Early Risk Signals - Credit Card Delinquency Watch

Complete System Documentation

Submitted By: Biswanath Tudu

Roll. No: AEC.24

Documentation Contents:

1. Executive Summary

The Credit Card Delinquency Watch system is a comprehensive machine learning solution designed to predict and monitor credit card delinquency risk. The system has been restructured from a monolithic architecture to a scalable, modular 3-layer design with clear separation of concerns.

Key Features:

- Gradient Boosting ML model with 5 engineered behavioral signals
- Real-time risk scoring and portfolio analysis
- RESTful API endpoints for flexible integration
- Interactive web dashboard with 3 analytical tabs
- Centralized configuration for easy parameter management
- Comprehensive documentation for team collaboration

Technology Stack:

- Backend: FastAPI, scikit-learn, pandas, numpy
- Frontend: HTML5, CSS3, JavaScript, Chart.js
- Configuration: Pydantic for data validation

2. Project Structure & Architecture

Folder Organization:
The system is organized into three main layers following the separation of concerns principle:

Backend Layer (backend/app/)
• **core/** - Core ML and data processing
  - config.py: Centralized configuration (20+ parameters)
  - data_loader.py: Data loading and feature engineering
  - model_trainer.py: ML model training and predictions

• **api/** - REST API endpoints
  - routes.py: 8 endpoints with dependency injection

• **services.py** - Business logic layer
  - RiskScoringService
  - InterventionService
  - CustomerService

• **models/** - Data validation schemas
  - Pydantic models for request/response validation

• **main.py** - Application factory and initialization

Frontend Layer (frontend/public/)
• index.html: Interactive dashboard with 3 tabs (600 lines)
  - Dashboard tab: Portfolio statistics and visualizations
  - Customers tab: Customer search and filtering
  - Scoring Tool: Individual customer risk assessment

Configuration & Entry Points
• run.py: Main entry point for starting the application
• requirements.txt: Python package dependencies

Documentation (docs/)
• 8 comprehensive markdown guides covering all aspects
• Architecture diagrams and design patterns
• Setup, deployment, and developer guides

3. API Documentation

The system provides 8 RESTful endpoints for portfolio analysis, customer scoring, and intervention recommendations. All endpoints are accessible at **http://localhost:8000/api/v1/**

Endpoint	Method	Purpose
/portfolio-summary	GET	Overall portfolio statistics
/signals	GET	Risk signal effectiveness analysis
/feature-importance	GET	ML model feature importance
/risk-distribution	GET	Risk tier distribution
/score-customer	POST	Score individual customer
/customers	GET	List all customers with scores
/dashboard-stats	GET	Complete dashboard statistics

API Features:

- Automatic Swagger/OpenAPI documentation at /docs
- CORS enabled for cross-origin requests
- Dependency injection for service abstraction
- Pydantic validation for request/response data
- Comprehensive error handling
- Health check endpoint at /health

4. Risk Signals & ML Model

The system uses 5 engineered behavioral signals to predict delinquency risk. These signals are derived from customer spending patterns and are normalized before being fed into the ML model.

ML Model Details:

- Algorithm: Gradient Boosting Classifier
- Parameters: n_estimators=100, learning rate=0.1
- Features: 5 engineered behavioral signals
- Output: Delinquency probability (0-1 range)
- Risk Classification: HIGH (≥ 3), MEDIUM (2–2.99), LOW (< 2)

5. Quick Start and Setup Guide

Local Development Setup (5 minutes):

 # Step 1: Create Virtual Environment
Python -m venv .venv

#Step 2: Activate Virtual Environment
Windows: .venv\Scripts\activate
Linux/Mac: source .venv/bin/activate

 # Step 3: Install Dependencies
pip install -r requirements.txt

 # Step 4: Run Application
python run.py

 # Step 5: Access Dashboard
Open browser: http://localhost:8000

 # Step 6: View API Documentation
Visit: http://localhost:8000/docs

Common Tasks:

Change Risk Threshold:
Edit backend/app/core/config.py RISK_HIGH_THRESHOLD

Add New Signal:
1. Add to data_loader.py engineer_signals() method
2. Define threshold in config.py
3. Retrain model

View API Docs:
Navigate to http://localhost:8000/docs in browser

6. Dashboard Features

Tab 1: Portfolio Dashboard
- Total customers and delinquency rate
- Risk tier distribution (pie chart)
- Score distribution (bar chart)
- Key performance indicators

Tab 2: Customer Management
- Search customers by ID or name
- Filter by risk tier (HIGH, MEDIUM, LOW)
- View detailed customer information
- Sort by risk score

Tab 3: Risk Scoring Tool
- Input customer metrics (spending, utilization, etc.)
- Real-time risk calculation
- Detailed risk assessment report
- Signal contribution analysis

7. Configuration Management

All system parameters are centralized in **backend/app/core/config.py** for easy management without modifying code. Key configuration areas:

Risk Thresholds:
• RISK_HIGH_THRESHOLD = 3.0
• RISK_MEDIUM_THRESHOLD = 2.0

Signal Thresholds:
• SPEND_DECLINE_THRESHOLD = -10%
• UTILIZATION_THRESHOLD = 80%
• PAYMENT_DECLINE_THRESHOLD = -50%
• CASH_SURGE_THRESHOLD = 30%

Intervention Costs:
• High-risk intervention: $20
• Medium-risk intervention: $7.50
• Low-risk intervention: $0.50

Model Parameters:
• n_estimators = 100
• learning_rate = 0.1

8. Developer Guide

Adding a New Signal:

1. Define signal calculation in data_loader.py
2. Add threshold to config.py
3. Include in engineer_signals() method
4. Update feature list in model_trainer.py
5. Retrain model

Adding a New API Endpoint:

1. Create service method in services.py
2. Add route in api/routes.py
3. Define Pydantic models if needed
4. Update API documentation
5. Test with /docs interface

Updating the Dashboard:

1. Modify frontend/public/index.html
2. Update JavaScript functions for API calls
3. Add Chart.js visualization if needed
4. Test in browser at http://localhost:8000

Code Style Guidelines:

- Follow PEP 8 conventions
- Use type hints for functions
- Write docstrings for modules and classes
- Keep functions focused (single responsibility)
- Use configuration for magic numbers

9. Architecture & Design Patterns

Modular 3-Layer Architecture:

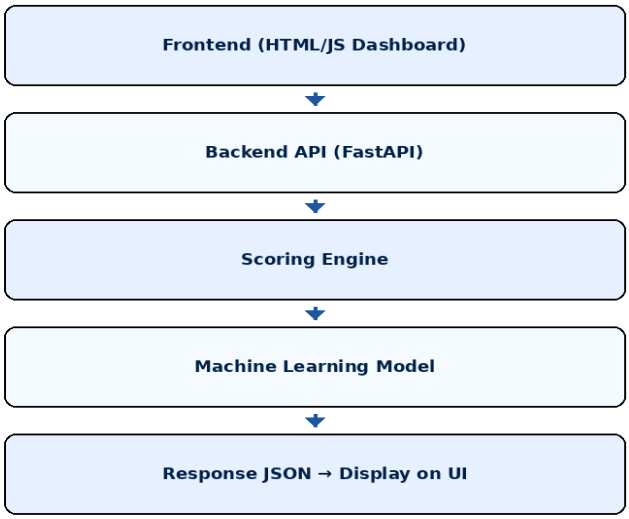
Layer 1 - API Routes (api/routes.py)
Handles HTTP requests and responses. Uses FastAPI for automatic documentation and dependency injection.

Layer 2 - Services (services.py)
Implements business logic. Three service classes isolate different concerns:
• RiskScoringService: Portfolio and individual scoring
• CustomerService: Customer data management

Layer 3 - Core (core/)
Manages data processing and ML operations:
• config.py: Configuration management
• data_loader.py: Feature engineering
• model_trainer.py: Model training and predictions

Design Patterns Used:
• **Dependency Injection:** Services receive dependencies in __init___
• **Factory Pattern:** create_app() and create_routes() functions
• **Separation of Concerns:** Each module has single responsibility
• **Configuration Management:** Centralized config.py
• **Data Validation:** Pydantic models for request/response

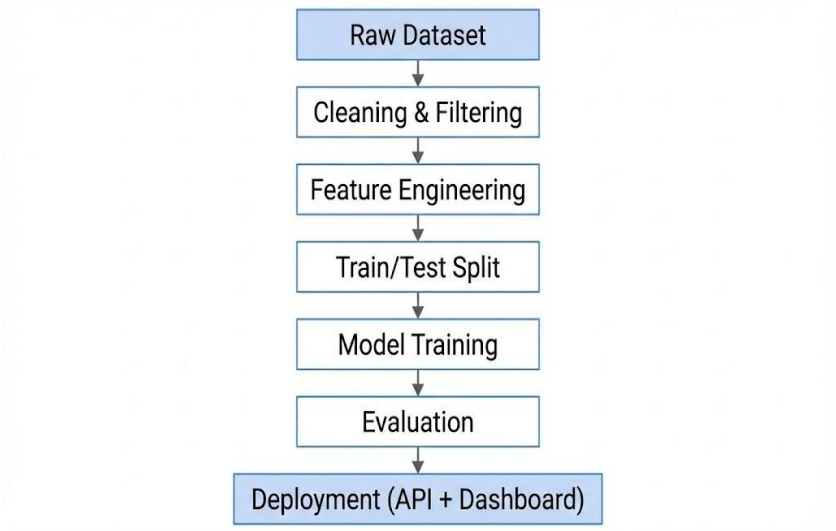
10. System Architecture



Components:

- **Frontend Dashboard** – User inputs data & views results
- **FastAPI Backend** – Hosts model and scoring logic
- **Model Loader** – Loads trained ML model
- **Scoring Service** – Computes signals, risk score
- **API Router** – Defines endpoints

11. Data Pipeline



12. Support & Additional Resources

Documentation Files (in docs/ folder):

- **QUICK_REFERENCE.md**
  Quick lookup for commands and common tasks

- **PROJECT_STRUCTURE.md**
  Detailed architecture and component descriptions

- **API_DOCUMENTATION.md**
  Complete API reference with request/response examples

- **SETUP_AND_DEPLOYMENT.md**
  Local development and production deployment guides

- **DEVELOPER_GUIDE.md**
  How to add features, test, and debug

- **ARCHITECTURE_DIAGRAM.md**
  Visual representations of system flow and components

- **DOCUMENTATION_INDEX.md**
  Navigation guide to all documentation

Technology References:
- FastAPI: https://fastapi.tiangolo.com
- scikit-learn: https://scikit-learn.org
- pandas: https://pandas.pydata.org
- Chart.js: https://www.chartjs.org

Troubleshooting:
- Port 8000 in use? Change in run.py: uvicorn.run(..., port=8001)
- Import errors? Reinstall: pip install -r requirements.txt
- Data issues? Verify csv_path in config.py

Conclusion

The Credit Card Delinquency Watch system is now fully restructured into a production-ready, scalable architecture. The modular design enables easy maintenance, testing, and extension. With comprehensive documentation and clear design patterns, the system is ready for team collaboration and future development.

Next Steps:

1. Review the project structure in the backend/ folder
2. Explore the API at http://localhost:8000/docs
3. Interact with the dashboard at http://localhost:8000
4. Read DEVELOPER_GUIDE.md to understand how to extend the system

For Questions:

Refer to the comprehensive documentation in the docs/ folder. Each guide covers specific aspects of the system with examples and best practices.