CHAPTER 8

SAMPLE CODING (App.py)

```
import os

import sqlite3

from datetime import datetime

import pandas as pd

from flask import Flask, render_template, request, flash, redirect, url_for, jsonify, session

from flask_mail import Mail, Message

from werkzeug.utils import secure_filename

import matplotlib

matplotlib.use('Agg')

import random

from threading import Thread

from uuid import uuid4

from model import load_model, validate_input, generate_visualization


app = Flask(__name__)

app.secret_key = 'your-secret-key-here'


# Configuration

UPLOAD_FOLDER = 'uploads'

ALLOWED_EXTENSIONS = {'csv'}

MODEL_FOLDER = 'models'

DB_NAME = 'transactions.db'
```

```
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
app.config['MODEL_FOLDER'] = MODEL_FOLDER
app.config['MAIL_SERVER'] = 'smtp.example.com'
app.config['MAIL_PORT'] = 587
app.config['MAIL_USE_TLS'] = True
app.config['MAIL_USERNAME'] = 'developerharry18@gmail.com'
app.config['MAIL_PASSWORD'] = 'itst zeic zutz cknw'
app.config['MAIL_DEFAULT_SENDER'] = 'Tax_Evasion@gmail.com'

mail = Mail(app)

# Create directories if they don't exist
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
os.makedirs(MODEL_FOLDER, exist_ok=True)

# Initialize database
def init_db():
    with sqlite3.connect(DB_NAME) as conn:
        cursor = conn.cursor()
        cursor.execute('''
        CREATE TABLE IF NOT EXISTS transactions (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            transaction_id TEXT UNIQUE,
            country TEXT,
            amount REAL,
```

```
    transaction_type TEXT,
    tax_amount INTEGER,
    prediction_result TEXT,
    confidence INTEGER,
    timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
    user_email TEXT,
    notes TEXT
)
''')
cursor.execute('''
CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    email TEXT UNIQUE,
    name TEXT,
    organization TEXT,
    subscription_type TEXT,
    last_login DATETIME,
    is_admin BOOLEAN DEFAULT 0
)
''')
cursor.execute('''
CREATE TABLE IF NOT EXISTS user_auth (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    email TEXT UNIQUE,
    password_hash TEXT,
```

```
        FOREIGN KEY (email) REFERENCES users(email)
    )
    ''')
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS audit_log (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        user_email TEXT,
        action TEXT,
        details TEXT,
        timestamp DATETIME DEFAULT CURRENT_TIMESTAMP
    )
    ''')
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS model_metrics (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        prediction_result TEXT,
        confidence INTEGER,
        timestamp DATETIME DEFAULT CURRENT_TIMESTAMP
    )
    ''')
    conn.commit()

init_db()

# Load the model
```

```
model = load_model()


# Feature 1: Asynchronous email sending
def send_async_email(app, msg):
    with app.app_context():
        try:
            mail.send(msg)
        except Exception as e:
            app.logger.error(f"Error sending email: {e}")


def send_email(subject, recipients, body):
    msg = Message(subject, recipients=recipients)
    msg.body = body
    Thread(target=send_async_email, args=(app, msg)).start()


# Feature 4: Log audit trail
def log_audit(user_email, action, details):
    with sqlite3.connect(DB_NAME) as conn:
        cursor = conn.cursor()
        cursor.execute('''
        INSERT INTO audit_log (user_email, action, details)
        VALUES (?, ?, ?)
        ''', (user_email, action, details))
        conn.commit()
```

```
# Feature 5: Generate report
def generate_report():
    with sqlite3.connect(DB_NAME) as conn:
        df = pd.read_sql('SELECT * FROM transactions', conn)

    if df.empty:
        return None

    report = {
        'total_transactions': len(df),
        'legal_count': len(df[df['prediction_result'] == 'Legal']),
        'illegal_count': len(df[df['prediction_result'] == 'Illegal']),
        'highest_amount': df['amount'].max(),
        'most_common_country': df['country'].mode()[0],
        'visualization': generate_visualization(df)
    }
    return report

# Feature 6: Save transaction to database
def save_transaction(data, prediction_result, confidence,tax_amount, user_email=None,
notes=None ):
    transaction_id = str(uuid4())

    with sqlite3.connect(DB_NAME) as conn:
        cursor = conn.cursor()
        cursor.execute('''
```

```
        INSERT INTO transactions (
            transaction_id, country, amount, transaction_type,tax_amount,
            prediction_result, confidence, user_email, notes
        )
        VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
        ''', (
            transaction_id, data['country'], float(data['amount']),
            data['transaction_type'],tax_amount, prediction_result, confidence,
            user_email, notes
        ))
        conn.commit()
    return transaction_id

# Feature 7: User management
def add_user(email, name, organization, subscription_type='basic', is_admin=False):
    with sqlite3.connect(DB_NAME) as conn:
        cursor = conn.cursor()
        try:
            cursor.execute('''
            INSERT INTO users (email, name, organization, subscription_type, is_admin)
            VALUES (?, ?, ?, ?, ?)
            ''', (email, name, organization, subscription_type, is_admin))
            conn.commit()
            return True
        except sqlite3.IntegrityError:
```

```
        return False


# Feature 8: Transaction search

def search_transactions(search_term=None, start_date=None, end_date=None,
result_type=None):

    query = 'SELECT * FROM transactions WHERE 1=1'

    params = []


    if search_term:

        query += ' AND (country LIKE ? OR transaction_type LIKE ?)'

        params.extend([f'%{search_term}%', f'%{search_term}%'])


    if start_date:

        query += ' AND timestamp >= ?'

        params.append(start_date)


    if end_date:

        query += ' AND timestamp <= ?'

        params.append(end_date)


    if result_type:

        query += ' AND prediction_result = ?'

        params.append(result_type)


    query += ' ORDER BY timestamp DESC LIMIT 100'
```

```
    with sqlite3.connect(DB_NAME) as conn:
        df = pd.read_sql(query, conn, params=params)

    return df

# Feature 9: Model performance monitoring
def log_prediction_metrics(prediction_result, confidence):
    with sqlite3.connect(DB_NAME) as conn:
        cursor = conn.cursor()
        cursor.execute('''
        INSERT INTO model_metrics (prediction_result, confidence, timestamp)
        VALUES (?, ?, CURRENT_TIMESTAMP)
        ''', (prediction_result, confidence))
        conn.commit()

# Feature 10: Data export
def export_data(format='csv'):
    with sqlite3.connect(DB_NAME) as conn:
        df = pd.read_sql('SELECT * FROM transactions', conn)

    if df.empty:
        return None

    if format == 'csv':
        return df.to_csv(index=False)
```

```
    elif format == 'json':
        return df.to_json(orient='records')
    else:
        return None

# Tax calculation functions
def calculate_tax(income, age):
    tax = 0

    # Tax calculation for General Taxpayer
    if age < 60:
        if income <= 300000:
            tax = 0
        elif income <= 600000:
            tax = (income - 300000) * 0.05
        elif income <= 900000:
            tax = (income - 600000) * 0.10 + 15000  # 5% on ₹3L to ₹6L
        elif income <= 1200000:
            tax = (income - 900000) * 0.15 + 45000  # 10% on ₹6L to ₹9L
        elif income <= 1500000:
            tax = (income - 1200000) * 0.20 + 90000  # 15% on ₹9L to ₹12L
        else:
            tax = (income - 1500000) * 0.30 + 150000  # 20% on ₹12L to ₹15L

    # Tax calculation for Senior Citizens (60-80)
```

```
    elif 60 <= age < 80:
        if income <= 300000:
            tax = 0
        elif income <= 500000:
            tax = (income - 300000) * 0.05
        elif income <= 1000000:
            tax = (income - 500000) * 0.20 + 10000  # 5% on ₹3L to ₹5L
        else:
            tax = (income - 1000000) * 0.30 + 90000  # 20% on ₹5L to ₹10L

    # Tax calculation for Super Senior Citizens (80+)
    elif age >= 80:
        if income <= 500000:
            tax = 0
        elif income <= 1000000:
            tax = (income - 500000) * 0.20
        else:
            tax = (income - 1000000) * 0.30 + 100000  # 20% on ₹5L to ₹10L

    return tax

def get_indian_tax_brackets(age_group, year):
    """Indian tax brackets for FY 2023-24 (AY 2024-25)"""
    # Common brackets for all groups up to 50 years
    brackets = [
```

```
    (0, 300000, 0),          # 0% tax
    (300000, 600000, 0.05),   # 5% tax
    (600000, 900000, 0.10),   # 10% tax
    (900000, 1200000, 0.15),  # 15% tax
    (1200000, 1500000, 0.20), # 20% tax
    (1500000, float('inf'), 0.30) # 30% tax
]

# Additional slabs for senior citizens (60-80 years)
if age_group == 'senior_citizen':
    brackets = [
        (0, 300000, 0),          # 0% tax
        (300000, 500000, 0.05),   # 5% tax
        (500000, 1000000, 0.20),  # 20% tax
        (1000000, float('inf'), 0.30) # 30% tax
    ]

# Additional slabs for super senior citizens (80+ years)
elif age_group == 'super_senior':
    brackets = [
        (0, 500000, 0),          # 0% tax
        (500000, 1000000, 0.20),  # 20% tax
        (1000000, float('inf'), 0.30) # 30% tax
    ]
```

```
    return brackets


def calculate_indian_tax(income, brackets):
    """Calculate tax based on Indian tax brackets"""
    tax = 0
    for i, bracket in enumerate(brackets):
        lower, upper, rate = bracket
        if income > lower:
            if i == len(brackets) - 1:  # Last bracket
                taxable_in_bracket = income - lower
            else:
                taxable_in_bracket = min(income, upper) - lower
            tax += taxable_in_bracket * rate
    return tax


def calculate_rebate(tax_amount, taxable_income, age_group):
    """Calculate rebate under Section 87A"""
    rebate = 0
    if age_group in ['general', 'women']:
        if taxable_income <= 700000:  # Up to 7 lakhs
            rebate = min(tax_amount, 25000)
    elif age_group == 'senior_citizen':
        if taxable_income <= 750000:  # Up to 7.5 lakhs
            rebate = min(tax_amount, 25000)
    return rebate
```

```
def get_tax_breakdown(taxable_income, brackets):
    """Generate detailed tax breakdown"""
    breakdown = []
    for i, bracket in enumerate(brackets):
        lower, upper, rate = bracket
        if taxable_income > lower:
            if i == len(brackets) - 1:  # Last bracket
                amount = taxable_income - lower
            else:
                amount = min(taxable_income, upper) - lower
            tax = amount * rate
            breakdown.append({
                'range': "₹{:,.0f} - ₹{:,.0f}".format(lower, upper),
                'rate': "{:.0%}".format(rate),
                'amount': "₹{:,.0f}".format(amount),
                'tax': "₹{:,.0f}".format(tax)
            })
    return breakdown


# File upload helper functions
def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS
```

```
from flask import current_app

def process_uploaded_file(filepath, user_email):
    try:
        # Load and preprocess data
        df = pd.read_csv(filepath)

        # Validate required columns
        required_columns = ['Country', 'Amount', 'Transaction Type']
        missing_cols = [col for col in required_columns if col not in df.columns]
        if missing_cols:
            raise ValueError(f"Missing required columns: {', '.join(missing_cols)}")

        # Add missing columns with default values
        for col in ['Person Involved', 'Industry', 'Destination Country']:
            if col not in df.columns:
                df[col] = 'Unknown'

        if 'Money Laundering Risk Score' not in df.columns:
            df['Money Laundering Risk Score'] = 5

        if 'Shell Companies Involved' not in df.columns:
            df['Shell Companies Involved'] = 0

        # Add datetime features
```

```
now = datetime.now()
df['Transaction_Year'] = now.year
df['Transaction_Month'] = now.month
df['Transaction_Day'] = now.day
df['Transaction_DayOfWeek'] = now.weekday()
df['Transaction_Hour'] = now.hour
df['Reported by Authority'] = False

# Make predictions
predictions = model.predict(df)
probas = model.predict_proba(df)[:, 1]

# Save results to database
with sqlite3.connect(DB_NAME) as conn:
    for i, row in df.iterrows():
        confidence = probas[i] if predictions[i] == 1 else (1 - probas[i])

        if confidence >= 0.60:
            result = "Legal"
        else:
            result = "Illegal"

        # Fix the deprecated warning by using .iloc properly
        income = df.iloc[i, 2]  # Changed from [2] to ,2
        age = 45
```

```
            tax_amount = calculate_tax(income, age)
            cursor = conn.cursor()
            cursor.execute('''
            INSERT INTO transactions (
                transaction_id, country, amount, transaction_type, tax_amount,
                prediction_result, confidence, user_email
            )
            VALUES (?, ?, ?, ?, ?, ?, ?, ?)
            ''', (
                str(uuid4()), row['Country'], float(row['Amount']),
                row['Transaction Type'], tax_amount, result, confidence, user_email
            ))
        conn.commit()

    # Send completion email - using current_app within app context
    if user_email:
        with current_app.app_context():
            send_email(
                "Bulk Transaction Processing Complete",
                [user_email],
                f"Your file {os.path.basename(filepath)} has been processed successfully."
            )

    # Log audit
```

```
        with current_app.app_context():
            log_audit(
                user_email or 'anonymous',
                'bulk_upload',
                f'Processed file {os.path.basename(filepath)} with {len(df)} transactions'
            )

    except Exception as e:
        # Log error and send error email within app context
        current_app.logger.error(f"Error processing uploaded file: {e}")
        if user_email:
            with current_app.app_context():
                send_email(
                    "Bulk Transaction Processing Failed",
                    [user_email],
                    f"An error occurred while processing your file {os.path.basename(filepath)}:
{str(e)}"
                )
    finally:
        try:
            os.remove(filepath)
        except Exception as e:
            current_app.logger.error(f"Error removing temporary file: {e}")
# Routes
@app.route('/')
def index():
```

```
    return render_template('index.html')


@app.route('/analyze', methods=['GET', 'POST'])
def analyze():
    if request.method == 'POST':
        # Get form data
        form_data = {
            'country': request.form.get('country'),
            'amount': request.form.get('amount'),
            'transaction_type': request.form.get('transaction_type'),
            'person_involved': request.form.get('person_involved'),
            'industry': request.form.get('industry'),
            'destination_country': request.form.get('destination_country'),
            'risk_score': request.form.get('risk_score'),
            'shell_companies': request.form.get('shell_companies'),
            'financial_institution': request.form.get('financial_institution'),
            'tax_haven': request.form.get('tax_haven'),
            'notes': request.form.get('notes')
        }

        # Validate input
        is_valid, message = validate_input(form_data)
        if not is_valid:
            flash(message, 'error')
            return redirect(url_for('analyze'))
```