TALENT TREK Internship Project

Finance Assistant

- PUDARI VYSHNAVI

Description:

A Flask-based web application that helps users manage their personal finances. It classifies expenses, recommends budgets using machine learning, and provides financial insights through visual reports and an AI chatbot

Technologies Used:

Python, Flask, Pandas, NumPy, Scikit-learn, Matplotlib, HTML/CSS etc.

Project includes:

1. Complete Flask Web Application
2. 3 Machine Learning Models
3. Professional Frontend with Bootstrap
4. File Upload C Processing
5. Data Visualization Charts
6. PDF Report Generation etc.

Project Structure:

```
finance_assistant/
├── app.py                          # Main Flask application
├── config.py                       # Configuration settings
├── requirements.txt                # Python dependencies
├── README.md                       # Project documentation
├── sample_data.csv                 # Sample transaction data
├── ml_models/
│   ├── __init__.py
│   ├── expense_classifier.py       # ML model for expense classification
│   ├── budget_recommender.py       # ML model for budget recommendations
│   └── nlp_chatbot.py              # NLP chatbot model
├── utils/
│   ├── __init__.py
│   ├── data_processor.py           # Data processing utilities
│   ├── security.py                 # Security and validation
│   ├── visualization.py            # Chart generation
│   ├── report_generator.py         # PDF report generation
│   └── error_handler.py            # Error handling and logging
├── static/
│   ├── css/
│   │   └── style.css               # Main stylesheet
│   ├── js/
│   │   └── main.js                 # Frontend JavaScript
│   ├── uploads/                    # File upload directory
│   ├── reports/                    # PDF reports directory
│   └── exports/                    # Data exports directory
└── templates/
    ├── base.html                   # Base template
    ├── dashboard.html              # Main dashboard
    └── chat.html                   # Chat interface
```

COMPLETE SOURCE CODE

1. app.py:

```
from flask import Flask, render_template, request, jsonify, session, send_file
import pandas as pd
import numpy as np
import json
import os
from datetime import datetime, timedelta
import traceback

# Import custom modules
from ml_models.expense_classifier import ExpenseClassifier
from ml_models.budget_recommender import BudgetRecommender
from ml_models.nlp_chatbot import NLPChatbot
from utils.data_processor import DataProcessor
from utils.security import SecurityManager
from utils.visualization import FinanceVisualizer
from utils.report_generator import PDFReportGenerator
from utils.error_handler import ErrorHandler

app = Flask(__name__)
app.config.from_object('config.Config')

# Initialize managers
security_manager = SecurityManager()
data_processor = DataProcessor()
finance_visualizer = FinanceVisualizer()
pdf_generator = PDFReportGenerator()
error_handler = ErrorHandler(app)

# Initialize error handling
error_handler.register_handlers(app)

# Global variables for ML models
expense_classifier = None
budget_recommender = None
nlp_chatbot = None
```

```
def  initialize_ml_models():  """Initialize
        ML models"""
        global expense_classifier, budget_recommender, nlp_chatbot try:
                expense_classifier            =            ExpenseClassifier()
                budget_recommender      =      BudgetRecommender()
                nlp_chatbot = NLPChatbot()
                print("[INFO] ML models initialized successfully") except Exception
        as e:
                print(f"[ERROR] Error initializing ML models: {e}") raise e


# Request logging middleware @app.before_request
def before_request():
        error_handler.log_api_request()

@app.after_request
def after_request(response):
        return error_handler.log_api_response(response)

@app.route('/') def
index():
        """Main dashboard page"""
        return render_template('dashboard.html')

@app.route('/chat') def
chat_interface():
        """Chat interface for financial queries""" return
        render_template('chat.html')

@app.route('/api/health') def
health_check():
        """API health check endpoint""" return
        jsonify({
                'status': 'healthy',
                'timestamp': datetime.now().isoformat(), 'version': '1.0.0',
                'models_loaded': all(model is not None for model in [expense_classifier, budget_recommender,
nlp_chatbot])
        })

@app.route('/api/analyze-spending', methods=['POST'])
```

```
def analyze_spending():
        """API endpoint for spending analysis with visualization""" try:
                data = request.get_json()

                # Validate input
                if not data or 'transactions' not in data: return jsonify({
                                'status': 'error',
                                'message': 'No transactions provided'
                        }), 400

                transactions = data.get('transactions', [])

                # Validate each transaction
                for transaction in transactions:
                        if not security_manager.validate_transaction_data(transaction): return jsonify({
                                        'status': 'error',
                                        'message': 'Invalid transaction data format'
                                }), 400

                # Classify transactions using ML
                classified_transactions = expense_classifier.predict_batch(transactions)

                # Analyze spending patterns spending_analysis =
budget_recommender.analyze_spending_patterns(classified_transactions)

                # Generate budget recommendations
                total_spending = spending_analysis.get('total_spending', 0) monthly_income =
                data.get('monthly_income', total_spending * 1.5) budget_recommendations =
                budget_recommender.recommend_budget(
                        monthly_income=monthly_income,
                        current_spending=total_spending
                )

                # Generate charts
                category_data = spending_analysis.get('category_breakdown', {}) charts =
                finance_visualizer.generate_dashboard_charts(
                        classified_transactions, category_data,
                        budget_recommendations
```

```
            )

            return jsonify({ 'status':'success',
                'classified_transactions': classified_transactions, 'spending_analysis':
                spending_analysis, 'budget_recommendations': budget_recommendations,
                'charts': charts
            })

        except Exception as e:
            error_handler.logger.error(f"Spending analysis error: {str(e)}") return jsonify({
                'status': 'error',
                'message': 'Failed to analyze spending data'
            }), 500

@app.route('/api/upload-transactions', methods=['POST']) def
upload_transactions():
        """API endpoint for file upload and processing""" try:
            if 'file' not in request.files:
                return jsonify({'status': 'error', 'message': 'No file uploaded'}),
400

            file = request.files['file'] if
            file.filename == '':
                return jsonify({'status': 'error', 'message': 'No file selected'}),
400

            # Validate file type
            if not security_manager.check_file_safety(file.filename): return jsonify({
                    'status': 'error',
                    'message': 'Invalid file type. Please upload CSV or Excel files

only.'                 }), 400


            # Save uploaded file temporarily
            timestamp = datetime.now().strftime('%Y%m%d_%H%M%S') file_path =
f"static/uploads/{timestamp}_{security_manager.sanitize_input(file.filename)}" os.makedirs('static/uploads',
            exist_ok=True)
```

```
            file.save(file_path)

            # Process the file
            processing_result = data_processor.process_uploaded_file(file_path)

            if 'error' in processing_result: # Clean up
                temporary file try:
                    os.remove(file_path)
                except:
                    pass

                return jsonify({ 'status':'error',
                    'message': processing_result['error']
                }), 400

            # Classify transactions using ML transactions =
            processing_result['data']
            classified_transactions = expense_classifier.predict_batch(transactions)

            # Generate insights
            spending_analysis =
budget_recommender.analyze_spending_patterns(classified_transactions)

            # Generate charts
            category_data = spending_analysis.get('category_breakdown', {}) charts =
finance_visualizer.generate_dashboard_charts(classified_transactions, category_data)

            # Clean up temporary file try:
                os.remove(file_path) except:
                pass

            return jsonify({ 'status':'success',
                'transactions': classified_transactions, 'insights':
                processing_result['insights'], 'spending_analysis':
                spending_analysis, 'charts': charts,
```

```
                'total_records': processing_result['total_records']
            })

    except Exception as e:
        error_handler.logger.error(f"File upload error: {str(e)}") return jsonify({
            'status': 'error',
            'message': 'Failed to process uploaded file'
        }), 500


@app.route('/api/budget-recommendation', methods=['POST']) def
get_budget_recommendation():
    """API endpoint for budget recommendations""" try:
        data = request.get_json()
        monthly_income = data.get('monthly_income')

        if not monthly_income: return
            jsonify({
                'status': 'error',
                'message': 'Monthly income is required'
            }), 400

        try:
            monthly_income = float(monthly_income) if
            monthly_income <= 0:
                raise ValueError("Income must be positive") except
        (ValueError, TypeError):
            return jsonify({ 'status':'error',
                'message': 'Valid monthly income is required'
            }), 400

        current_spending = data.get('current_spending') savings_goal =
        data.get('savings_goal')

        recommendations = budget_recommender.recommend_budget(
            monthly_income=monthly_income, current_spending=current_spending,
            savings_goal=savings_goal
        )

        return jsonify({
```

```
                'status': 'success', 'recommendations':
                recommendations
            })

    except Exception as e:
        error_handler.logger.error(f"Budget recommendation error: {str(e)}") return jsonify({
            'status': 'error',
            'message': 'Failed to generate budget recommendations'
        }), 500


@app.route('/api/chat', methods=['POST']) def
chat_response():
    """API endpoint for chatbot queries with NLP""" try:
        user_message = request.json.get('message', '')

        if not user_message.strip(): return
            jsonify({
                'status': 'error',
                'message': 'Empty message received'
            }), 400

        # Sanitize input
        sanitized_message = security_manager.sanitize_input(user_message)

        # Generate response using NLP chatbot
        response = nlp_chatbot.generate_response(sanitized_message)

        # Add sentiment analysis for personalized touch
        sentiment = nlp_chatbot.analyze_financial_sentiment(sanitized_message)

        return jsonify({ 'status': 'success',
            'response': response,
            'sentiment': sentiment,
            'intent': nlp_chatbot.predict_intent(sanitized_message)
        })

    except Exception as e: error_handler.logger.error(f"Chat error: {str(e)}")
        return jsonify({
            'status': 'error',
```

```
                'message': 'Sorry, I encountered an error processing your message.'
            }), 500

@app.route('/api/generate-pdf-report', methods=['POST']) def
generate_pdf_report():
    """API endpoint to generate PDF financial report""" try:
        data = request.get_json()

        if not data:
            return jsonify({ 'status':'error',
                'message': 'No analysis data provided'
            }), 400

        # Create reports directory os.makedirs('static/reports',
        exist_ok=True)

        # Generate unique filename
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S") filename
        = f"financial_report_{timestamp}.pdf" filepath =
        f"static/reports/{filename}"

        # Generate PDF report
        success = pdf_generator.generate_financial_report(data, filepath)

        if success:
            return jsonify({ 'status':'success',
                'message': 'PDF report generated successfully', 'download_url':
                f'/static/reports/{filename}'
            })
        else:
            return jsonify({ 'status':'error',
                'message': 'Failed to generate PDF report'
            }), 500

    except Exception as e:
        error_handler.logger.error(f"PDF generation error: {str(e)}") return jsonify({
            'status': 'error',
            'message': 'Failed to generate PDF report'
```

```
        }), 500

@app.route('/api/export-data', methods=['POST']) def
export_data():
    """API endpoint to export data as Excel/CSV""" try:
        data = request.get_json()
        transactions = data.get('transactions', [])
        export_format = data.get('format', 'excel')  # excel or csv

        if not transactions: return
            jsonify({
                'status': 'error',
                'message': 'No data to export'
            }), 400

        # Create exports directory os.makedirs('static/exports',
        exist_ok=True)

        # Generate unique filename
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")

        if export_format == 'excel':
            filename = f"financial_data_{timestamp}.xlsx" filepath =
            f"static/exports/{filename}"

            # Create DataFrame and export to Excel df =
            pd.DataFrame(transactions)
            df.to_excel(filepath, index=False, engine='openpyxl')

        else:  # CSV
            filename = f"financial_data_{timestamp}.csv" filepath =
            f"static/exports/{filename}"

            # Create DataFrame and export to CSV df =
            pd.DataFrame(transactions) df.to_csv(filepath,
            index=False)

        return jsonify({ 'status': 'success',
            'message': f'Data exported successfully as {export_format.upper()}', 'download_url':
            f'/static/exports/{filename}'
```

```
            })

        except Exception as e:
            error_handler.logger.error(f"Data export error: {str(e)}") return jsonify({
                'status': 'error',
                'message': 'Failed to export data'
            }), 500

@app.route('/static/reports/<filename>') def
download_report(filename):
        """Serve generated PDF reports""" try:
            return send_file(f'static/reports/{filename}', as_attachment=True) except FileNotFoundError:
            return jsonify({'status': 'error', 'message': 'Report not found'}), 404

@app.route('/static/exports/<filename>') def
download_export(filename):
        """Serve exported data files""" try:
            return send_file(f'static/exports/{filename}', as_attachment=True) except FileNotFoundError:
            return jsonify({'status': 'error', 'message': 'Export file not found'}),
404

if __name__== '__main__':
        # Create necessary directories os.makedirs('ml_models/saved_models',
        exist_ok=True) os.makedirs('static/uploads', exist_ok=True)
        os.makedirs('static/reports', exist_ok=True) os.makedirs('static/exports',
        exist_ok=True)

        print("[INFO] Initializing AI Finance Assistant...")

        # Initialize ML models try:
            initialize_ml_models()
            print("[INFO] Starting Finance Assistant Server...")

            # Run the application
            app.run(debug=True, host='0.0.0.0', port=5000)
```

```
    except Exception as e:
        print(f"[ERROR] Failed to start Finance Assistant: {e}")
        traceback.print_exc()
```

2. config.py:

```
import os
from datetime import timedelta

class Config:
    SECRET_KEY = os.environ.get('SECRET_KEY') or 'dev-secret-key-2024'
    DEBUG = True
    TEMPLATES_AUTO_RELOAD = True

    # Session configuration
    PERMANENT_SESSION_LIFETIME = timedelta(minutes=30)

    # File upload settings
    MAX_CONTENT_LENGTH = 16 * 1024 * 1024  # 16MB max file size

    # ML Model paths
    MODEL_PATH = 'ml_models/saved_models/'

    # Sample categories for expense classification
    EXPENSE_CATEGORIES = [
        'Food & Dining', 'Transportation', 'Shopping',
        'Entertainment', 'Bills & Utilities', 'Healthcare',
        'Education', 'Travel', 'Other'
    ]
```

3. requirements.txt:

```
# Web Framework
flask==2.3.3

# Data Processing
pandas==2.0.3
numpy==1.24.3
openpyxl==3.1.2

# Machine Learning
```

```
scikit-learn==1.3.0
nltk==3.8.1
joblib==1.3.2

# Visualization & Reporting
matplotlib==3.7.2
seaborn==0.12.2
reportlab==4.0.4

# Natural Language Processing
textblob==0.17.1

# Security
pyjwt==2.8.0

# Utilities
python-dotenv==1.0.0
```

4. ml_models/expense_classifier.py:

```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import joblib
import os

class ExpenseClassifier:
    def __init__(self):
        self.model = None
        self.vectorizer = None
        self.categories = [
            'Food & Dining', 'Transportation', 'Shopping',
            'Entertainment', 'Bills & Utilities', 'Healthcare',
            'Education', 'Travel', 'Other'
        ]
        self.model_path = 'ml_models/saved_models/expense_classifier.pkl'
        self.vectorizer_path = 'ml_models/saved_models/vectorizer.pkl'
```

```
        self.load_or_train_model()

    def load_or_train_model(self):
        """Load existing model or train a new one""" if
        os.path.exists(self.model_path) and
os.path.exists(self.vectorizer_path):
            try:
                self.model = joblib.load(self.model_path) self.vectorizer =
                joblib.load(self.vectorizer_path) print("[INFO] Pre-trained expense
                classifier loaded
successfully")

            except:
                print("[WARNING] Error loading pre-trained models. Training new
models...")
                self.train_sample_model()
        else:
            print("[INFO] Training new expense classification model...") self.train_sample_model()

    def train_sample_model(self):
        """Train model with sample financial data"""
        # Sample training data - real transactions with descriptions and categories
        sample_data = { 'description':[
                'mcdonalds restaurant', 'starbucks coffee', 'supermarket
grocery',
                'uber ride', 'gas station', 'bus fare', 'train ticket', 'amazon shopping', 'clothing store',
                'electronics purchase', 'netflix subscription', 'movie tickets', 'concert event',
                'electricity bill', 'water bill', 'internet provider', 'doctor visit', 'pharmacy', 'hospital
                bill',
                'online course', 'book store', 'university fee', 'flight tickets', 'hotel booking',
                'airbnb rental', 'bank fee', 'atm withdrawal', 'insurance payment'
            ],
            'category': [
                'Food & Dining', 'Food & Dining', 'Food & Dining', 'Transportation', 'Transportation',
                'Transportation',
'Transportation',
                'Shopping', 'Shopping', 'Shopping', 'Entertainment', 'Entertainment',
                'Entertainment',
                'Bills & Utilities', 'Bills & Utilities', 'Bills & Utilities', 'Healthcare', 'Healthcare', 'Healthcare',
```

```
                    'Education', 'Education', 'Education', 'Travel', 'Travel',
                    'Travel',
                    'Other', 'Other', 'Other'
                ]
            }

            df = pd.DataFrame(sample_data)

            # Create TF-IDF features from transaction descriptions
            self.vectorizer = TfidfVectorizer(max_features=100, stop_words='english')
            X = self.vectorizer.fit_transform(df['description']).toarray() y = df['category']

            # Train Random Forest classifier
            self.model = RandomForestClassifier(n_estimators=100, random_state=42) self.model.fit(X, y)

            # Save the trained model and vectorizer
            os.makedirs('ml_models/saved_models', exist_ok=True)
            joblib.dump(self.model, self.model_path) joblib.dump(self.vectorizer,
            self.vectorizer_path)

        print("[INFO] Expense classification model trained and saved successfully")

    def predict_category(self, transaction_description): """Predict category for a
            transaction description""" try:
                if self.model is None or self.vectorizer is None: return "Other"

                # Transform the input description X_input =
self.vectorizer.transform([transaction_description.lower()]).toarray() prediction =
                self.model.predict(X_input)[0]
                confidence = np.max(self.model.predict_proba(X_input))

                # Return "Other" if confidence is too low if confidence
                < 0.3:
                    return "Other"

                return prediction except
            Exception as e:
```

```
            print(f"Error in prediction: {e}")
            return "Other"

    def predict_batch(self, transactions):
        """Predict categories for multiple transactions"""
        results = []
        for transaction in transactions:
            if isinstance(transaction, str):
                category = self.predict_category(transaction)
                results.append({
                    'description': transaction,
                    'category': category
                })
            elif isinstance(transaction, dict):
                description = transaction.get('description', '')
                category = self.predict_category(description)
                results.append({
                    **transaction,
                    'predicted_category': category
                })
        return results
```

5. ml_models/budget_recommender.py:

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import joblib
import os

class BudgetRecommender:
    def __init__(self):
        self.scaler = StandardScaler()
        self.model_path = 'ml_models/saved_models/budget_recommender.pkl'
        self.scaler_path = 'ml_models/saved_models/budget_scaler.pkl'
        self.load_or_train_model()

    def load_or_train_model(self):
        """Load existing model or train a new one"""
```

```
        if os.path.exists(self.model_path) and os.path.exists(self.scaler_path): try:
                self.model = joblib.load(self.model_path) self.scaler =
                joblib.load(self.scaler_path) print("[INFO] Pre-trained budget
                recommender loaded
successfully")

            except:
                print("[WARNING] Error loading pre-trained models. Training new
models...")
                self.train_sample_model()
        else:
            print("[INFO] Training new budget recommendation model...") self.train_sample_model()

    def train_sample_model(self):
        """Train model with sample user financial profiles""" # Sample data:
        [monthly_income, essential_spending,
discretionary_spending, savings_goal] sample_data =
        np.array([
            [3000, 1500, 800, 500],              # Conservative spender
            [5000,   2000,   1500,   1000],    #  Balanced spender
            [7000,   2500,   3000,   800],     #  Aggressive spender
            [4000,   1800,   1000,   700],     #  Moderate saver
            [6000,   2200,   2000,   1200],    #  Aggressive saver
             [3500,   1600,   900, 600],       #  Cautious spender
             [8000,   3000,   2500, 1500],     #  High income balanced
            [4500, 2000, 1200, 800],   # Moderate income
        ])

        # Scale the data
        scaled_data = self.scaler.fit_transform(sample_data)

        # Train K-means clustering to identify spending patterns self.model =
        KMeans(n_clusters=3, random_state=42) self.model.fit(scaled_data)

        # Save models
        os.makedirs('ml_models/saved_models', exist_ok=True)
        joblib.dump(self.model, self.model_path) joblib.dump(self.scaler,
        self.scaler_path)

    print("[INFO] Budget recommendation model trained and saved successfully")
```

```
    def recommend_budget(self, monthly_income, current_spending=None, savings_goal=None):
        """Generate personalized budget recommendations""" try:
            # Default values if not provided if
            current_spending is None:
                current_spending = monthly_income * 0.6  # Assume 60% spending if savings_goal
            is None:
                savings_goal = monthly_income * 0.2  # Assume 20% savings goal

            # Create user profile
            user_profile = np.array([[monthly_income, current_spending, monthly_income - current_spending
- savings_goal, savings_goal]])

            # Scale and predict cluster
            scaled_profile = self.scaler.transform(user_profile) cluster =
            self.model.predict(scaled_profile)[0]

            # Generate recommendations based on cluster if cluster ==
            0:  # Conservative
                recommendations = {
                    'category': 'Conservative Budget', 'essential_spending': monthly_income *
                    0.5, 'discretionary_spending': monthly_income * 0.2, 'savings':
                    monthly_income * 0.3,
                    'advice': 'Great saving habits! Consider investing your savings for higher returns.'
                }
            elif cluster == 1:  # Balanced
                recommendations = {
                    'category': 'Balanced Budget', 'essential_spending': monthly_income
                    * 0.55, 'discretionary_spending': monthly_income * 0.25, 'savings':
                    monthly_income * 0.2,
                    'advice': 'Good balance! Try to increase savings by 5% next
month.'
                }
            else:  # Aggressive recommendations = {
                    'category': 'Optimized Budget', 'essential_spending': monthly_income
                    * 0.45, 'discretionary_spending': monthly_income * 0.15, 'savings':
                    monthly_income * 0.4,
```

```
                                'advice': 'Consider reducing discretionary spending to
increase savings.'
                        }

                # Add personalized tips
                savings_rate = (recommendations['savings'] / monthly_income) * 100 if savings_rate >= 30:
                        recommendations['tip'] = 'Excellent savings rate! You\'re on track for financial
freedom.'
                elif savings_rate >= 20:
                        recommendations['tip'] = 'Good savings rate. Consider automating
your savings.'
                else:
                        recommendations['tip'] = 'Try to increase your savings rate by cutting unnecessary expenses.'

                return recommendations

        except Exception as e:
                print(f"Error in budget recommendation: {e}") # Return
                default recommendations
                return {
                        'category': 'Standard Budget', 'essential_spending': monthly_income
                        * 0.5, 'discretionary_spending': monthly_income * 0.3, 'savings':
                        monthly_income * 0.2,
                        'advice': 'Follow the 50-30-20 rule: 50% needs, 30% wants, 20%
savings.',
                        'tip': 'Track your expenses to identify areas for improvement.'
                }

    def analyze_spending_patterns(self, transactions): """Analyze spending
        patterns from transaction data""" if not transactions:
                return {"error": "No transactions provided"}

        df = pd.DataFrame(transactions)

        # Basic analysis
        total_spending = df['amount'].sum() if 'amount' in df.columns else 0 category_spending =
        df.groupby('category')['amount'].sum().to_dict() if
'category' in df.columns else {}
```

```
        insights = {
            'total_spending': total_spending,
            'category_breakdown': category_spending,
            'transaction_count': len(transactions),
            'average_transaction': total_spending / len(transactions) if
transactions else 0
        }

        return insights
```

C.ml_models/nlp_chatbot.py:

```
import pandas as pd
import numpy as np
import re
import json
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
import joblib
import os
import random

class NLPChatbot:
    def __init__(self):
        self.model = None
        self.vectorizer = None
        self.model_path = 'ml_models/saved_models/nlp_chatbot.pkl'
        self.intent_patterns = self._initialize_intent_patterns()
        self.load_or_train_model()

    def _initialize_intent_patterns(self):
        """Define financial intents and their patterns"""
        return {
            'budget_help': {
                'patterns': [
                    'how to create budget', 'make a budget', 'budget planning',
                    'budget help', 'create spending plan', 'monthly budget',
                    'how much should I spend', 'budget allocation', 'budgeting',
                    'financial planning', 'money management', 'expense planning',
                    'create a budget', 'budget setup', 'spending budget'
                ],
```

```
                'responses': [
                    "I can help you create a personalized budget! Please provide your monthly income and I'll generate optimal spending categories using AI analysis.",
                    "Budget planning is essential! I'll analyze your income and expenses to create a balanced budget. What's your monthly income?",
                    "Let me help with budget creation! I use machine learning to recommend spending allocations based on financial best practices and your spending patterns.",
                    "Great! I can create a customized budget for you. To get started, what's your monthly take-home income? I'll then suggest spending categories."
                ]
            },
            'savings_advice': { 'patterns': [
                    'how to save money', 'savings tips', 'save more', 'increase savings', 'savings strategies', 'money saving', 'build emergency fund', 'savings goal', 'save for future', 'saving money', 'financial savings', 'grow savings', 'save more money', 'savings plan', 'emergency fund'
                ],
                'responses': [
                    "Great question! I recommend saving 20% of your income, automating savings transfers, cutting unnecessary subscriptions, and setting specific financial goals.",
                    "Savings strategies that work: 1) Pay yourself first 2) Set specific goals 3) Reduce dining out 4) Use the 24-hour rule for purchases 5) Track all expenses.",
                    "To boost savings effectively: Start with an emergency fund (3-6 months of expenses), then focus on high-yield savings accounts and investment options for long-term growth.",
                    "Based on financial analysis, I suggest: Automate your savings, review recurring expenses monthly, set clear financial goals, and consider the 50-30-20 rule (50% needs, 30% wants, 20% savings)."
                ]
            },
            'spending_analysis': { 'patterns': [
                    'analyze my spending', 'where my money going', 'spending patterns',
                    'review expenses', 'spending breakdown', 'expense analysis', 'how am I spending', 'spending habits', 'track spending', 'money tracking', 'expense tracking', 'financial analysis',
```

```
                    'spending review', 'where does my money go'
                ],
                'responses': [
                    "I can analyze your spending patterns using machine learning! Upload your transaction
data, and I'll categorize expenses, identify trends, and provide optimization suggestions.",
                    "Let me examine your spending! With AI-powered analysis, I can show you
detailed category breakdowns, spending trends over time, and suggest areas for optimization.",
                    "Spending analysis is my specialty! I'll use advanced ML algorithms to categorize
your transactions automatically and provide insights on your financial habits and patterns.",
                    "Perfect! I can perform comprehensive spending analysis.
Please upload your bank statements or transaction data, and I'll provide detailed insights with visual charts and
recommendations."
                ]
            },
            'debt_management': { 'patterns': [
                    'pay off debt', 'debt reduction', 'credit card debt', 'loan repayment', 'get out
                    of debt', 'debt free', 'manage debt', 'debt consolidation', 'reduce debt',
                    'paying off loans', 'credit card payoff', 'debt strategy', 'debt payoff', 'eliminate debt'
                ],
                'responses': [
                    "For effective debt management, I recommend either the snowball method (paying
smallest debts first for motivation) or avalanche method (targeting highest interest debts first for maximum
savings).",
                    "Debt reduction strategy that works: 1) List all debts with amounts and interest
rates 2) Choose your payoff method 3) Make minimum payments on all 4) Apply extra funds to your target
debt 5) Celebrate milestones!",
                    "Consider debt consolidation or balance transfers for high- interest debt. Always
prioritize debts with interest rates above 7-8%. Remember: the average credit card interest is 16-25%, so focus
there first.",
                    "Let me help with debt management! Key strategies include: Creating a debt payoff
plan, negotiating lower interest rates, considering consolidation loans, and building an emergency fund to avoid
new debt."
                ]
            },
            'investment_advice': { 'patterns': [
                    'investment advice', 'where to invest', 'stock market', 'mutual funds', 'retirement
                    planning', 'build wealth',
```

```
                    'investment strategies', 'grow money', 'investing', 'financial investments', 'wealth
                    building', 'retirement
savings',
                    'investment tips', 'grow my money'
                ],
                'responses': [
                    "For smart investing, consider this approach: 1) Build emergency fund first 2)
Maximize employer retirement match 3) Invest in low-cost index funds 4) Diversify across asset classes 5)
Think long-term and avoid emotional decisions.",
                    "Investment basics for beginners: Start with tax-advantaged accounts (401k, IRA),
focus on low-cost diversified funds, understand your risk tolerance, and remember that time in the market
beats timing the market.",
                    "While I can't give specific investment advice, I recommend: Starting with low-cost
index funds (like S&P 500 funds), dollar-cost averaging, and consulting a financial advisor for personalized
investment strategies.",
                    "For wealth building through investments: Consider your time horizon, risk
tolerance, and financial goals. Diversification is key - don't put all your eggs in one basket. Regular
contributions and patience are your best allies."
                ]
            },
            'financial_goals': { 'patterns': [
                    'set financial goals', 'achieve financial freedom', 'financial planning', 'long term
                    goals', 'retirement

planning',                  'buy house', 'save for college', 'financial objectives', 'money goals', 'financial
                    targets', 'goal setting', 'financial milestones', 'plan for future'
                ],
                'responses': [
                    "Setting SMART financial goals (Specific, Measurable, Achievable, Relevant,
Time-bound) is crucial for financial success! Let me help you define and track your objectives.",
                    "Financial goal planning framework: 1) Define clear, specific objectives 2) Break
them into achievable milestones 3) Create actionable steps 4) Set timelines 5) Track progress regularly 6) Adjust
as needed.",
                    "I can help you set and track financial goals! Common goals include: Emergency fund
(3-6 months), retirement savings, down payment for house, education fund, and debt freedom. What are you
working towards?",
                    "For effective financial goal setting: Start with your 'why', make goals specific and
time-bound, automate savings towards them, review progress monthly, and celebrate small wins along the way
to stay motivated."
```

```
                ]
            },
            'expense_tracking': {
                'patterns': [
                    'track expenses', 'monitor spending', 'expense tracking', 'track my money',
                    'spending monitor', 'money management', 'financial tracking', 'budget tracking',
                    'expense
management',
                    'track finances', 'spending tracker'
                ],
                'responses': [
                    "Expense tracking is fundamental to financial health! I recommend: Categorizing
all transactions, reviewing weekly, using budgeting apps, and identifying spending patterns to make informed
decisions.",
                    "For effective expense tracking: Record every transaction, categorize spending, set
monthly limits per category, review patterns regularly, and adjust your budget based on actual spending
behavior.",
                    "I can help you track expenses efficiently! The key is consistency - track daily,
categorize accurately, review weekly, and adjust monthly. This helps identify leaks and opportunities for
savings.",
                    "Smart expense tracking involves: Using digital tools for automation, setting
spending alerts, categorizing transactions (needs vs wants), and analyzing trends to optimize your financial
decisions."
                ]
            },
            'income_optimization': {
                'patterns': [
                    'increase income', 'make more money', 'side income', 'passive income', 'career
                    advancement', 'salary increase', 'income streams', 'extra income', 'financial
                    growth', 'boost income', 'multiple income sources'
                ],
                'responses': [
                    "For income optimization, consider: Developing high-income skills, negotiating
salary increases, exploring side hustles, creating passive income streams, and investing in career
advancement opportunities.",
                    "Strategies to increase income: 1) Upskill in high-demand areas 2) Negotiate your
salary 3) Start a side business 4) Invest in income- generating assets 5) Network for better opportunities 6)
Consider freelance work.",
                    "Building multiple income streams is key to financial security. Options include:
Rental income, dividend investments, online businesses, freelance work, and creating digital products. Start
with one additional stream and grow from there.",
```

```
                "Income optimization involves both active and passive strategies. Focus on: Career advancement, skill development, entrepreneurial ventures, and smart investments. Remember: diversifying income sources reduces financial risk."
            ]
        },
        'greeting': {
            'patterns': [
                'hello', 'hi', 'hey', 'good morning', 'good afternoon', 'what\'s up', 'how are you',
                'greetings', 'good evening', 'hello there', 'hi there', 'hey there'
            ],
            'responses': [
                "Hello! I'm your AI Finance Assistant powered by machine learning. I can help with budgeting, savings analysis, spending optimization, and financial planning! How can I assist you today?",
                "Hi there! I'm here to help you manage your finances better using advanced AI analysis. What would you like to discuss today - budgeting, savings, or financial insights?",
                "Greetings! I'm your financial AI assistant. I can analyze spending patterns, create personalized budgets, provide savings recommendations, and answer your money questions. How can I help you?",
                "Welcome! I'm your AI-powered finance expert. I use machine learning to provide personalized financial advice, budget planning, and spending analysis. What financial topic can I help you with today?"
            ]
        },
        'thanks': {
            'patterns': [
                'thank you', 'thanks', 'appreciate it', 'helpful',
                'great help', 'awesome', 'good job', 'thank you so much', 'thanks a lot', 'much appreciated', 'you\'re helpful'
            ],
            'responses': [
                "You're welcome! Happy to help with your financial journey. Remember, consistent small steps lead to big financial progress over time.", "Glad I could assist! Financial success is a marathon, not a sprint. Let me know if you have more questions about your finances.",
                "You're welcome! Remember the key to financial health: track spending, save consistently, invest wisely, and review your progress regularly.",
                "My pleasure! Financial empowerment comes from understanding and action. Keep learning and taking steps toward your goals. I'm here whenever you need guidance."
            ]
```

```
                },
                'general_help': {
                        'patterns': [
                                'what can you do', 'help me', 'what help', 'assist me', 'what are your features', 'how
                                can you help', 'capabilities', 'what do you do', 'your functions'
                        ],
                        'responses': [
                                "I'm your AI Finance Assistant! I can: Analyze spending patterns, create
personalized budgets, provide savings recommendations, answer financial questions, generate reports, and
help with debt management and investment strategies.",
                                "As your financial AI assistant, I can help with: Expense categorization, budget
creation, savings planning, investment guidance, debt payoff strategies, financial goal setting, and providing
data-driven financial insights.",
                                "My capabilities include: Machine learning-powered expense analysis, personalized
budget recommendations, NLP-based financial advice, data visualization, PDF report generation, and
comprehensive financial planning assistance.",
                                "I'm designed to be your comprehensive financial assistant! I can process
transaction data, categorize spending, suggest optimizations, create budgets, answer money questions, and help
you achieve your financial goals using AI analysis."
                        ]
                }
        }

    def load_or_train_model(self):
        """Load existing NLP model or train a new one""" if
        os.path.exists(self.model_path):
                try:
                        self.model = joblib.load(self.model_path)
                        print("[INFO] Pre-trained NLP chatbot loaded successfully") except Exception as e:
                        print(f"[WARNING] Error loading NLP model: {e}. Training new

model...")                      self.train_nlp_model()

        else:
                print("[INFO] Training new NLP chatbot model...") self.train_nlp_model()

    def train_nlp_model(self):
        """Train the NLP model with financial intent data"""
```

```
            try:
                # Prepare training data
                training_data = [] labels = []

                for intent, data in self.intent_patterns.items(): for pattern in
                    data['patterns']:
                        training_data.append(pattern)
                        labels.append(intent)

                # Add variations and synonyms for better coverage variations = [
                    # Budget variations
                    ('budget', 'budget_help'), ('budgeting', 'budget_help'), ('financial plan', 'budget_help'),
                    ('money plan', 'budget_help'),

                    # Savings variations
                    ('save', 'savings_advice'), ('saving', 'savings_advice'), ('emergency fund',
                    'savings_advice'), ('save cash',
'savings_advice'),

                    # Spending variations
                    ('spending', 'spending_analysis'), ('expenses', 'spending_analysis'),
                    ('where money', 'spending_analysis'), ('money analysis', 'spending_analysis'),

                    # Debt variations
                    ('debt', 'debt_management'), ('loans', 'debt_management'), ('credit cards',
                    'debt_management'), ('pay debt',
'debt_management'),

                    # Investment variations
                    ('invest', 'investment_advice'), ('stocks', 'investment_advice'), ('retirement',
                    'investment_advice'), ('grow wealth',
'investment_advice'),

                    # Goals variations
                    ('goals', 'financial_goals'), ('financial goals', 'financial_goals'),
                    ('future planning', 'financial_goals'), ('money goals', 'financial_goals'),
```

```
        # Tracking variations
        ('track', 'expense_tracking'), ('monitor', 'expense_tracking'), ('money tracking',
        'expense_tracking'),

        # Income variations
        ('income', 'income_optimization'), ('earn more', 'income_optimization'),
        ('make money', 'income_optimization'),

        # Greeting variations
        ('hello', 'greeting'), ('hi', 'greeting'), ('hey', 'greeting'),

        # Thanks variations
        ('thank', 'thanks'), ('appreciate', 'thanks'),

        # Help variations
        ('help', 'general_help'), ('what can you', 'general_help')
]

for word, intent in variations: training_data.append(word)
        labels.append(intent)

# Create and train pipeline with improved parameters self.model =
Pipeline([
        ('tfidf', TfidfVectorizer( ngram_range=(1,
                3), max_features=2000,
                stop_words='english', min_df=2,
                max_df=0.8
        )),
        ('classifier', MultinomialNB(alpha=0.1))
])

self.model.fit(training_data, labels)

# Save the model os.makedirs('ml_models/saved_models',
exist_ok=True) joblib.dump(self.model, self.model_path)
print("[INFO] NLP chatbot model trained and saved successfully")
```

```
            # Print training summary
            print(f"            - Trained on {len(training_data)} examples") print(f"        -
            Covering {len(set(labels))} financial intents")

        except Exception as e:
            print(f"[ERROR] Error training NLP model: {e}") raise e

    def predict_intent(self, message):
        """Predict the intent of a user message""" try:
            if self.model is None: return
                'general_help'

            # Clean the message
            cleaned_message = self.clean_message(message)

            # Predict intent
            intent = self.model.predict([cleaned_message])[0]
            confidence = np.max(self.model.predict_proba([cleaned_message]))

            # Return intent if confidence is high enough, else general help
            if confidence > 0.4:  # Slightly higher threshold for better accuracy return intent
            else:
                return 'general_help'

        except Exception as e:
            print(f"Error in intent prediction: {e}") return 'general_help'

    def clean_message(self, message):
        """Clean and preprocess message for NLP""" try:
            if not isinstance(message, str): return ""

            # Convert to lowercase message =
            message.lower()

            # Remove special characters but keep basic punctuation and numbers message =
            re.sub(r'[^\w\s\?\.]', '', message)
```

```
            # Remove extra whitespace
            message = ' '.join(message.split())

            # Remove common filler words
            filler_words = ['like', 'um', 'uh', 'you know', 'actually', 'basically']
            for word in filler_words:
                message = re.sub(r'\b' + word + r'\b', '', message)

            # Remove extra spaces again
            message = ' '.join(message.split())

            return message.strip()

        except Exception as e:
            print(f"Error cleaning message: {e}") return ""

    def generate_response(self, message, user_context=None): """Generate
        response based on predicted intent""" try:
            intent = self.predict_intent(message)
            confidence = self.get_prediction_confidence(message)

            # Get appropriate response
            if intent in self.intent_patterns:
                responses = self.intent_patterns[intent]['responses'] response =
                random.choice(responses)

                # Add confidence indicator for low confidence predictions if confidence < 0.6:
                    response = self.add_uncertainty_qualifier(response)
            else:
                # Default response for unknown intents response =
                self.get_general_response(message)

            # Add context-aware follow-up if available if user_context:
                response += self.get_contextual_followup(user_context, intent)

            return response
```

```
            except Exception as e:
                print(f"Error generating response: {e}")
                return "I apologize, but I'm having trouble processing your request right now. Could you
please try rephrasing your question?"

    def get_prediction_confidence(self, message): """Get the confidence
        score for a prediction""" try:
            if self.model is None: return 0.0

            cleaned_message = self.clean_message(message) probabilities =
            self.model.predict_proba([cleaned_message]) return np.max(probabilities)

        except Exception as e:
            print(f"Error getting prediction confidence: {e}") return 0.0

    def add_uncertainty_qualifier(self, response):
        """Add qualifier for low-confidence predictions""" qualifiers = [
            " Based on what I understand,", " I think
            this might help:",
            " From what I can tell,",
            " This general advice might be useful:"
        ]
        return random.choice(qualifiers) + " " + response

    def get_general_response(self, message):
        """Generate response for general/unrecognized queries"""
        # Check for financial keywords to provide more relevant responses financial_keywords = {
            'money': "I specialize in money management and financial planning.", 'finance': "I can help with
            various financial topics including
budgeting and investments.",
            'budget': "I'd be happy to help with budget creation and expense
tracking.",
            'save': "I can provide savings strategies and help you build your
emergency fund.",
            'spend': "I can analyze spending patterns and suggest optimizations.",
```

```
                    'debt': "I offer debt management strategies and payoff plans.", 'invest': "I can provide general
                    investment guidance and retirement
planning tips.",
                    'income': "I can help with income optimization and financial growth strategies.",
                    'expense': "I excel at expense tracking and categorization.", 'retirement': "I can assist with
                    retirement planning and long-term
    financial goals."
                }

            message_lower = message.lower()
            for keyword, response in financial_keywords.items(): if keyword in
                    message_lower:
                        general_responses = [
                            f"I understand you're asking about {keyword}. {response} Could you provide
more specific details?",
                            f"It sounds like you're interested in {keyword}. {response} What specific aspect
would you like to explore?",
                            f"Regarding {keyword}, {response.lower()} How can I assist you further with
this?"
                        ]
                        return random.choice(general_responses)

            # Default general responses general_responses = [
                    "I understand you're asking about financial topics. I specialize in budgeting, savings analysis,
spending optimization, and financial planning. Could you provide more specific details about what you'd like
to know?",
                    "As an AI finance assistant, I can help with budget creation, expense analysis, savings strategies,
investment guidance, and financial planning. What specific area would you like to explore?",
                    "I'm here to help with your financial questions! I can analyze spending patterns, create
budgets, suggest savings strategies, provide investment tips, and assist with debt management. What would
you like to discuss?",
                    "I'm your AI Finance Assistant! I specialize in helping with budgeting, savings, spending
analysis, investment strategies, and financial planning. How can I assist with your finances today?"
            ]

            return random.choice(general_responses)

        def get_contextual_followup(self, user_context, intent): """Add context-aware
            follow-up questions"""
```

```
            followups = {
                'budget_help': " Would you like me to create a personalized budget based on your income and spending patterns?",
                'spending_analysis': " Have you uploaded your transaction data for detailed analysis? I can provide visual charts and specific recommendations.",
                'savings_advice': " What are your current savings goals? I can help create a customized savings plan.",
                'debt_management': " Would you like me to help create a detailed debt payoff plan based on your current situation?",
                'investment_advice': " Are you interested in learning about specific investment strategies for your financial goals?",
                'financial_goals': " Would you like to set up a tracking system for your financial goals with milestone planning?",
                'expense_tracking': " I can help you set up an automated expense tracking system. Would you like guidance on getting started?",
                'income_optimization': " Are you looking for specific strategies to increase your income through side hustles or career advancement?"
            }

            return followups.get(intent, "")

        def analyze_financial_sentiment(self, message):
            """Enhanced sentiment analysis for financial context""" try:
                positive_words = [
                    'good', 'great', 'excellent', 'happy', 'proud', 'success', 'improve', 'better', 'awesome',
                    'fantastic', 'amazing', 'progress', 'achieved', 'saved', 'profit', 'gain', 'win'
                ]

                negative_words = [
                    'worried', 'stress', 'struggle', 'difficult', 'problem', 'bad', 'poor', 'debt', 'loss', 'broke',
                    'broke', 'emergency', 'crisis', 'anxious', 'overwhelmed', 'stuck', 'frustrated'
                ]

                concern_words = [
                    'concerned', 'nervous', 'uncertain', 'confused', 'help', 'advice', 'guidance', 'question',
                    'worry', 'anxiety'
                ]

                message_lower = message.lower()
```

```
            positive_count = sum(1 for word in positive_words if word in message_lower)
            negative_count = sum(1 for word in negative_words if word in message_lower)
            concern_count = sum(1 for word in concern_words if word in message_lower)

            if positive_count > negative_count and positive_count > concern_count:
                return "positive"
            elif negative_count > positive_count and negative_count > concern_count:
                return "negative"
            elif concern_count > positive_count and concern_count > negative_count:
                return "concerned" else:
                return "neutral"

        except Exception as e:
            print(f"Error in sentiment analysis: {e}") return "neutral"

    def get_intent_description(self, intent):
        """Get description of what an intent handles"""
        intent_descriptions = {
            'budget_help': "Budget creation and financial planning", 'savings_advice': "Savings
            strategies and emergency funds", 'spending_analysis': "Expense tracking and spending
            patterns", 'debt_management': "Debt payoff strategies and consolidation",
            'investment_advice': "Investment guidance and retirement planning", 'financial_goals': "Goal
            setting and financial milestones", 'expense_tracking': "Expense monitoring and
            categorization", 'income_optimization': "Income growth and multiple streams", 'greeting':
            "Welcome messages and introductions",
            'thanks': "Gratitude and appreciation responses", 'general_help': "General
            assistance and capabilities"
        }
        return intent_descriptions.get(intent, "General financial assistance")

    def get_available_intents(self):
        """Get list of all available intents with descriptions"""
```

```
        return {intent: self.get_intent_description(intent) for intent in
self.intent_patterns.keys()}

# Test function for the chatbot
def test_chatbot():
    """Test function to verify chatbot functionality"""
    chatbot = NLPChatbot()

    test_messages = [
        "How do I create a budget?",
        "I want to save more money",
        "Where is all my money going?",
        "I have credit card debt",
        "Should I invest in stocks?",
        "Hello!",
        "Thank you for your help",
        "What can you do for me?",
        "I'm worried about my finances"
    ]

    print("🤖 Testing NLP Chatbot...")
    print("=" * 50)

    for message in test_messages:
        intent = chatbot.predict_intent(message)
        response = chatbot.generate_response(message)
        sentiment = chatbot.analyze_financial_sentiment(message)
        confidence = chatbot.get_prediction_confidence(message)

        print(f"Input: '{message}'")
        print(f"Intent: {intent} (confidence: {confidence:.2f})")
        print(f"Sentiment: {sentiment}")
        print(f"Response: {response}")
        print("-" * 50)

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

7.utils_data_processor.py:

```
import pandas as pd
import numpy as np
```

```
import json
from datetime import datetime, timedelta import re

class DataProcessor: def _init_
    (self):
        self.supported_formats = ['.csv', '.xlsx', '.xls']

    def process_uploaded_file(self, file_path): """Process uploaded
        financial data file""" try:
            if file_path.endswith('.csv'): df =
                pd.read_csv(file_path)
            elif file_path.endswith(('.xlsx', '.xls')): df =
                pd.read_excel(file_path)
            else:
                return {"error": "Unsupported file format"}

            # Clean and standardize the data df =
            self.clean_financial_data(df)

            # Generate basic insights
            insights = self.generate_insights(df)

            return {
                "success": True,
                "data": df.to_dict('records'), "insights":
                insights, "total_records": len(df)
            }

        except Exception as e:
            return {"error": f"Error processing file: {str(e)}"}

    def clean_financial_data(self, df):
        """Clean and standardize financial data""" # Make a copy
        to avoid modifying original df_clean = df.copy()

        # Standardize column names
        df_clean.columns = [col.lower().strip().replace(' ', '_') for col in df_clean.columns]
```

```
        # Ensure required columns exist or create them if 'description' not
        in df_clean.columns:
            if 'transaction_description' in df_clean.columns: df_clean['description'] =
                df_clean['transaction_description']
            else:
                df_clean['description'] = 'Unknown Transaction'

        if 'amount' not in df_clean.columns: # Try to find
            amount column
            amount_cols = [col for col in df_clean.columns if 'amount' in col or 'transaction' in col]
            if amount_cols:
                df_clean['amount'] = df_clean[amount_cols[0]] else:
                df_clean['amount'] = 0

        # Clean amount values
        df_clean['amount'] = pd.to_numeric(df_clean['amount'], errors='coerce').fillna(0)

        # Clean description text df_clean['description'] =
df_clean['description'].astype(str).str.lower().str.strip()

        # Add date if not present
        if 'date' not in df_clean.columns:
            df_clean['date'] = datetime.now().strftime('%Y-%m-%d')

        return df_clean

    def generate_insights(self, df):
        """Generate basic financial insights from data""" if df.empty:
            return {}

        insights = {
            'total_transactions': len(df), 'total_spending':
            df['amount'].sum(), 'average_transaction':
            df['amount'].mean(), 'largest_transaction':
            df['amount'].max(), 'smallest_transaction':
            df['amount'].min(), 'spending_trend': 'stable'   # Default
            value
```

```
        }

        # Calculate weekly spending if date data is available
        if 'date' in df.columns:
            try:
                df['date'] = pd.to_datetime(df['date'])
                weekly_spending =
df.groupby(df['date'].dt.isocalendar().week)['amount'].sum()
                if len(weekly_spending) > 1:
                    if weekly_spending.iloc[-1] > weekly_spending.iloc[-2] * 1.1:
                        insights['spending_trend'] = 'increasing'
                    elif weekly_spending.iloc[-1] < weekly_spending.iloc[-2] *
0.9:
                        insights['spending_trend'] = 'decreasing'
            except:
                pass

        return insights

    def prepare_for_ml(self, transactions):
        """Prepare transaction data for machine learning"""
        ml_data = []
        for transaction in transactions:
            if isinstance(transaction, dict):
                ml_data.append({
                    'description': transaction.get('description', ''),
                    'amount': transaction.get('amount', 0),
                    'date': transaction.get('date', ''),
                    'original_category': transaction.get('category', '')
                })
        return ml_data
```

8.utils/security.py:

```
import hashlib
import secrets
import string
import re
from datetime import datetime, timedelta
import jwt
import os
```

```
class SecurityManager: def _
    init_(self):
        self.secret_key = os.environ.get('SECRET_KEY', 'finance_assistant_secret_2024')
        self.token_expiry = timedelta(hours=24)

    def hash_password(self, password):
        """Hash password using SHA-256 with salt""" salt =
        secrets.token_hex(16)
        password_hash = hashlib.sha256((password + salt).encode()).hexdigest() return
        f"{salt}${password_hash}"

    def verify_password(self, password, hashed_password): """Verify
        password against hash"""
        try:
            salt, stored_hash = hashed_password.split('$') computed_hash =
            hashlib.sha256((password +
salt).encode()).hexdigest()
            return secrets.compare_digest(computed_hash, stored_hash) except:
            return False

    def generate_auth_token(self, user_id): """Generate JWT token
        for authentication""" payload = {
            'user_id': user_id,
            'exp': datetime.utcnow() + self.token_expiry, 'iat':
            datetime.utcnow()
        }
        return jwt.encode(payload, self.secret_key, algorithm='HS256')

    def verify_auth_token(self, token): """Verify JWT
        token"""
        try:
            payload = jwt.decode(token, self.secret_key, algorithms=['HS256']) return payload['user_id']
        except jwt.ExpiredSignatureError: return None
        except jwt.InvalidTokenError: return None

    def generate_secure_token(self, length=32): """Generate a secure
        random token"""
```

```
        alphabet = string.ascii_letters + string.digits
        return ''.join(secrets.choice(alphabet) for _ in range(length))

    def sanitize_input(self, user_input): """Enhanced
        input sanitization""" if isinstance(user_input,
        str):
            # Remove potentially dangerous characters and scripts sanitized =
            re.sub(r'<script.*?</script>', '', user_input,
flags=re.IGNORECASE | re.DOTALL)
            sanitized = sanitized.replace('<', '&lt;').replace('>', '&gt;')
            sanitized = sanitized.replace('"', '&quot;').replace("'", '&#x27;')
            sanitized = sanitized.replace('(', '&#40;').replace(')', '&#41;') sanitized = re.sub(r'on\w+=',
            '', sanitized, flags=re.IGNORECASE) return sanitized.strip()
        return user_input

    def validate_email(self, email): """Validate email
        format"""
        pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$' return re.match(pattern,
        email) is not None

    def validate_transaction_data(self, transaction): """Enhanced
        transaction data validation"""
        if not isinstance(transaction, dict): return False

        required_fields = ['description', 'amount'] for field in
        required_fields:
            if field not in transaction: return False

        # Validate amount is numeric and reasonable try:
            amount = float(transaction['amount'])
            if amount < -1000000 or amount > 1000000:  # Reasonable limits return False
        except (ValueError, TypeError): return
            False

        # Validate description length and content description =
        str(transaction['description'])
        if len(description) > 500 or len(description.strip()) == 0: return False
```

```
        # Validate date if present
        if 'date' in transaction:
            try:
                datetime.strptime(transaction['date'], '%Y-%m-%d')
            except ValueError:
                return False

        return True

    def check_file_safety(self, filename):
        """Check if uploaded file is safe"""
        allowed_extensions = {'.csv', '.xlsx', '.xls'}
        allowed_mime_types = {
            'text/csv',
            'application/vnd.ms-excel',
            'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet'
        }

        file_ext = os.path.splitext(filename)[1].lower()
        return file_ext in allowed_extensions
```

3.utils/visualization.py:

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
from io import BytesIO
import base64
import os
from datetime import datetime
import matplotlib
matplotlib.use('Agg')  # Set backend to avoid GUI issues

class FinanceVisualizer:
    def __init__(self):
        plt.style.use('seaborn-v0_8')
        self.colors = ['#FF6B6B', '#4ECDC4', '#45B7D1', '#96CEB4', '#FFEAA7',
'#DDA0DD', '#98D8C8', '#F7DC6F']
```

```
    def create_spending_pie_chart(self, category_data, title="Spending by Category"):
        """Create a pie chart for spending categories""" try:
            fig, ax = plt.subplots(figsize=(10, 8))

            # Prepare data
            categories = list(category_data.keys()) amounts =
            list(category_data.values())

            # Create pie chart
            wedges, texts, autotexts = ax.pie( amounts,
                labels=categories,
                autopct='%1.1f%%',
                startangle=90,
                colors=self.colors[:len(categories)], shadow=True
            )

            # Style the text
            for autotext in autotexts: autotext.set_color('white')
                autotext.set_fontweight('bold')

            ax.set_title(title, fontsize=16, fontweight='bold', pad=20) plt.tight_layout()

            # Convert to base64 chart_buffer =
            BytesIO()
            plt.savefig(chart_buffer, format='png', dpi=100, bbox_inches='tight') chart_buffer.seek(0)
            chart_base64 = base64.b64encode(chart_buffer.getvalue()).decode('utf- plt.close()

8')

            return f"{chart_base64}"

        except Exception as e:
            print(f"Error creating pie chart: {e}") return None
```

```
    def create_spending_trend_chart(self, transactions_data, title="Spending Trend"):
        """Create a line chart for spending trends over time""" try:
            if not transactions_data: return None

            df = pd.DataFrame(transactions_data)

            # Convert date and aggregate by week if 'date' in
            df.columns:
                df['date'] = pd.to_datetime(df['date']) df_weekly =
df.groupby(df['date'].dt.to_period('W'))['amount'].sum().reset_index() df_weekly['date'] =
                df_weekly['date'].dt.start_time

                fig, ax = plt.subplots(figsize=(12, 6)) ax.plot(df_weekly['date'], df_weekly['amount'],
                marker='o',
linewidth=2, markersize=6, color='#4ECDC4')
                ax.fill_between(df_weekly['date'], df_weekly['amount'], alpha=0.3, color='#4ECDC4')

                ax.set_title(title, fontsize=16, fontweight='bold') ax.set_xlabel('Date')
                ax.set_ylabel('Amount ($)') ax.grid(True,
                alpha=0.3)

                plt.xticks(rotation=45) plt.tight_layout()

                # Convert to base64 chart_buffer =
                BytesIO()
                plt.savefig(chart_buffer, format='png', dpi=100, bbox_inches='tight')
                chart_buffer.seek(0) chart_base64 =
base64.b64encode(chart_buffer.getvalue()).decode('utf-8') plt.close()

                    return f"{chart_base64}"

        except Exception as e:
            print(f"Error creating trend chart: {e}")
```

```
                return None

            return None

    def create_category_bar_chart(self, category_data, title="Spending by Category"):
        """Create a bar chart for category spending""" try:
            fig, ax = plt.subplots(figsize=(12, 6))

            categories = list(category_data.keys()) amounts =
            list(category_data.values())

            bars = ax.bar(categories, amounts, color=self.colors[:len(categories)], alpha=0.8)

            # Add value labels on bars
            for bar, amount in zip(bars, amounts): height =
                bar.get_height()
                ax.text(bar.get_x() + bar.get_width()/2., height + max(amounts)*0.01,
                        f'${amount:.2f}', ha='center', va='bottom',
fontweight='bold')

            ax.set_title(title, fontsize=16, fontweight='bold')
            ax.set_ylabel('Amount ($)') plt.xticks(rotation=45, ha='right')
            plt.tight_layout()

            # Convert to base64 chart_buffer =
            BytesIO()
            plt.savefig(chart_buffer, format='png', dpi=100, bbox_inches='tight') chart_buffer.seek(0)
            chart_base64 = base64.b64encode(chart_buffer.getvalue()).decode('utf- plt.close()

8')

            return f"{chart_base64}"

        except Exception as e:
            print(f"Error creating bar chart: {e}") return None
```

```
    def create_budget_comparison_chart(self, current_spending, recommended_budget, title="Budget vs
Actual Spending"):
        """Create a comparison chart between current spending and recommended budget"""
        try:
            categories = ['Essential', 'Discretionary', 'Savings']

            fig, ax = plt.subplots(figsize=(10, 6))

            x = np.arange(len(categories)) width = 0.35

            # Use sample data if actual spending not available if not
            current_spending:
                current_spending = [recommended_budget['essential_spending'] *
1.1,
                                    recommended_budget['discretionary_spending'] *
0.8,
                                    recommended_budget['savings'] * 0.7]

            ax.bar(x - width/2, current_spending, width, label='Current', color='#FF6B6B', alpha=0.8)
            ax.bar(x + width/2, [recommended_budget['essential_spending'],
                                 recommended_budget['discretionary_spending'],
                                 recommended_budget['savings']],
                   width, label='Recommended', color='#4ECDC4', alpha=0.8)

            ax.set_title(title, fontsize=16, fontweight='bold') ax.set_ylabel('Amount ($)')
            ax.set_xticks(x) ax.set_xticklabels(categories)
            ax.legend()
            ax.grid(True, alpha=0.3)

            plt.tight_layout()

            # Convert to base64 chart_buffer =
            BytesIO()
            plt.savefig(chart_buffer, format='png', dpi=100, bbox_inches='tight') chart_buffer.seek(0)
            chart_base64 = base64.b64encode(chart_buffer.getvalue()).decode('utf-

8')
```

```
            plt.close()

            return f"{chart_base64}"

        except Exception as e:
            print(f"Error creating budget comparison chart: {e}")
            return None

    def generate_dashboard_charts(self, transactions_data, category_data,
budget_data=None):
        """Generate all charts for dashboard"""
        charts = {}

        # Pie chart for categories
        charts['pie_chart'] = self.create_spending_pie_chart(category_data)

        # Bar chart for categories
        charts['bar_chart'] = self.create_category_bar_chart(category_data)

        # Trend chart
        charts['trend_chart'] =
self.create_spending_trend_chart(transactions_data)

        # Budget comparison if available
        if budget_data:
            charts['budget_chart'] = self.create_budget_comparison_chart(None,
budget_data)

        return charts
```

10. utils/report_generator.py:

```
from reportlab.lib.pagesizes import letter, A4
from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer, Table,
TableStyle, Image
from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle
from reportlab.lib.units import inch
from reportlab.lib import colors
from reportlab.graphics.shapes import Drawing
from reportlab.graphics.charts.piecharts import Pie
from reportlab.graphics.charts.barcharts import VerticalBarChart
```

```
from reportlab.graphics import renderPDF import
pandas as pd
from datetime import datetime import os
from io import BytesIO import
base64

class PDFReportGenerator: def _init_
    (self):
        self.styles = getSampleStyleSheet() self.title_style =
        ParagraphStyle(
            'CustomTitle', parent=self.styles['Heading1'],
            fontSize=18,
            spaceAfter=30, textColor=colors.HexColor('#2C3E50')
        )
        self.heading_style = ParagraphStyle(
            'CustomHeading',
            parent=self.styles['Heading2'], fontSize=14,
            spaceAfter=12, textColor=colors.HexColor('#34495E')
        )
        self.normal_style = ParagraphStyle(
            'CustomNormal', parent=self.styles['Normal'],
            fontSize=10,
            spaceAfter=6, textColor=colors.HexColor('#2C3E50')
        )

    def generate_financial_report(self, analysis_data, output_path="financial_report.pdf"):
        """Generate a comprehensive PDF financial report""" try:
            doc = SimpleDocTemplate(output_path, pagesize=A4, topMargin=0.5*inch) story = []

            # Title
            title = Paragraph("AI-Powered Financial Analysis Report", self.title_style)
            story.append(title)
```

```
                story.append(Spacer(1, 0.2*inch))

                # Date
                date_str = datetime.now().strftime("%B %d, %Y")
                date_para = Paragraph(f"Generated on: {date_str}", self.normal_style) story.append(date_para)
                story.append(Spacer(1, 0.3*inch))

                # Executive Summary
                story.append(Paragraph("Executive Summary", self.heading_style)) summary_text =
                self._generate_executive_summary(analysis_data) story.append(Paragraph(summary_text,
                self.normal_style)) story.append(Spacer(1, 0.2*inch))

                # Key Metrics
                story.append(Paragraph("Key Financial Metrics", self.heading_style)) metrics_table =
                self._create_metrics_table(analysis_data) story.append(metrics_table)
                story.append(Spacer(1, 0.2*inch))

                # Spending Analysis
                story.append(Paragraph("Spending Analysis", self.heading_style)) spending_text =
                self._generate_spending_analysis(analysis_data) story.append(Paragraph(spending_text,
                self.normal_style)) story.append(Spacer(1, 0.2*inch))

                # Category Breakdown
                if 'spending_analysis' in analysis_data and 'category_breakdown' in
analysis_data['spending_analysis']:
                    story.append(Paragraph("Category Breakdown", self.heading_style)) category_table =
self._create_category_table(analysis_data['spending_analysis']['category_breakdow n'])
                    story.append(category_table) story.append(Spacer(1,
                    0.2*inch))

                # Budget Recommendations
                if 'budget_recommendations' in analysis_data: story.append(Paragraph("Budget
                    Recommendations",
self.heading_style))
                    budget_text =
self._generate_budget_recommendations(analysis_data['budget_recommendations'])
                    story.append(Paragraph(budget_text, self.normal_style))
```

```
                    story.append(Spacer(1, 0.2*inch))

                    # Budget Table
                    budget_table =
self._create_budget_table(analysis_data['budget_recommendations']) story.append(budget_table)
                    story.append(Spacer(1, 0.2*inch))

                # Financial Advice
                story.append(Paragraph("Financial Health Tips", self.heading_style)) advice_text =
                self._generate_financial_advice(analysis_data) story.append(Paragraph(advice_text,
                self.normal_style))

                # Build PDF
                doc.build(story) return True

            except Exception as e:
                print(f"Error generating PDF report: {e}") return False

    def _generate_executive_summary(self, analysis_data): """Generate executive
        summary text"""
        spending_analysis = analysis_data.get('spending_analysis', {}) total_spending =
        spending_analysis.get('total_spending', 0) transaction_count =
        spending_analysis.get('transaction_count', 0)

        summary = f"""
        This report analyzes {transaction_count} transactions with total spending of ${total_spending:,.2f}.
        The AI-powered analysis provides insights into spending patterns, category distribution, and
personalized
        recommendations for improving your financial health. """
        return summary

    def _create_metrics_table(self, analysis_data): """Create key metrics
        table"""
        spending_analysis = analysis_data.get('spending_analysis', {}) insights =
        analysis_data.get('insights', {})

        data = [
            ['Metric', 'Value'],
```

```
                    ['Total Transactions', spending_analysis.get('transaction_count', ['Total Spending',
0)],
                    f"${spending_analysis.get('total_spending',
0):,.2f}"],
                    ['Average Transaction',

f"${spending_analysis.get('average_transaction', 0):.2f}"],
                    ['Largest Transaction', f"${insights.get('largest_transaction',
0):.2f}"],
                    ['Spending Trend', insights.get('spending_trend', 'N/A')]
                ]



            table = Table(data, colWidths=[2.5*inch, 2*inch]) table.setStyle(TableStyle([
                    ('BACKGROUND', (0, 0), (-1, 0), colors.HexColor('#34495E')),
                    ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
                    ('ALIGN', (0, 0), (-1, -1), 'LEFT'),
                    ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
                    ('FONTSIZE', (0, 0), (-1, 0), 12),
                    ('BOTTOMPADDING', (0, 0), (-1, 0), 12),
                    ('BACKGROUND', (0, 1), (-1, -1), colors.HexColor('#ECF0F1')),
                    ('GRID', (0, 0), (-1, -1), 1, colors.HexColor('#BDC3C7'))
            ]))

            return table

        def _generate_spending_analysis(self, analysis_data): """Generate spending analysis
            text"""
            spending_analysis = analysis_data.get('spending_analysis', {}) category_breakdown =
            spending_analysis.get('category_breakdown', {})

            if category_breakdown:
                    top_category = max(category_breakdown, key=category_breakdown.get) top_amount =
                    category_breakdown[top_category]
                    analysis_text = f"""
                    Your spending is primarily concentrated in {top_category} (${top_amount:,.2f}).
                    The analysis shows a balanced distribution across
{len(category_breakdown)} major spending categories.
                    Consider reviewing your expenses in the highest spending categories for potential
optimization.
                    """
            else:
```

```
                analysis_text = "Spending analysis shows consistent patterns across multiple categories. Regular
monitoring is recommended."

            return analysis_text

        def _create_category_table(self, category_data): """Create category
            breakdown table"""
            if not category_data:
                return Paragraph("No category data available", self.normal_style)

            data = [['Category', 'Amount', 'Percentage']] total =
            sum(category_data.values())

            for category, amount in sorted(category_data.items(), key=lambda x: x[1], reverse=True):
                percentage = (amount / total) * 100 if total > 0 else 0 data.append([category, f"${amount:,.2f}",
                f"{percentage:.1f}%"])

            table = Table(data, colWidths=[2*inch, 1.5*inch, 1*inch]) table.setStyle(TableStyle([
                ('BACKGROUND', (0, 0), (-1, 0), colors.HexColor('#3498DB')),
                ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
                ('ALIGN', (1, 0), (-1, -1), 'RIGHT'),
                ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
                ('FONTSIZE', (0, 0), (-1, 0), 10),
                ('BACKGROUND', (0, 1), (-1, -1), colors.HexColor('#F8F9FA')),
                ('GRID', (0, 0), (-1, -1), 1, colors.HexColor('#DEE2E6'))
            ]))

            return table

        def _generate_budget_recommendations(self, budget_data): """Generate budget
            recommendations text"""
            advice = budget_data.get('advice', '') tip =
            budget_data.get('tip', '')

            return f"""
            {advice} {tip} The recommended budget allocation follows proven financial principles
            to help you achieve your savings goals while maintaining a balanced lifestyle.
            """
```

```
    def _create_budget_table(self, budget_data): """Create budget
        recommendations table""" data = [
            ['Category', 'Recommended Amount', 'Purpose'],
            ['Essential Spending', f"${budget_data.get('essential_spending', 0):,.2f}", 'Necessary living
expenses'],
            ['Discretionary Spending', f"${budget_data.get('discretionary_spending', 0):,.2f}", 'Lifestyle
and entertainment'],
            ['Savings & Investments', f"${budget_data.get('savings', 0):,.2f}", 'Future financial goals']
        ]

        table = Table(data, colWidths=[2*inch, 1.5*inch, 2*inch]) table.setStyle(TableStyle([
            ('BACKGROUND', (0, 0), (-1, 0), colors.HexColor('#27AE60')),
            ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
            ('ALIGN', (1, 0), (1, -1), 'RIGHT'),
            ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
            ('BACKGROUND', (0, 1), (-1, -1), colors.HexColor('#E8F5E8')),
            ('GRID', (0, 0), (-1, -1), 1, colors.HexColor('#C8E6C9'))
        ]))

        return table

    def _generate_financial_advice(self, analysis_data): """Generate financial advice
        section"""
        advice = """
        <b>Financial Health Tips:</b>
        <br/><br/>
        • <b>Track Regularly:</b> Monitor your spending weekly to stay on track
        <br/>
        • <b>Automate Savings:</b> Set up automatic transfers to savings accounts
        <br/>
        • <b>Review Subscriptions:</b> Cancel unused services and memberships
        <br/>
        • <b>Emergency Fund:</b> Maintain 3-6 months of expenses in liquid
savings
        <br/>
        • <b>Debt Management:</b> Prioritize high-interest debt repayment
        <br/>
```

```
        • <b>Invest Early:</b> Take advantage of compound growth for long-term
goals
        <br/><br/>
        Remember: Small, consistent improvements lead to significant financial
progress over time.
        """

        return advice
```

11. utils/error_handler.py:

```
import logging
from flask import jsonify, request
import traceback
import sys

class ErrorHandler:
    def __init__(self, app=None):
        self.app = app
        self.setup_logging()

    def setup_logging(self):
        """Setup application logging"""
        logging.basicConfig(
            level=logging.INFO,

format='%(asctime)s %(levelname)s %(name)s %(threadName)s : %(message)s',
            handlers=[
                logging.FileHandler('finance_assistant.log'),
                logging.StreamHandler(sys.stdout)
            ]
        )

        self.logger = logging.getLogger(__name__)

    def register_handlers(self, app):
        """Register error handlers with Flask app"""
        self.app = app

        @app.errorhandler(400)
        def bad_request(error):
            return jsonify({
```

```
                'status': 'error',
                'message': 'Bad request - invalid input data', 'code': 400
        }), 400

@app.errorhandler(404) def
not_found(error):
        return jsonify({ 'status':'error',
                'message': 'Resource not found', 'code': 404
        }), 404

@app.errorhandler(405)
def method_not_allowed(error): return
        jsonify({
                'status': 'error',
                'message': 'Method not allowed', 'code': 405
        }), 405

@app.errorhandler(500)
def internal_server_error(error): self.logger.error(f"500 Error:
        {str(error)}") self.logger.error(traceback.format_exc())

        return jsonify({ 'status':'error',
                'message': 'Internal server error', 'code': 500
        }), 500

@app.errorhandler(Exception)
def handle_unexpected_error(error): self.logger.error(f"Unexpected error:
        {str(error)}") self.logger.error(traceback.format_exc())

        return jsonify({ 'status':'error',
                'message': 'An unexpected error occurred', 'code': 500
        }), 500
```

```
    def log_api_request(self):
        """Log API requests for debugging"""
        if self.app and self.app.debug:
            self.logger.info(f"API Request: {request.method} {request.path} -
{request.remote_addr}")

    def log_api_response(self, response):
        """Log API responses for debugging"""
        if self.app and self.app.debug:
            self.logger.info(f"API Response: {response.status_code}")
        return response
```

12. template/base.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>AI Finance Assistant</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet">
    <link href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.0.0/css/all.min.css" rel="stylesheet">
    <link href="{{ url_for('static', filename='css/style.css') }}"
rel="stylesheet">
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-dark bg-primary">
        <div class="container">
            <a class="navbar-brand" href="/">
                <i class="fas fa-robot"></i> AI Finance Assistant
            </a>
            <div class="navbar-nav ms-auto">
                <a class="nav-link" href="/">
                    <i class="fas fa-tachometer-alt"></i> Dashboard
                </a>
                <a class="nav-link" href="/chat">
                    <i class="fas fa-robot"></i> Chat Assistant
                </a>
            </div>
        </div>
```

```
    </nav>

    <div class="container mt-4">
        {% block content %}{% endblock %}
    </div>

    <footer class="bg-light text-center py-3 mt-5">
        <div class="container">
            <p class="mb-0 text-muted">
                <i class="fas fa-brain"></i> Powered by AI & Machine Learning |
                AI Finance Assistant v1.0
            </p>
        </div>
    </footer>

    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js
"></script>
    <script src="{{ url_for('static', filename='js/main.js') }}"></script>
</body>
</html>
```

13. template/dashboard.html:

```
{% extends "base.html" %}

{% block content %}
<div class="row">
    <div class="col-md-12">
        <div class="card">
            <div class="card-header bg-primary text-white">
                <h4><i class="fas fa-tachometer-alt"></i> Financial
Dashboard</h4>
            </div>
            <div class="card-body">
                <div class="alert alert-info">
                    <i class="fas fa-info-circle"></i>
                    Welcome to your AI Finance Assistant! Upload your transaction
data to get AI-powered insights, budgeting recommendations, and financial
analysis.
                </div>
```

```
                    <div class="row">
                        <div class="col-md-6">
                            <div class="card mb-4">
                                <div class="card-body text-center">
                                    <h5><i class="fas fa-upload text-primary"></i>
Upload Financial Data</h5>
                                    <p class="text-muted">Upload CSV or Excel files
with your transaction data</p>
                                    <input type="file" id="fileUpload" class="form-
control mb-3" accept=".csv,.xlsx,.xls">
                                    <button class="btn btn-primary w-100"
onclick="uploadFile()">
                                        <i class="fas fa-cloud-upload-alt"></i>
Upload & Analyze
                                    </button>
                                    <small class="text-muted mt-2 d-block"> Supported formats:
                                        CSV, Excel (.xlsx, .xls)
                                    </small>
                                </div>
                            </div>
                        </div>

                        <div class="col-md-6">
                            <div class="card mb-4">
                                <div class="card-body text-center">
                                    <h5><i class="fas fa-robot text-success"></i> AI
Assistant</h5>
                                    <p class="text-muted">Get personalized financial
advice and answers</p>
                                        <a href="/chat" class="btn btn-success w-100">
                                            <i class="fas fa-comments"></i> Start Chat
                                    </a>
                                    <small class="text-muted mt-2 d-block">
                                        Ask about budgets, savings, investments, and
more
                                    </small>
                                </div>
                            </div>
                        </div>
                    </div>

                    <div id="analysisResults" class="mt-4" style="display: none;">
                        <!-- Analysis results will be displayed here -->
                    </div>
```

```
                <!-- Sample Data Section -->
                <div class="card mt-4">
                    <div class="card-header bg-light">
                        <h6><i class="fas fa-download"></i> Need Sample
Data?</h6>
                    </div>
                    <div class="card-body">
                        <p>Download our sample CSV file to test the
application:</p>
                        <button class="btn btn-outline-primary"
onclick="downloadSampleData()">
                            <i class="fas fa-file-csv"></i> Download Sample CSV
                        </button>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>

<!-- Loading Spinner -->
<div id="loadingSpinner" class="loading-spinner text-center mt-4" style="display:
none;">
    <div class="spinner-border text-primary" style="width: 3rem; height: 3rem;"
role="status">
        <span class="visually-hidden">Loading...</span>
    </div>
    <p class="mt-2">AI is analyzing your financial data...</p>
</div>
{% endblock %}
```

14. template/chat.html:

```
{% extends "base.html" %}

{% block content %}
<div class="row">
    <div class="col-md-12">
        <div class="card">
            <div class="card-header bg-success text-white">
                <h4><i class="fas fa-robot"></i> AI Finance Assistant Chat</h4>
```

```
                </div>
                <div class="card-body">
                    <div class="row">
                        <div class="col-md-8">
                            <div id="chatContainer" class="chat-container">
                                <div class="chat-message bot-message">
                                    <strong><i class="fas fa-robot"></i> Finance
AI:</strong><br>
                                    Hello! I'm your AI Finance Assistant powered by
machine learning. I can help you analyze spending, create budgets, provide savings recommendations, and
answer financial questions. How can I assist you today?
                                </div>
                            </div>

                            <div class="input-group mt-3">
                                <input type="text" id="userInput" class="form- control"
placeholder="Ask about budgets, savings, investments, or financial advice..."
onkeypress="handleKeyPress(event)">
                                <button class="btn btn-success"
onclick="sendMessage()">
                                    <i class="fas fa-paper-plane"></i> Send
                                </button>
                            </div>

                            <div class="mt-3">
                                <small class="text-muted">
                                    <i class="fas fa-lightbulb"></i> Try asking: "How can I save
                                    more money?" or "Create a budget
for $5000 monthly income" or "Analyze my spending patterns"
                                </small>
                            </div>
                        </div>

                        <div class="col-md-4">
                            <div class="card">
                                <div class="card-header bg-info text-white">
                                    <h6><i class="fas fa-bolt"></i> Quick
Actions</h6>
                                </div>
                                <div class="card-body">
                                    <button class="btn btn-outline-primary btn-sm w-
100 mb-2" onclick="quickQuestion('How to create a budget?')">
```

```
                                    <i class="fas fa-coins"></i> Budget Help
                                </button>
                                <button class="btn btn-outline-success btn-sm w-
100 mb-2" onclick="quickQuestion('Savings tips')">
                                    <i class="fas fa-piggy-bank"></i> Savings
Tips
                                </button>
                                <button class="btn btn-outline-info btn-sm w-100
mb-2" onclick="quickQuestion('Analyze my spending')">
                                    <i class="fas fa-chart-pie"></i> Spending
Analysis
                                </button>
                                <button class="btn btn-outline-warning btn-sm w-
100 mb-2" onclick="quickQuestion('Investment advice')">
                                    <i class="fas fa-chart-line"></i> Investments
                                </button>
                                <button class="btn btn-outline-danger btn-sm w-
100 mb-2" onclick="quickQuestion('Debt management')">
                                    <i class="fas fa-credit-card"></i> Debt Help
                                </button>
                                <button class="btn btn-outline-secondary btn-sm w-100 mb-2"
onclick="quickQuestion('Financial goals')">
                                    <i class="fas fa-bullseye"></i> Goal Setting
                                </button>

                                <hr>
                                <div class="text-center">
                                    <a href="/" class="btn btn-secondary btn-sm">
                                        <i class="fas fa-arrow-left"></i> Back to
Dashboard
                                    </a>
                                </div>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>

    <!-- Loading Spinner -->
    <div id="loadingSpinner" class="loading-spinner" style="display: none;">
        <div class="text-center">
```

```
        <div class="spinner-border text-success" role="status">
            <span class="visually-hidden">Loading...</span>
        </div>
        <p class="mt-2">AI is thinking...</p>
    </div>
</div>
{% endblock %}
```

15. static/css/style.css:

```
body {
    background-color: #f8f9fa;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    line-height: 1.6;
}

.card {
    box-shadow: 0 0.125rem 0.5rem rgba(0, 0, 0, 0.1);
    border: 1px solid rgba(0, 0, 0, 0.125);
    margin-bottom: 1rem;
    border-radius: 0.5rem;
}

.card-header {
    border-radius: 0.5rem 0.5rem 0 0 !important;
    font-weight: 600;
}

.navbar-brand {
    font-weight: 700;
    font-size: 1.5rem;
}

.navbar-brand i {
    margin-right: 0.5rem;
}

.chat-container {
    height: 400px;
    overflow-y: auto;
    border: 1px solid #dee2e6;
    border-radius: 0.5rem;
```

```
        padding: 1rem;
        background-color: #f8f9fa;
        background-image: linear-gradient(rgba(255,255,255,0.9), rgba(255,255,255,0.9)),
                                    url('data:image/svg+xml,<svg xmlns="http://www.w3.org/2000/svg"
viewBox="0 0 100 100" fill="%23007bff" opacity="0.03"><circle cx="50" cy="50" r="2"/></svg>');
}

.chat-message {
        margin-bottom: 1rem; padding:
        0.75rem 1rem; border-radius:
        0.75rem; position: relative;
        animation: fadeIn 0.3s ease-in;
}

@keyframes fadeIn {
        from { opacity: 0; transform: translateY(10px); } to { opacity: 1;
        transform: translateY(0); }
}

.user-message {
        background: linear-gradient(135deg, #007bff, #0056b3); color: white;
        margin-left: 20%;
        border-bottom-right-radius: 0.25rem;
}

.bot-message {
        background-color: white; color: #333;
        margin-right: 20%;
        border: 1px solid #e9ecef;
        border-bottom-left-radius: 0.25rem;
}

.loading-spinner { text-align:
        center; padding: 1rem;
        background: rgba(255, 255, 255, 0.9); border-
        radius: 0.5rem;
        margin: 1rem 0;
```

```
}

.progress {
        height: 8px;
        margin-top: 0.5rem;
}

.badge {
        font-size: 0.75em; padding:
        0.35em 0.65em;
}

.btn {
        border-radius: 0.375rem; font-
        weight: 500;
        transition: all 0.2s ease-in-out;
}

.btn:hover {
        transform: translateY(-1px);
        box-shadow: 0 0.25rem 0.5rem rgba(0, 0, 0, 0.1);
}

.alert {
        border-radius: 0.5rem; border:
        none;
}

.table th {
        border-top: none; font-
        weight: 600;
        background-color: #f8f9fa;
}

.input-group {
        border-radius: 0.5rem;
        overflow: hidden;
}

.form-control {
        border-radius: 0.375rem;
}
```

```
.form-control:focus {
    box-shadow: 0 0 0 0.2rem rgba(0, 123, 255, 0.25); border-color:
    #007bff;
}

/* Budget recommendation styles */
.budget-recommendation {
    background: linear-gradient(135deg, #f8f9fa, #e9ecef); border-radius:
    0.5rem;
    padding: 1rem;
    border-left: 4px solid #28a745;
}

.budget-breakdown { display:
    grid;
    grid-template-columns: repeat(auto-fit, minmax(150px, 1fr)); gap: 1rem;
    margin: 1rem 0;
}

.budget-item { padding:
    1rem;
    border-radius: 0.375rem; text-
    align: center;
}

.budget-item.essential { background-
    color: #28a745; color: white;
}

.budget-item.discretionary {
    background-color: #ffc107; color:
    #212529;
}

.budget-item.savings { background-
    color: #17a2b8; color: white;
}

.budget-item h4 {
```

```
        margin: 0;
        font-weight: bold;
}

.budget-item small { opacity:
        0.9;
}

.advice-section { background:
        white; padding: 1rem;
        border-radius: 0.375rem;
        margin-top: 1rem;
}

/* Chart container styles */
.chart-container { background:
        white; padding: 1rem;
        border-radius: 0.5rem; margin:
        1rem 0;
        box-shadow: 0 0.125rem 0.25rem rgba(0, 0, 0, 0.075);
}

.chart-container img { max-
        width: 100%; height:
        auto;
        border-radius: 0.375rem;
}

/* Responsive adjustments */ @media (max-
width: 768px) {
        .user-message { margin-left:
                10%;
        }

        .bot-message {
                margin-right: 10%;
        }

        .chat-container { height: 300px;
        }
```

```
    .budget-breakdown {
        grid-template-columns: 1fr;
    }
}

/* Custom scrollbar for chat */
.chat-container::-webkit-scrollbar {
    width: 6px;
}

.chat-container::-webkit-scrollbar-track {
    background: #f1f1f1;
    border-radius: 3px;
}

.chat-container::-webkit-scrollbar-thumb {
    background: #c1c1c1;
    border-radius: 3px;
}

.chat-container::-webkit-scrollbar-thumb:hover {
    background: #a8a8a8;
}

/* Export buttons */
.export-buttons {
    display: flex;
    gap: 0.5rem;
    flex-wrap: wrap;
}

.export-buttons .btn {
    flex: 1;
    min-width: 120px;
}
```

1C.static/js/main.js:

```
// Enhanced main.js with complete functionality
```

```
// File upload functionality function
uploadFile() {
        const fileInput = document.getElementById('fileUpload'); const file =
        fileInput.files[0];

        if (!file) {
                alert('Please select a CSV or Excel file first!'); return;
        }

        // Validate file type
        const validExtensions = ['.csv', '.xlsx', '.xls']; const fileExtension =
file.name.toLowerCase().substring(file.name.lastIndexOf('.')); if
        (!validExtensions.includes(fileExtension)) {
                alert('Please upload a CSV or Excel file (.csv, .xlsx, .xls)'); return;
        }

        const formData = new FormData();
        formData.append('file', file);

        // Show loading state
        const resultsDiv = document.getElementById('analysisResults'); const loadingSpinner =
        document.getElementById('loadingSpinner');

        resultsDiv.style.display = 'none';
        loadingSpinner.style.display = 'block';

        // Make API call to upload and analyze
        fetch('/api/upload-transactions', {
                method: 'POST', body:
                formData
        })
        .then(response => response.json())
        .then(data => { loadingSpinner.style.display = 'none';

                if (data.status === 'success') {
                        displayAnalysisResults(data);
                } else {
                        resultsDiv.innerHTML = `
                                <div class="alert alert-danger">
```

```
                        <h5><i class="fas fa-exclamation-triangle"></i> Analysis
Failed</h5>
                        <p>${data.message || 'Error processing your file'}</p>
                    </div>
                `;

                resultsDiv.style.display = 'block';
            }
        })
        .catch(error => { loadingSpinner.style.display = 'none';
            console.error('Error:', error); resultsDiv.innerHTML
            = `
                <div class="alert alert-danger">
                    <h5><i class="fas fa-exclamation-triangle"></i> Upload
Failed</h5>
                    <p>Error uploading file: ${error.message}</p>
                </div>
            `;
            resultsDiv.style.display = 'block';
        });
}

function displayAnalysisResults(data) {
    const resultsDiv = document.getElementById('analysisResults');

    // Basic insights
    const insights = data.insights || {};
    const spendingAnalysis = data.spending_analysis || {}; const charts =
    data.charts || {};

    let resultsHTML = `
        <div class="alert alert-success">
            <h4><i class="fas fa-chart-bar"></i> AI Analysis Complete!</h4>
            <p>Successfully analyzed ${data.total_records} transactions</p>
        </div>

        <div class="row">
            <div class="col-md-6">
                <div class="card mb-3">
                    <div class="card-header bg-primary text-white">
                        <h6><i class="fas fa-chart-pie"></i> Spending
Overview</h6>
                    </div>
```

```
                                <div class="card-body">
                                        <p><strong>Total Spending:</strong>
$${spendingAnalysis.total_spending?.toFixed(2) || '0.00'}</p>
                                        <p><strong>Average Transaction:</strong>
$${spendingAnalysis.average_transaction?.toFixed(2) || '0.00'}</p>
                                        <p><strong>Transaction Count:</strong>
${spendingAnalysis.transaction_count || 0}</p>
                                        <p><strong>Spending Trend:</strong> <span class="badge bg-
${insights.spending_trend === 'increasing' ? 'warning' : insights.spending_trend === 'decreasing' ?
'success' : 'info'}">${insights.spending_trend || 'stable'}</span></p>
                                </div>
                        </div>
                </div>

                <div class="col-md-6">
                        <div class="card mb-3">
                                <div class="card-header bg-info text-white">
                                        <h6><i class="fas fa-tags"></i> Category Breakdown</h6>
                                </div>
                                <div class="card-body">
        `;

        // Add category breakdown
        if (spendingAnalysis.category_breakdown) { for (const
                [category, amount] of
Object.entries(spendingAnalysis.category_breakdown)) { resultsHTML +=
                        `<p><strong>${category}:</strong>
$${amount.toFixed(2)}</p>`;
                }
        } else {
                resultsHTML += `<p>No category data available</p>`;
        }

        resultsHTML += `
                                </div>
                        </div>
                </div>
        </div>
        `;

        // Add charts if available if
        (charts.pie_chart) {
```

```
            resultsHTML += `
                <div class="row">
                    <div class="col-md-6">
                        <div class="chart-container">
                            <h6><i class="fas fa-chart-pie"></i> Spending
Distribution</h6>
                            <img src="${charts.pie_chart}" alt="Spending Pie Chart"
class="img-fluid">
                        </div>
                    </div>
                    <div class="col-md-6">
                        <div class="chart-container">
                            <h6><i class="fas fa-chart-bar"></i> Category
Spending</h6>
                            <img src="${charts.bar_chart}" alt="Category Bar Chart"
class="img-fluid">
                        </div>
                    </div>
                </div>
            `;
        }

        resultsHTML += `
            <div class="card mb-3">
                <div class="card-header bg-warning">
                    <h6><i class="fas fa-lightbulb"></i> Next Steps</h6>
                </div>
                <div class="card-body">
                    <div class="export-buttons">
                        <button class="btn btn-primary" onclick="getBudgetRecommendation()">
                            <i class="fas fa-coins"></i> Get Budget Recommendations
                        </button>
                        <button class="btn btn-success" onclick="generatePDFReport()">
                            <i class="fas fa-file-pdf"></i> Generate PDF Report
                        </button>
                        <button class="btn btn-info" onclick="exportData('excel')">
                            <i class="fas fa-file-excel"></i> Export to Excel
                        </button>
                        <button class="btn btn-secondary" onclick="exportData('csv')">
                            <i class="fas fa-file-csv"></i> Export to CSV
                        </button>
```

```
                                <a href="/chat" class="btn btn-outline-success">
                                    <i class="fas fa-robot"></i> Chat with AI
                                </a>
                            </div>
                        </div>
                    </div>
                `;

                resultsDiv.innerHTML = resultsHTML;
                resultsDiv.style.display = 'block';

                // Store transaction data for later use window.transactionData =
                data.transactions || []; window.analysisData = data;
            }

            function getBudgetRecommendation() {
                const monthlyIncome = prompt('Please enter your monthly income (for personalized budget
            recommendations):');

                if (!monthlyIncome || isNaN(monthlyIncome) || monthlyIncome <= 0) { alert('Please enter a valid
                    monthly income amount.');
                    return;
                }

                // Show loading
                const resultsDiv = document.getElementById('analysisResults'); const
                originalContent = resultsDiv.innerHTML; resultsDiv.innerHTML = originalContent
                + `
                    <div class="text-center mt-3">
                        <div class="spinner-border text-primary"></div>
                        <p>Generating AI budget recommendations...</p>
                    </div>
                `;

                fetch('/api/budget-recommendation', { method: 'POST',
                    headers: {
                        'Content-Type': 'application/json',
                    },
                    body: JSON.stringify({
                        monthly_income: parseFloat(monthlyIncome)
                    })
```

```
        })
        .then(response => response.json())
        .then(data => {
            if (data.status === 'success') { displayBudgetRecommendations(data.recommendations);
            } else {
                alert('Error getting budget recommendations: ' + data.message); resultsDiv.innerHTML =
                originalContent;
            }
        })
        .catch(error => { console.error('Error:', error);
            alert('Error getting budget recommendations');
            resultsDiv.innerHTML = originalContent;
        });
}

function displayBudgetRecommendations(recommendations) {
        const resultsDiv = document.getElementById('analysisResults');

        const budgetHTML = `
            <div class="alert alert-info">
                <h4><i class="fas fa-coins"></i> AI Budget Recommendations</h4>
                <p><strong>Plan:</strong> ${recommendations.category}</p>
            </div>

            <div class="budget-recommendation">
                <div class="budget-breakdown">
                    <div class="budget-item essential">
                        <strong>Essential Spending</strong>
                        <h4>$${recommendations.essential_spending.toFixed(2)}</h4>
                        <small>Housing, Food, Utilities</small>
                    </div>
                    <div class="budget-item discretionary">
                        <strong>Discretionary</strong>

<h4>$${recommendations.discretionary_spending.toFixed(2)}</h4>
                        <small>Entertainment, Dining</small>
                    </div>
                    <div class="budget-item savings">
                        <strong>Savings & Investments</strong>
                        <h4>$${recommendations.savings.toFixed(2)}</h4>
                        <small>Future Financial Goals</small>
```

```
                        </div>
                    </div>
                    <div class="advice-section">
                        <p><strong>Advice:</strong> ${recommendations.advice}</p>
                        <p><strong>Tip:</strong> ${recommendations.tip}</p>
                    </div>
                </div>

                <div class="mt-3 text-center">
                    <button class="btn btn-secondary" onclick="location.reload()">
                        <i class="fas fa-redo"></i> Analyze Another File
                    </button>
                </div>
            `;

            resultsDiv.innerHTML = budgetHTML;
}

function generatePDFReport() { if
        (!window.analysisData) {
            alert('No analysis data available. Please upload and analyze a file first.');
            return;
        }

        // Show loading
        const resultsDiv = document.getElementById('analysisResults'); const
        originalContent = resultsDiv.innerHTML; resultsDiv.innerHTML = originalContent
        + `
            <div class="text-center mt-3">
                <div class="spinner-border text-danger"></div>
                <p>Generating PDF report...</p>
            </div>
        `;

        fetch('/api/generate-pdf-report', { method: 'POST',
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify(window.analysisData)
        })
        .then(response => response.json())
```

```
        .then(data => {
            if (data.status === 'success') {
                // Create download link resultsDiv.innerHTML =
                originalContent + `
                    <div class="alert alert-success mt-3">
                        <h5><i class="fas fa-file-pdf"></i> PDF Report
Generated!</h5>
                        <p>Your financial report has been generated successfully.</p>
                        <a href="${data.download_url}" class="btn btn-success"
download>
                            <i class="fas fa-download"></i> Download PDF Report
                        </a>
                    </div>
                `;
            } else {

                alert('Error generating PDF report: ' + data.message); resultsDiv.innerHTML = originalContent;
            }
        })
        .catch(error => { console.error('Error:', error);
            alert('Error generating PDF report'); resultsDiv.innerHTML =
            originalContent;
        });
}

function exportData(format) {
        if (!window.transactionData || window.transactionData.length === 0) { alert('No transaction data
            available to export.');
            return;
        }

        fetch('/api/export-data', { method:
            'POST', headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify({
                transactions: window.transactionData, format: format
            })
        })
        .then(response => response.json())
        .then(data => {
```

```
            if (data.status === 'success') {
                // Create download link
                const downloadLink = document.createElement('a'); downloadLink.href =
                data.download_url; downloadLink.download = '';
                downloadLink.click();
            } else {
                alert('Error exporting data: ' + data.message);
            }
        })
        .catch(error => { console.error('Error:', error);
            alert('Error exporting data');
        });
}

function downloadSampleData() {
        // Create sample CSV content
        const sampleCSV = `Date,Description,Amount,Category 2024-10-
01,Starbucks Coffee,5.75,Food & Dining
2024-10-02,Supermarket Groceries,85.30,Food & Dining 2024-10-
03,Uber Ride to Work,15.50,Transportation 2024-10-04,Amazon
Shopping,45.99,Shopping
2024-10-05,Electricity Bill,120.00,Bills & Utilities 2024-10-06,Movie
Tickets,25.00,Entertainment
2024-10-07,Gas Station,45.25,Transportation
2024-10-08,Restaurant Dinner,65.80,Food & Dining 2024-10-
09,Netflix Subscription,15.99,Entertainment 2024-10-10,Clothing
Store,89.50,Shopping
2024-10-11,Pharmacy,35.25,Healthcare
2024-10-12,Online Course,199.00,Education 2024-10-
13,Book Store,24.99,Education 2024-10-14,Hotel
Booking,156.00,Travel
2024-10-15,Internet Bill,65.00,Bills & Utilities`;

        // Create and trigger download
        const blob = new Blob([sampleCSV], { type: 'text/csv' }); const url =
        window.URL.createObjectURL(blob);
        const a = document.createElement('a'); a.href = url;
        a.download = 'sample_financial_data.csv';
        document.body.appendChild(a);
        a.click(); document.body.removeChild(a);
```

```
        window.URL.revokeObjectURL(url);
}

// Chat functionality let
chatContext = {
        hasUploadedData: false,
        monthlyIncome: null,
        lastIntent: null
};

function handleKeyPress(event) { if
        (event.key === 'Enter') {
                sendMessage();
        }
}

function sendMessage() {
        const userInput = document.getElementById('userInput'); const message =
        userInput.value.trim();

        if (!message) return;

        // Add user message to chat
        addMessageToChat(message, 'user'); userInput.value = '';

        // Show loading document.getElementById('loadingSpinner').style.display = 'block';

        // Send to backend with NLP fetch('/api/chat', {
                method: 'POST', headers:
                {
                        'Content-Type': 'application/json',
                },
                body: JSON.stringify({ message:
                        message, context: chatContext
                })
        })
        .then(response => response.json())
        .then(data => { document.getElementById('loadingSpinner').style.display = 'none';
```

```
            if (data.status === 'success') {
                // Update chat context if
                (data.intent) {
                    chatContext.lastIntent = data.intent;
                }

                // Add bot response with sentiment styling addMessageToChat(data.response, 'bot',
                data.sentiment);

                // Handle specific intents handleIntentActions(data.intent, message);
            } else {
                addMessageToChat('Sorry, I encountered an error. Please try again.',
'bot');

            }
        })
        .catch(error => { document.getElementById('loadingSpinner').style.display = 'none';
            console.error('Error:', error);
            addMessageToChat('Sorry, I encountered an error. Please try again.',
    'bot');
        });
}

function addMessageToChat(message, sender, sentiment = 'neutral') { const chatContainer =
    document.getElementById('chatContainer'); const messageDiv =
    document.createElement('div'); messageDiv.className = `chat-message ${sender}-
    message`;

    // Add sentiment-based styling
    if (sender === 'bot' && sentiment === 'positive') { messageDiv.style.borderLeft = '4px solid #28a745';
    } else if (sender === 'bot' && sentiment === 'negative') { messageDiv.style.borderLeft = '4px solid #dc3545';
    }

    const senderName = sender === 'user' ? '<i class="fas fa-user"></i> You' : '<i class="fas fa-robot"></i>
Finance AI';
    messageDiv.innerHTML = `<strong>${senderName}:</strong><br>${message}`;

    chatContainer.appendChild(messageDiv); chatContainer.scrollTop =
    chatContainer.scrollHeight;
```

```
}

function handleIntentActions(intent, userMessage) { switch(intent) {
        case 'budget_help':
            // If user mentions income in message, try to extract it const incomeMatch =
            userMessage.match(/\$?(\d+[,.]?\d*)/); if (incomeMatch) {
                const income = parseFloat(incomeMatch[1].replace(',', ''));
                getBudgetRecommendationFromChat(income);
            }
            break;
        case 'spending_analysis':
            if (!chatContext.hasUploadedData) { setTimeout(() => {
                    addMessageToChat("💡 <strong>Pro Tip:</strong> You can upload your
transaction data on the dashboard to get personalized spending analysis!", 'bot', 'positive');
                }, 1000);
            }
            break;
    }
}

function getBudgetRecommendationFromChat(prefilledIncome = null) {
    const monthlyIncome = prefilledIncome || prompt('Please enter your monthly income (for personalized
budget recommendations):');

    if (!monthlyIncome || isNaN(monthlyIncome) || monthlyIncome <= 0) { if (!prefilledIncome)
        {
            alert('Please enter a valid monthly income amount.');
        }
        return;
    }

    // Show loading in chat
    addMessageToChat(`<i class="fas fa-cog fa-spin"></i> Generating AI budget recommendations for
$${monthlyIncome} monthly income...`, 'bot');

    fetch('/api/budget-recommendation', { method: 'POST',
        headers: {
            'Content-Type': 'application/json',
```

```
            },
            body: JSON.stringify({
                monthly_income: parseFloat(monthlyIncome)
            })
        })
        .then(response => response.json())
        .then(data => {
            if (data.status === 'success') { displayBudgetInChat(data.recommendations);
            } else {
                addMessageToChat('Sorry, I encountered an error generating budget recommendations.', 'bot');
            }
        })
        .catch(error => { console.error('Error:', error);
            addMessageToChat('Error getting budget recommendations. Please try again.', 'bot');
        });
}

function displayBudgetInChat(recommendations) { const
        budgetHTML = `
            <div class="budget-recommendation">
                <h6><i class="fas fa-coins"></i> ${recommendations.category}
Plan</h6>
                    <div class="budget-breakdown">
                        <div class="budget-item essential">
                            <strong>Essential:</strong>
$${recommendations.essential_spending.toFixed(2)}
                            <small>Housing, Food, Utilities</small>
                        </div>
                        <div class="budget-item discretionary">
                            <strong>Discretionary:</strong>
$${recommendations.discretionary_spending.toFixed(2)}
                            <small>Entertainment, Dining</small>
                        </div>
                        <div class="budget-item savings">
                            <strong>Savings:</strong>
$${recommendations.savings.toFixed(2)}
                            <small>Future Goals</small>
                        </div>
                    </div>
                    <div class="advice-section">
```

```
                <p><strong>Advice:</strong> ${recommendations.advice}</p>
                <p><strong>Tip:</strong> ${recommendations.tip}</p>
            </div>
        </div>
    `;

    addMessageToChat(budgetHTML, 'bot', 'positive');
}

function quickQuestion(question) {
    document.getElementById('userInput').value = question;
    sendMessage();
}

// Initialize tooltips
document.addEventListener('DOMContentLoaded', function() {
    var tooltipTriggerList = [].slice.call(document.querySelectorAll('[data-bs-
toggle="tooltip"]'));
    var tooltipList = tooltipTriggerList.map(function(tooltipTriggerEl) {
        return new bootstrap.Tooltip(tooltipTriggerEl);
    });
});
```

17. sample_data.csv:

```
Date,Description,Amount,Category
2024-10-01,Starbucks Coffee,5.75,Food & Dining
2024-10-02,Supermarket Groceries,85.30,Food & Dining
2024-10-03,Uber Ride to Work,15.50,Transportation
2024-10-04,Amazon Shopping,45.99,Shopping
2024-10-05,Electricity Bill,120.00,Bills & Utilities
2024-10-06,Movie Tickets,25.00,Entertainment
2024-10-07,Gas Station,45.25,Transportation
2024-10-08,Restaurant Dinner,65.80,Food & Dining
2024-10-09,Netflix Subscription,15.99,Entertainment
2024-10-10,Clothing Store,89.50,Shopping
2024-10-11,Pharmacy,35.25,Healthcare
2024-10-12,Online Course,199.00,Education
```

```
2024-10-13,Book Store,24.99,Education
2024-10-14,Hotel Booking,156.00,Travel
2024-10-15,Internet Bill,65.00,Bills & Utilities
```

18. README.md:

```
# AI-Powered Virtual Personal Finance Assistant

## 🚀 Overview
An intelligent web application that uses machine learning to analyze spending patterns, generate budgets, provide savings recommendations, and answer financial queries through an interactive chatbot.

## ✨ Features
- **AI-Powered Expense Classification** - Automatically categorizes transactions using Random Forest
- **Smart Budget Recommendations** - Personalized budget planning using K-means clustering
- **NLP Chatbot** - Natural language financial advisor with intent recognition
- **Data Visualization** - Interactive charts and graphs (pie charts, bar charts, trends)
- **PDF Report Generation** - Professional financial reports with insights
- **Data Export** - Excel and CSV export capabilities
- **Secure File Processing** - Safe upload and data handling
- **Real-time Analysis** - Instant insights with machine learning models

## 🛠️ Installation

1. **Clone or create project directory**
```bash
mkdir finance_assistant
cd finance_assistant
```

2. **Download or clone this repository**
```bash
git clone https://github.com/yourusername/finance_assistant.git
cd finance_assistant
```

3. **Install Python 3.11+**
```

Download and install Python 3.11 or later from [python.org](https://www.python.org/downloads/). Ensure that the Python installation path is added to your system's PATH variable.

4. **Set up a virtual environment**

It is recommended to use a virtual environment to manage dependencies. From the project root, run:

For Windows:
```bash
python -m venv .venv
.venv\Scripts\activate
```
For macOS/Linux:
```bash
python3 -m venv .venv source
.venv/bin/activate
```

5. **Install dependencies**
```bash
pip install -r requirements.txt
```

6. **Run the application**
```bash python
app.py
```

▶ Run locally (recommended)

Below are two reliable ways to run the project on Windows. The project includes two virtual environments in the repository (`.venv` and `.venv311`). The easiest, lowest-risk option is to use the provided `.venv311` (Python 3.11) which already contains working wheels for binary libraries like scikit-learn.

Option A — Use the existing `.venv311` (quick, recommended)

PowerShell (recommended):

```powershell
# Activate the pre-created Python 3.11 environment
C:/Users/sudar/OneDrive/Desktop/finance_assistant/.venv311/Scripts/Activate.ps1

# From the project root (after activation), run the app python app.py
```

Or run directly without activation:

```powershell C:/Users/sudar/OneDrive/Desktop/finance_assistant/.venv311/Scripts/python.exe app.py
```

Option B — Recreate a fresh `.venv` using Python 3.11 (recommended for a single environment)

1. Install Python 3.11 on your system (if not installed).
2. From the project root run (PowerShell):

```powershell
# create a new venv (this will overwrite or replace an existing .venv if you choose)
python3.11 -m venv .venv
.\.venv\Scripts\Activate.ps1

# upgrade pip and install dependencies
python -m pip install -U pip setuptools wheel python -m pip
install -r requirements.txt

# run the app
python app.py
```

Troubleshooting & notes
- If you get "No module named 'sklearn'" when using `.venv`, it's likely because `.venv` uses Python 3.13 and scikit-learn wheels for Windows/Python 3.13 may not be available. Use `.venv311` or recreate `.venv` with Python 3.11.
- If pip tries to build scikit-learn from source you will need a C build toolchain (MSVC) and Cython — this is slow and error-prone on Windows; prefer Python 3.11 wheels instead.

```
- If you add new packages, pin them in `requirements.txt` or `requirements-
pinned-for-install.txt` for reproducible installs.
```

13.create virtual environment:

python -m venv finance_env #

On Windows:

finance_env\Scripts\activate #

On Mac/Linux:

source finance_env/bin/activate

***20*. install dependencies:**

pip install -r requirements.txt

***21*. run the application:**

python app.py

***22*. open the browser:**

http://192.168.0.104:5000

OUTPUT:

PS C:\Users\sudar\OneDrive\Desktop\finance_assistant> C:/Users/sudar/OneDrive/Desktop/finance_assistant/.venv311/Scripts/python.exe app.py [INFO] Expense classification model trained and saved successfully [INFO] Budget recommendation model

trained and saved successfully [INFO] Initializing AI Finance Assistant... [INFO] Pre-trained expense classifier loaded successfully [INFO] Pre-trained budget recommender loaded successfully [INFO] Pre-trained NLP chatbot loaded successfully [INFO] ML models initialized successfully [INFO] Starting Finance Assistant Server...

- Serving Flask app 'app'
- Debug mode: on 2025-10-07 15:34:08,525 INFO werkzeug MainThread : WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
- Running on all addresses (0.0.0.0)
- * Running on http://127.0.0.1:5000
- * Running on http://192.168.0.104:5000 2025-10-08 21:32:56,902 INFO werkzeug MainThread : Press CTRL+C to quit

  [INFO] Starting Finance Assistant Server...

  2025-10-08 21:32:58,849 WARNING werkzeug MainThread : * Debugger is active!

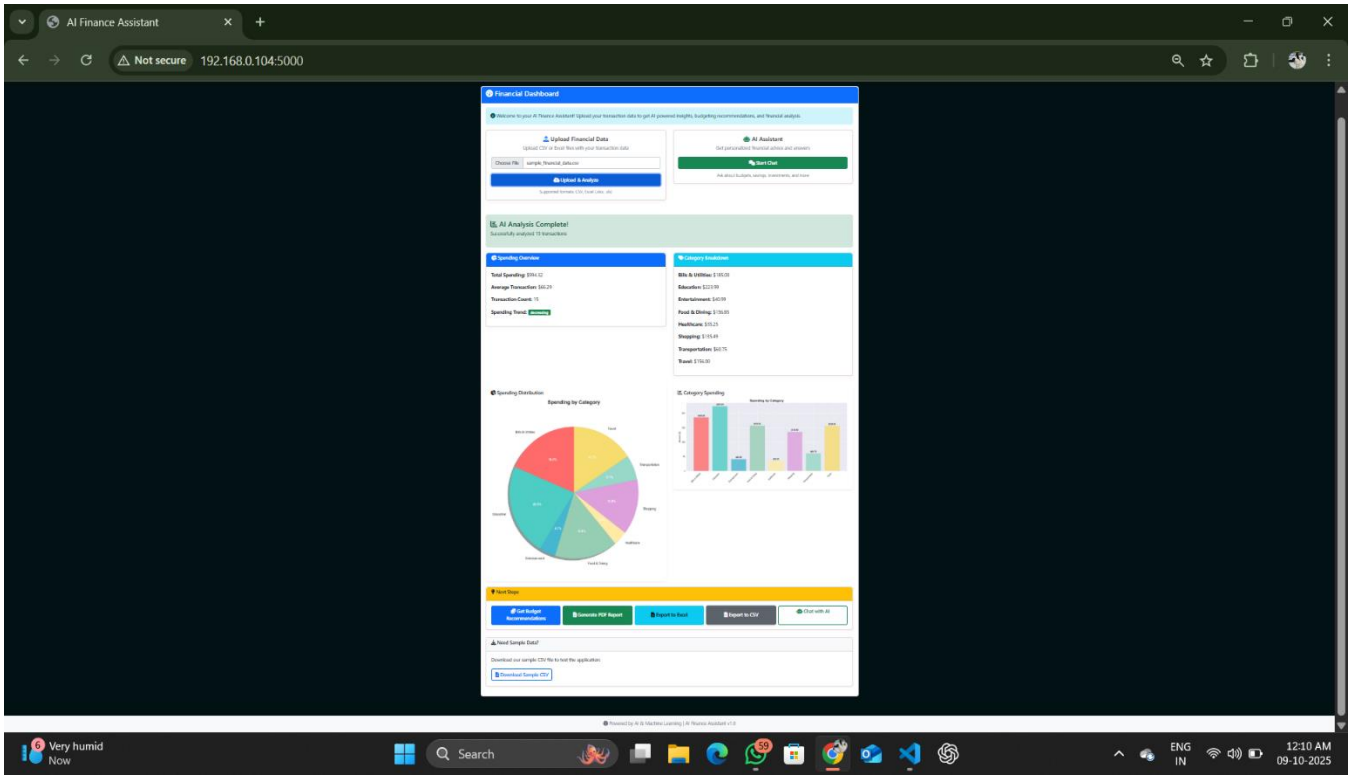
  2025-10-08 21:32:58,850 INFO werkzeug MainThread : * Debugger PIN: 253-650-417

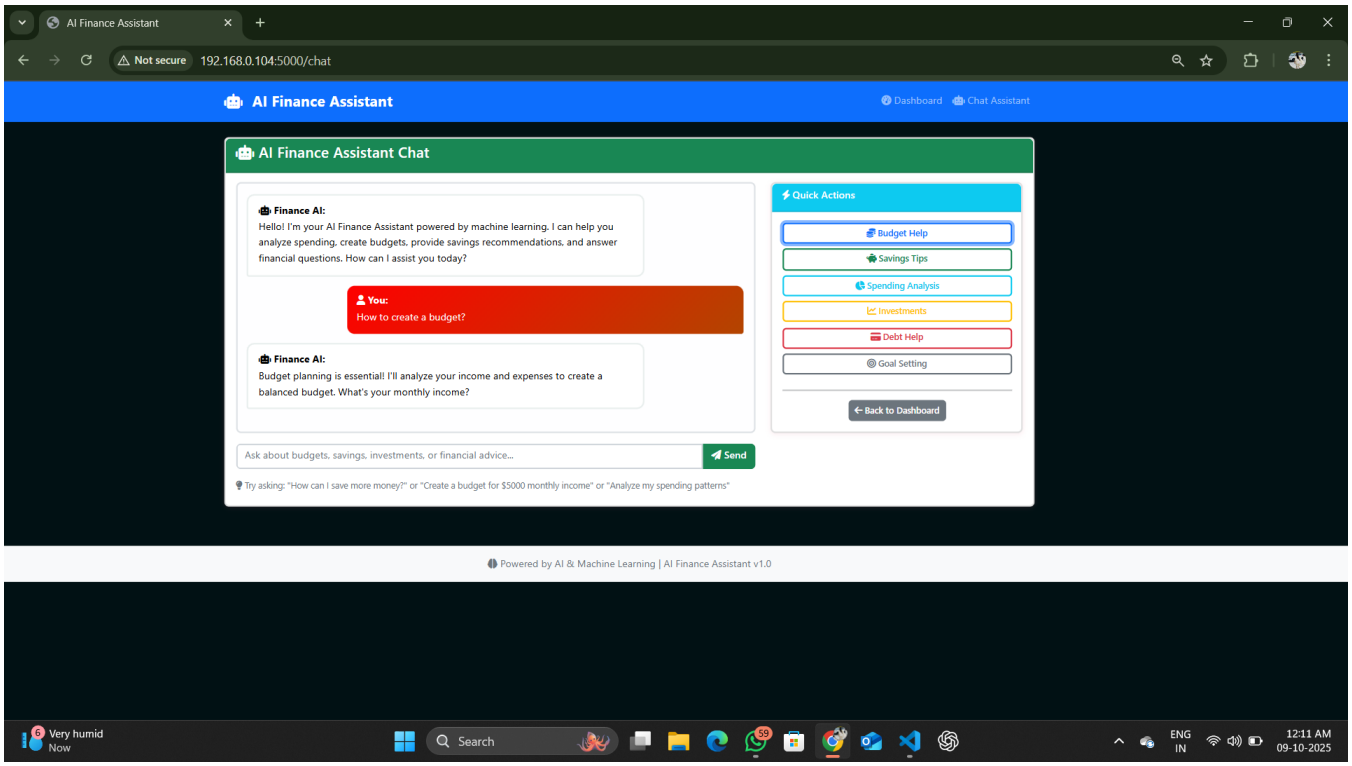
  [INFO] Starting Finance Assistant Server...

  2025-10-08 21:32:58,849 WARNING werkzeug MainThread : * Debugger is active!

  2025-10-08 21:32:58,850 INFO werkzeug MainThread : * Debugger PIN: 253-650-417

OUTPUT:


AI Finance Assistant
Not secure
192.168.0.104:5000
Financial Dashboard
Upload Financial Data
AI Assistant
AI Analysis Complete!
Spending Overview
Category Breakdown
Spending Distribution
Spending by Category
Category Spending
Next Steps
Need Sample Data?
Very humid
Now
Search
ENG
IN
12:10 AM
09-10-2025


AI Finance Assistant
Not secure
192.168.0.104:5000/chat
AI Finance Assistant
Dashboard
Chat Assistant
AI Finance Assistant Chat
Finance AI:
Hello! I'm your AI Finance Assistant powered by machine learning. I can help you analyze spending, create budgets, provide savings recommendations, and answer financial questions. How can I assist you today?
You:
How to create a budget?
Finance AI:
Budget planning is essential! I'll analyze your income and expenses to create a balanced budget. What's your monthly income?
Quick Actions
Budget Help
Savings Tips
Spending Analysis
Investments
Debt Help
Goal Setting
Back to Dashboard
Ask about budgets, savings, investments, or financial advice...
Send
Try asking: "How can I save more money?" or "Create a budget for $5000 monthly income" or "Analyze my spending patterns"
Powered by AI & Machine Learning | AI Finance Assistant v1.0

Very humid
Now
Search
ENG
IN
12:11 AM
09-10-2025