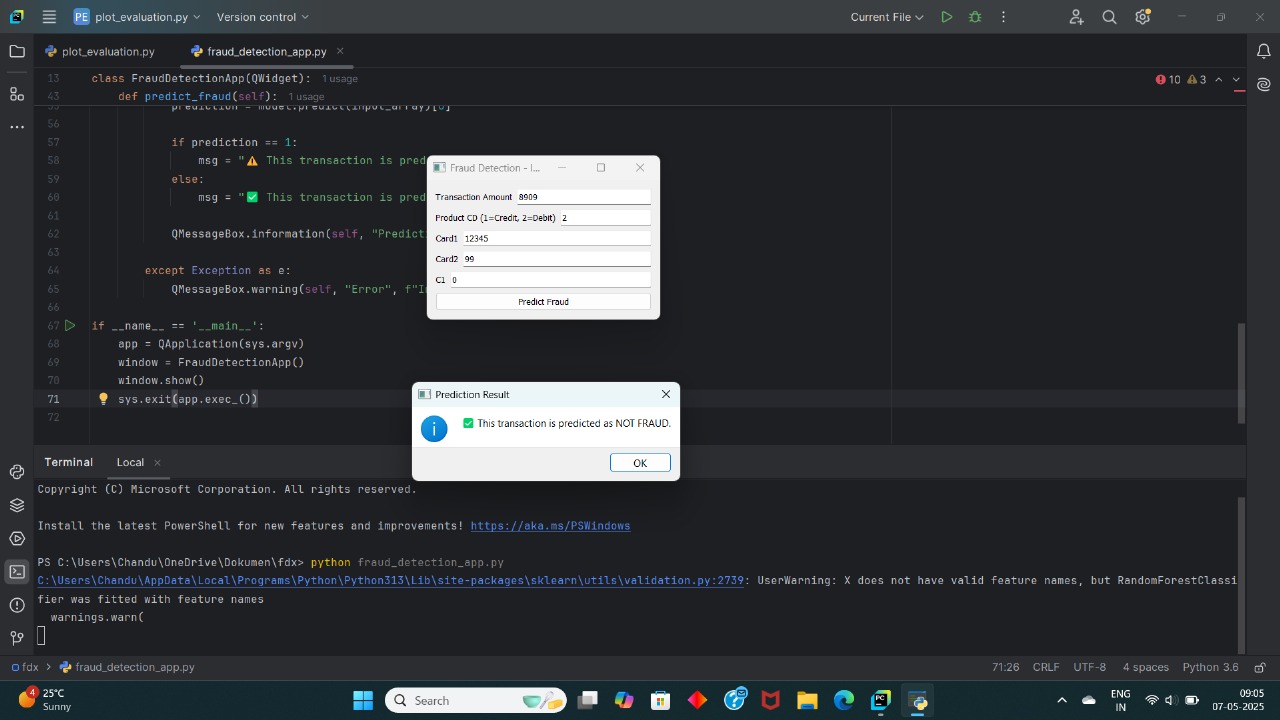
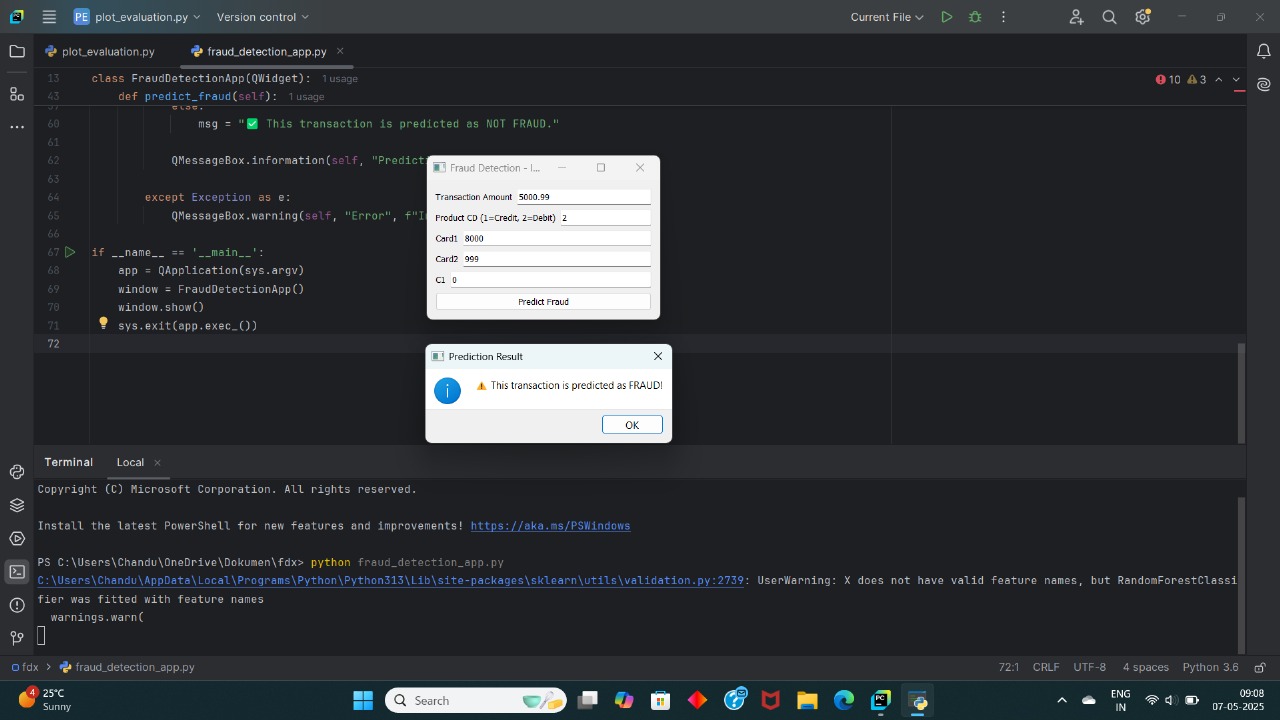
# Fraud\_detection\_app.py

import sys  
import pickle  
import numpy as np  
from PyQt5.QtWidgets import (  
 QApplication, QWidget, QLabel, QLineEdit,  
 QPushButton, QVBoxLayout, QHBoxLayout, QMessageBox  
)  
  
# Load the trained model  
with open('model.pkl', 'rb') as f:  
 model = pickle.load(f)  
  
class FraudDetectionApp(QWidget):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setWindowTitle('Fraud Detection - IEEE Dataset')  
 self.initUI()  
  
 def initUI(self):  
 layout = QVBoxLayout()  
  
 # Example fields - you can add more depending on your model  
 self.features = {  
 'Transaction Amount': QLineEdit(),  
 'Product CD (1=Credit, 2=Debit)': QLineEdit(),  
 'Card1': QLineEdit(),  
 'Card2': QLineEdit(),  
 'C1': QLineEdit(),  
 }  
  
 for label, widget in self.features.items():  
 row = QHBoxLayout()  
 row.addWidget(QLabel(label))  
 row.addWidget(widget)  
 layout.addLayout(row)  
  
 self.predict\_button = QPushButton('Predict Fraud')  
 self.predict\_button.clicked.connect(self.predict\_fraud)  
 layout.addWidget(self.predict\_button)  
  
 self.setLayout(layout)  
  
 def predict\_fraud(self):  
 try:  
 # Get feature values  
 input\_data = []  
 for label, widget in self.features.items():  
 value = float(widget.text())  
 input\_data.append(value)  
  
 # Convert to model input shape  
 input\_array = np.array(input\_data).reshape(1, -1)  
  
 # Predict  
 prediction = model.predict(input\_array)[0]  
  
 if prediction == 1:  
 msg = "⚠️ This transaction is predicted as FRAUD!"  
 else:  
 msg = "✅ This transaction is predicted as NOT FRAUD."  
  
 QMessageBox.information(self, "Prediction Result", msg)  
  
 except Exception as e:  
 QMessageBox.warning(self, "Error", f"Invalid Input: {e}")  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app = QApplication(sys.argv)  
 window = FraudDetectionApp()  
 window.show()  
 sys.exit(app.exec\_())

# output: is not fraud



# Is fraud



# Plot\_evaluation.py

import sys  
import pickle  
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay, roc\_curve, auc  
from PyQt5.QtWidgets import (  
 QApplication, QWidget, QVBoxLayout, QPushButton, QFileDialog, QMessageBox  
)  
from PyQt5.QtCore import Qt  
  
# Load the model  
with open('model.pkl', 'rb') as f:  
 model = pickle.load(f)  
  
# Load test data  
# Assume you have 'X\_test.csv' and 'y\_test.csv' saved  
X\_test = pd.read\_csv('X\_test.csv')  
y\_test = pd.read\_csv('y\_test.csv').values.ravel() # Convert to 1D array if necessary  
  
class PlotApp(QWidget):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setWindowTitle('Model Evaluation - Confusion Matrix & ROC Curve')  
 self.setGeometry(100, 100, 400, 200)  
 self.initUI()  
  
 def initUI(self):  
 layout = QVBoxLayout()  
  
 self.cm\_button = QPushButton('Plot Confusion Matrix')  
 self.cm\_button.clicked.connect(self.plot\_confusion\_matrix)  
 layout.addWidget(self.cm\_button)  
  
 self.roc\_button = QPushButton('Plot ROC Curve')  
 self.roc\_button.clicked.connect(self.plot\_roc\_curve)  
 layout.addWidget(self.roc\_button)  
  
 self.setLayout(layout)  
  
 def plot\_confusion\_matrix(self):  
 try:  
 y\_pred = model.predict(X\_test)  
 cm = confusion\_matrix(y\_test, y\_pred)  
 disp = ConfusionMatrixDisplay(confusion\_matrix=cm)  
 disp.plot(cmap='Blues')  
 plt.title('Confusion Matrix')  
 plt.show()  
 except Exception as e:  
 QMessageBox.warning(self, "Error", f"Could not plot Confusion Matrix: {e}")  
  
 def plot\_roc\_curve(self):  
 try:  
 if hasattr(model, "predict\_proba"):  
 y\_scores = model.predict\_proba(X\_test)[:,1] # Probability estimates  
 else:  
 y\_scores = model.decision\_function(X\_test) # For models like SVM  
  
 fpr, tpr, thresholds = roc\_curve(y\_test, y\_scores)  
 roc\_auc = auc(fpr, tpr)  
  
 plt.figure()  
 plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc\_auc:.2f})')  
 plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')  
 plt.xlim([0.0, 1.0])  
 plt.ylim([0.0, 1.05])  
 plt.xlabel('False Positive Rate')  
 plt.ylabel('True Positive Rate')  
 plt.title('Receiver Operating Characteristic')  
 plt.legend(loc="lower right")  
 plt.show()  
  
 except Exception as e:  
 QMessageBox.warning(self, "Error", f"Could not plot ROC Curve: {e}")  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app = QApplication(sys.argv)  
 window = PlotApp()  
 window.show()  
 sys.exit(app.exec\_())

# output

