Fraud Detection usiing Naive Bayes

In [1]:
```
import re
import joblib
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import PyPDF2
import pdfplumber
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score

# Modern GUI imports
from PyQt5.QtWidgets import (QApplication, QMainWindow, QVBoxLayout, QHBoxLayout
                             QLabel, QTextEdit, QPushButton, QFileDialog, QStatu
                             QTabWidget, QScrollArea, QGroupBox)
from PyQt5.QtCore import Qt, QSize
from PyQt5.QtGui import QFont, QPixmap, QIcon
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
from matplotlib.figure import Figure
```

In [2]:
```
class EmailFraudDetector(QMainWindow):
    def __init__(self, model, vectorizer):
        super().__init__()
        self.model = model
        self.vectorizer = vectorizer
        self.init_ui()

    def init_ui(self):
        self.setWindowTitle("AI Fraud Email Detector")
        self.setWindowIcon(QIcon('icon.png'))  # Add your icon file
        self.setMinimumSize(QSize(800, 600))

        # Create main widget and layout
        main_widget = QWidget()
        self.setCentralWidget(main_widget)
        main_layout = QVBoxLayout(main_widget)

        # Create tab widget
        tab_widget = QTabWidget()
        main_layout.addWidget(tab_widget)

        # Create tabs
        self.create_detection_tab(tab_widget)
        self.create_analysis_tab(tab_widget)
        self.create_help_tab(tab_widget)

        # Add status bar
        self.status_bar = QStatusBar()
        self.setStatusBar(self.status_bar)
        self.status_bar.showMessage("Ready")

    def create_detection_tab(self, tab_widget):
        detection_tab = QWidget()
        layout = QVBoxLayout(detection_tab)

        # Header
        header = QLabel("Fraud Email Detection System")
        header.setFont(QFont('Arial', 16, QFont.Bold))
        header.setAlignment(Qt.AlignCenter)
        layout.addWidget(header)

        # Email input area
        input_group = QGroupBox("Email Content")
        input_layout = QVBoxLayout()

        self.email_input = QTextEdit()
        self.email_input.setPlaceholderText("Paste email content here or load fr
        input_layout.addWidget(self.email_input)

        # Button row
        button_layout = QHBoxLayout()

        classify_btn = QPushButton("Classify Email")
        classify_btn.setStyleSheet("background-color: #4CAF50; color: white;")
        classify_btn.clicked.connect(self.classify_email)
        button_layout.addWidget(classify_btn)

        load_pdf_btn = QPushButton("Load PDF")
        load_pdf_btn.setStyleSheet("background-color: #2196F3; color: white;")
        load_pdf_btn.clicked.connect(self.load_pdf)
        button_layout.addWidget(load_pdf_btn)

        clear_btn = QPushButton("Clear")
        clear_btn.setStyleSheet("background-color: #f44336; color: white;")
        clear_btn.clicked.connect(self.clear_input)
        button_layout.addWidget(clear_btn)

        input_layout.addLayout(button_layout)
        input_group.setLayout(input_layout)
        layout.addWidget(input_group)

        # Results display
        result_group = QGroupBox("Analysis Results")
        result_layout = QVBoxLayout()

        self.result_label = QLabel("Result will appear here...")
        self.result_label.setFont(QFont('Arial', 14))
        self.result_label.setAlignment(Qt.AlignCenter)
        result_layout.addWidget(self.result_label)

        # Confidence meter
        self.confidence_label = QLabel("Confidence: ")
        self.confidence_label.setFont(QFont('Arial', 12))
        result_layout.addWidget(self.confidence_label)

        result_group.setLayout(result_layout)
        layout.addWidget(result_group)

        tab_widget.addTab(detection_tab, "Detection")

    def create_analysis_tab(self, tab_widget):
        analysis_tab = QWidget()
        layout = QVBoxLayout(analysis_tab)
        # Header
        header = QLabel("Model Analysis Dashboard")
        header.setFont(QFont('Arial', 16, QFont.Bold))
        header.setAlignment(Qt.AlignCenter)
        layout.addWidget(header)

        # Create matplotlib figure
        self.figure = Figure(figsize=(10, 8), dpi=100)
        self.canvas = FigureCanvas(self.figure)

        # Add scroll area for the plots
        scroll = QScrollArea()
        scroll.setWidgetResizable(True)
        scroll.setWidget(self.canvas)
        layout.addWidget(scroll)

        # Generate initial plots
        self.generate_plots()

        tab_widget.addTab(analysis_tab, "Analysis")

    def create_help_tab(self, tab_widget):
        help_tab = QWidget()
        layout = QVBoxLayout(help_tab)

        # Header
        header = QLabel("Help & Documentation")
        header.setFont(QFont('Arial', 16, QFont.Bold))
        header.setAlignment(Qt.AlignCenter)
        layout.addWidget(header)

        # Help content
        help_content = QLabel(
            "<h3>How to Use This Application</h3>"
            "<p>1. <b>Detection Tab:</b> Paste email content or load from PDF, t
            "<p>2. <b>Analysis Tab:</b> View model performance metrics and stati
            "<h3>About the Model</h3>"
            "<p>This system uses a Multinomial Naive Bayes classifier trained on
            "to detect potential fraud attempts with high accuracy.</p>"
            "<h3>Tips for Best Results</h3>"
            "<p>- Include full email headers when possible</p>"
            "<p>- Check for suspicious links or requests for personal informatio
            "<p>- Be cautious of urgent or threatening language</p>"
        )
        help_content.setWordWrap(True)
        help_content.setOpenExternalLinks(True)

        scroll = QScrollArea()
        scroll.setWidgetResizable(True)
        scroll.setWidget(help_content)
        layout.addWidget(scroll)

        tab_widget.addTab(help_tab, "Help")

    def generate_plots(self):
        """Generate model analysis plots"""
        self.figure.clear()

        # Example plot 1: Accuracy
        ax1 = self.figure.add_subplot(221)
        ax1.bar(['Train', 'Test'], [0.98, 0.96], color=['blue', 'green'])
        ax1.set_title('Model Accuracy')
        ax1.set_ylim(0, 1)

        # Example plot 2: Feature importance
        ax2 = self.figure.add_subplot(222)
        features = ['urgent', 'payment', 'account', 'verify', 'click']
        importance = [0.85, 0.76, 0.72, 0.68, 0.65]
        ax2.barh(features, importance, color='orange')
        ax2.set_title('Top Fraud Indicators')

        # Example plot 3: Class distribution
        ax3 = self.figure.add_subplot(223)
        labels = ['Legitimate', 'Fraud']
        counts = [1200, 800]
        ax3.pie(counts, labels=labels, autopct='%1.1f%%', colors=['green', 'red'
        ax3.set_title('Dataset Distribution')

        # Example plot 4: Confusion matrix
        ax4 = self.figure.add_subplot(224)
        cm = [[950, 50], [30, 770]]
        sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=ax4,
                    xticklabels=['Legit', 'Fraud'], yticklabels=['Legit', 'Fraud
        ax4.set_title('Confusion Matrix')
        ax4.set_xlabel('Predicted')
        ax4.set_ylabel('Actual')

        self.figure.tight_layout()
        self.canvas.draw()

    def classify_email(self):
        """Classify the email content"""
        email_text = self.email_input.toPlainText().strip()

        if not email_text:
            self.result_label.setText("Please enter email content to analyze")
            self.result_label.setStyleSheet("color: red;")
            return

        try:
            # Clean and predict
            clean_text = cl_em_text(email_text)
            features = self.vectorizer.transform([clean_text])
            prediction = self.model.predict(features)
            proba = self.model.predict_proba(features)[0]

            # Display results
            if prediction[0] == 1:
                result = "FRAUD DETECTED!"
                color = "red"
                confidence = proba[1]
            else:
                result = "Legitimate Email"
                color = "green"
                confidence = proba[0]

            self.result_label.setText(result)
            self.result_label.setStyleSheet(f"color: {color}; font-weight: bold;
            self.confidence_label.setText(f"Confidence: {confidence*100:.2f}%")
            self.status_bar.showMessage("Classification complete")

        except Exception as e:
            self.result_label.setText(f"Error during classification: {str(e)}")
            self.result_label.setStyleSheet("color: red;")

    def load_pdf(self):
        """Load email content from PDF file"""
        options = QFileDialog.Options()
        file_name, _ = QFileDialog.getOpenFileName(
            self, "Open PDF File", "", "PDF Files (*.pdf)", options=options)

        if file_name:
            try:
                with open(file_name, "rb") as file:
                    reader = PyPDF2.PdfReader(file)
                    text = ""
                    for page in reader.pages:
                        if page.extract_text():
                            text += page.extract_text() + "\n"
                    self.email_input.setPlainText(text.strip())
                    self.status_bar.showMessage(f"Loaded PDF: {file_name}")
            except Exception as e:
                self.status_bar.showMessage(f"Error reading PDF: {str(e)}")

    def clear_input(self):
        """Clear the email input field"""
        self.email_input.clear()
        self.result_label.setText("Result will appear here...")
        self.result_label.setStyleSheet("")
        self.confidence_label.setText("Confidence: ")
        self.status_bar.showMessage("Input cleared")
```

In [ ]:

```
# Preprocessing functions
def cl_em_text(email_text):
    """Clean email text by removing URLs, numbers, and punctuation"""
    email_text = re.sub(r'http\S+', '', email_text, flags=re.IGNORECASE)
    email_text = re.sub(r'\b\d+\b', '', email_text)
    email_text = re.sub(r'[^\w\s]', ' ', email_text)  # Replace punctuation with
    email_text = re.sub(r'\s+', ' ', email_text)  # Collapse multiple spaces/new
    return email_text.lower().strip()

def extract_features(email_texts):
    """Convert emails into numerical features"""
    vectorizer = CountVectorizer(stop_words='english', max_features=1000)
    features = vectorizer.fit_transform(email_texts)
    return features, vectorizer

def tr_model(features, labels):
    """Train the machine learning model for email scam detection"""
    X_train, X_test, Y_train, Y_test = train_test_split(features, labels, test_s
    model = MultinomialNB()
    model.fit(X_train, Y_train)
    predictions = model.predict(X_test)
    print(f"Model Accuracy: {accuracy_score(Y_test, predictions)}")
    joblib.dump(model, "scam_detector_model.pkl")
    return model

def load_dataset(file_path):
    """Load the dataset from a CSV file and extract emails and labels"""
    df = pd.read_csv(file_path)
    df.dropna(inplace=True)  # Remove any null values
    return df['email_text'], df['labels']

def main():
    # Load data and train model
    data_file = "emails.csv"
    emails, labels = load_dataset(data_file)
    features, vectorizer = extract_features(emails)
    model = tr_model(features, labels)

    # Create and show the GUI
    app = QApplication([])
    window = EmailFraudDetector(model, vectorizer)
    window.show()
    app.exec_()

if __name__ == "__main__":
    main()
```

```
Model Accuracy: 1.0
```

In [ ]: