 'Previous_Fraud_Flag']
print(df.columns)
# 5. Scale the 'Amount' column
scaler = StandardScaler()
df['Amount'] = scaler.fit_transform(df[['Amount']])
print("Before scaling:", df['Amount'].head())
df['Amount'] = scaler.fit_transform(df[['Amount']])
print("After scaling:", df['Amount'].head())
X = df.drop('Is_Fraud', axis=1)
y = df['Is_Fraud']
print(df.shape)
print(df.head())
 7. Split the data into train and test before resampling
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train distribution:\n", y_train.value_counts(normalize=True))
print("y_test distribution:\n", y_test.value_counts(normalize=True))
# 8. Handle class imbalance on training data
rus = RandomUnderSampler(random_state=42)
X_train_resampled, y_train_resampled = rus.fit_resample(X_train,
y_train)
print(f"Resampled training set shape: {X_train_resampled.shape}")
from sklearn.preprocessing import LabelEncoder
import pandas as pd
```

```
# Step 1: Encode label columns (if necessary)
le = LabelEncoder()
X_train_resampled['Merchant_Category'] =
le.fit_transform(X_train_resampled['Merchant_Category'])
X_test['Merchant_Category'] = le.transform(X_test['Merchant_Category'])

# Step 2: Apply one-hot encoding
X_train_resampled = pd.get_dummies(X_train_resampled)
X_test = pd.get_dummies(X_test)

# Step 3: Align test set with train set
X_test = X_test.reindex(columns=X_train_resampled.columns,
fill_value=0)

# Step 4: Fit the model
model.fit(X_train_resampled, y_train_resampled)

# Optional: print dtypes to verify all features are numeric
print(X_train_resampled.dtypes)
from sklearn.preprocessing import LabelEncoder
import pandas as pd
from sklearn.ensemble import RandomForestClassifier # Import the model

# Step 1: Encode label columns (if necessary)
le = LabelEncoder()
X_train_resampled['Merchant_Category'] =
le.fit_transform(X_train_resampled['Merchant_Category'])
X_test['Merchant_Category'] = le.transform(X_test['Merchant_Category'])

# Step 2: Apply one-hot encoding
X_train_resampled = pd.get_dummies(X_train_resampled)
X_test = pd.get_dummies(X_test)

# Step 3: Align test set with train set
X_test = X_test.reindex(columns=X_train_resampled.columns,
fill_value=0)

# Step 4: Fit the model
model = RandomForestClassifier(random_state=42) # Initialize the model
model.fit(X_train_resampled, y_train_resampled)

# Optional: print dtypes to verify all features are numeric
print(X_train_resampled.dtypes)
# 11. Feature Importance Plot
import matplotlib.pyplot as plt
import seaborn as sns

# Use the final feature names after encoding
```

```
# Changed line: Use the columns from the encoded data
feature_names = X_train_resampled.columns

# Get feature importances from the trained model
importances = model.feature_importances_

# Create a DataFrame for plotting
feat_df = pd.DataFrame({'Feature': feature_names, 'Importance':
importances})
feat_df.sort_values(by='Importance', ascending=False, inplace=True)

# Plot top 10 features
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feat_df.head(10))
plt.title('Top 10 Important Features')
plt.tight_layout()
plt.show()
import pandas as pd
import matplotlib.pyplot as plt

# Dataset load panrathu
df = pd.read_csv("credit_card_fraud_dataset.csv")

# Fraud count edukkarthu
fraud_counts = df['Is_Fraud'].value_counts()
labels = ['Not Fraud', 'Fraud']
colors = ['lightgreen', 'salmon']

# Pie chart draw panrathu
plt.figure(figsize=(6, 6))
plt.pie(fraud_counts, labels=labels, autopct='%1.1f%%', startangle=140,
colors=colors)
plt.title('Fraud vs Not Fraud Transactions (Pie Chart)')
plt.axis('equal')  # Equal aspect ratio
plt.show()
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd # Import pandas

# Dataset load
df = pd.read_csv("credit_card_fraud_dataset.csv")

# Create fraud summary DataFrame
fraud_summary = df.groupby('Transaction_Type')['Is_Fraud'].agg(
    ['sum', 'count']
).reset_index()
fraud_summary.columns = ['Transaction Type', 'Fraud Count', 'Count']
```

```
# Bar chart visualization
plt.figure(figsize=(6, 4))
sns.barplot(data=fraud_summary, x='Transaction Type', y='Count',
palette='Set2')
plt.title("Fraud vs Not Fraud Transactions")
plt.ylabel("Number of Transactions")
plt.xlabel("Transaction Type")
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
import smtplib
from email.mime.text import MIMEText
import os
import pandas as pd # Import pandas

# Set up environment variables (replace with your email and app
password)
os.environ['EMAIL'] = 'madhankumarm824@gmail.com'
os.environ['EMAIL_PASSWORD'] = 'Prince5502'  # Use your app-specific
password

# Function to trigger alert
def check_fraud_alert(row):
    alerts = []

    # Rules for triggering alerts
    if row['Amount'] > 5000:
        alerts.append("High Amount")
    if row['Card_Present_Flag'] == 0:
        alerts.append("Card Not Present")
    if row['Is_3DS_Authenticated'] == "No":
        alerts.append("3DS Not Authenticated")
    if row['Previous_Fraud_Flag'] == "Yes":
        alerts.append("Previous Fraud History")

    # Return alert message
    if alerts:
        return "ALERT: " + ", ".join(alerts)
    else:
        return "No Alert"

# Dataset load
df = pd.read_csv("credit_card_fraud_dataset.csv")

# Apply the alert logic to each transaction
df['Fraud_Alert'] = df.apply(check_fraud_alert, axis=1)

# Show transactions with alerts
alerts_df = df[df['Fraud_Alert'] != "No Alert"]
```

```
def send_email_alert(message):
    sender = os.getenv('EMAIL')
    receiver = 'madhankumarm824@gmail.com'
    password = os.getenv('EMAIL_PASSWORD')

    msg = MIMEText(message)
    msg['Subject'] = 'FRAUD ALERT!'
    msg['From'] = sender
    msg['To'] = receiver

    try:
        with smtplib.SMTP('smtp.gmail.com', 587) as server:
            server.starttls()
            server.login(sender, password)
            server.sendmail(sender, receiver, msg.as_string())
            print("Email sent successfully!")
    except Exception as e:
        print(f"Error sending email: {e}")

# Combine all alerts into one message
alert_messages = []
for _, row in alerts_df.iterrows():
    alert_messages.append(f"Transaction ID:
{row['Transaction_ID']}\nDetails: {row['Fraud_Alert']}\n")

# Send only if there are alerts
if alert_messages:
    full_message = "\n---\n".join(alert_messages)
    send_email_alert(full_message)
```