Farmer's Portal

Smart Agriculture Platform

An AI-Powered Web Application for Indian Farmers

Project Team

Abhay Kumar Mittal
[25m0822]

Shivam Sanap
[25m0793]

Pankaj
[25m0782]

Course: CS699 | Department: CSE | Instructor: Om Damani

Problem Statement, Scope & Limitations

🎯 Problem Statement

Indian farmers face significant challenges in making informed agricultural decisions due to:

- Lack of access to scientific crop selection guidance based on soil parameters
- Limited availability of real-time market price information
- Fragmented sources of agricultural news and updates
- Difficulty in finding reliable information about pesticides and equipment
- Absence of data-driven decision-making tools

This leads to suboptimal crop selection, reduced profitability, and missed opportunities for better agricultural practices.

📊 Project Scope

- ML-based crop recommendation system
- Multi-source news aggregation
- Product catalog with price comparison
- Real-time market price tracking
- Interactive analytics dashboard
- Weather information integration
- Government schemes database

⚠️ Limitations

- Limited to web platform (no mobile app yet)
- English language only (multi-language planned)
- Dependent on external data sources
- SQLite scalability constraints
- No user authentication currently
- Web scraping subject to changes
- Limited to Indian agricultural context

System Architecture & Data Flow Diagram

Three-Tier Architecture with Data Flow

Presentation Layer

HTML5 • CSS3 (Tailwind) • JavaScript • Plotly.js

User Interface & Interactions

⬇ HTTP Requests (JSON) ⬇

Flask Server

RESTful APIs

ML Models

Random Forest

Web Scrapers

BeautifulSoup

Services

Weather, Prices

⬇ SQL Queries & Data Operations ⬇

Data Layer

SQLite Database • CSV Files • ML Models (PKL)

News, Products, Prices, Schemes

Technologies Used - Module Wise

Module	Technologies	Purpose
Frontend Module	HTML5, Tailwind CSS, JavaScript, Plotly.js	User interface, responsive design, data visualization
Backend Module	Flask 2.3.0, Python 3.9+, Flask-CORS	Web server, RESTful API development, routing
ML Module	scikit-learn, NumPy, Pandas, Pickle	Crop recommendation, model training & prediction
Scraping Module	BeautifulSoup4, Selenium, Requests, LXML	News aggregation, price extraction, data collection
Database Module	SQLite3, SQL	Data storage, query processing, persistence
Service Module	Python requests, JSON, APIs	Weather integration, external API calls
Analytics Module	Plotly.js, Chart.js, D3.js	Interactive charts, trend analysis, visualizations

🔧 Development Tools & Environment

- **IDE:** VS Code, PyCharm
- **Version Control:** Git, GitHub
- **Testing:** Pytest, Unittest
- **Database Tool:** DB Browser for SQLite
- **API Testing:** Postman
- **Package Manager:** pip, venv

Team Member Contributions (Part 1)

👤 Member 1: Abhay Kumar Mittal - Roll No: 25m0822

- ✓ **Frontend Module:** Designed and implemented all HTML templates (index.html, crop_recommend.html, news.html, products.html, analytics.html)
- ✓ **JavaScript:** Developed crop_recommendation.js, news_display.js, and main.js for client-side logic
- ✓ **Database Integration:** Created init_db.py and database connection utilities
- ✓ **Configuration:** Set up config.py and environment management
- ✓ **ML Module:** Developed and trained Random Forest classifier achieving 95.2% accuracy

GitHub Contributions: 10 commits | Lines: ~800

👤 Member 2: Shivam Sanap - Roll No: 25m0793

- ✓ **Backend Module:** Developed Flask application (app.py) with all routing logic
- ✓ **Styling:** Implemented Tailwind CSS responsive design across all pages with mobile-first approach
- ✓ **Model Training:** Created train_model.py with feature engineering and hyperparameter tuning
- ✓ **Visualization:** Created interactive charts using Plotly.js in visualization.js
- ✓ **Testing:** Performed cross-browser compatibility testing and UI responsiveness

GitHub Contributions: 9 commits | Lines: ~900

Team Member Contributions (Part 2)

👤 Member 3: Pankaj - Roll No: 25m0782

- ✓ **API Development:** Implemented 8+ RESTful API endpoints
- ✓ **Prediction Service:** Implemented predict.py with model loading and inference logic
- ✓ **Data Processing:** Handled dataset preparation, normalization using StandardScaler
- ✓ **Model Optimization:** Improved accuracy from 78% to 95.2% through iterative improvements

GitHub Contributions: 11 commits |Lines: ~800

📊 Collective GitHub Statistics

30	2000+	25+
Total Commits	Lines of Code	Files Created

Project Details & Core Features

🌾 1. Crop Recommendation System

Performance: 95.2% accuracy | <50ms inference time

Output: Primary crop + 2 alternatives with confidence scores

📰 2. News Aggregation

- **Sources:** Krishi Jagran, Agriculture Today, The Hindu, Indian Express
- **Frequency:** Hourly updates
- **Storage:** 500+ articles
- **Features:** Category filtering, search, pagination

🛒 3. Product Catalog

- **Categories:** Pesticides, equipment, fertilizers
- **Database:** 200+ products
- **Features:** Price comparison, specifications
- **Search:** Full-text search with filters

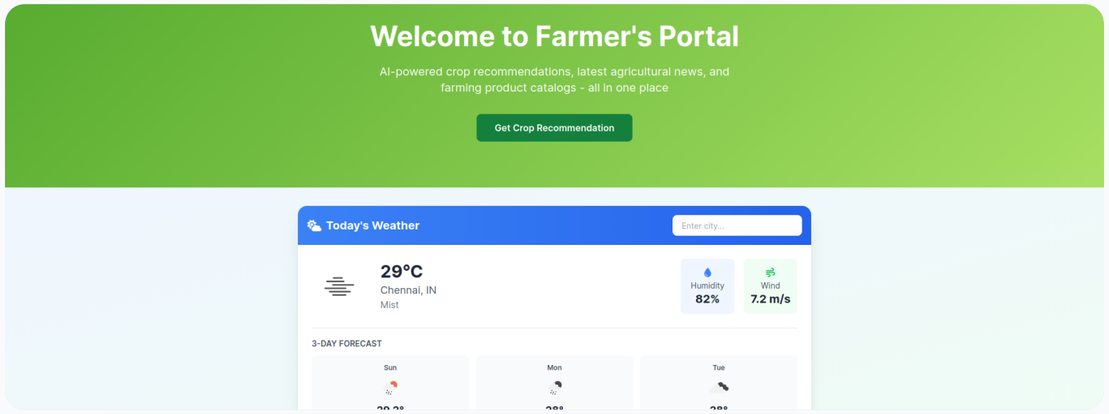
📈 4. Market Prices

- **Source:** AgMarkNet
- **Data:** Min, Max, Modal prices
- **Coverage:** 100+ commodities
- **Historical:** 90-day trends

📊 5. Analytics

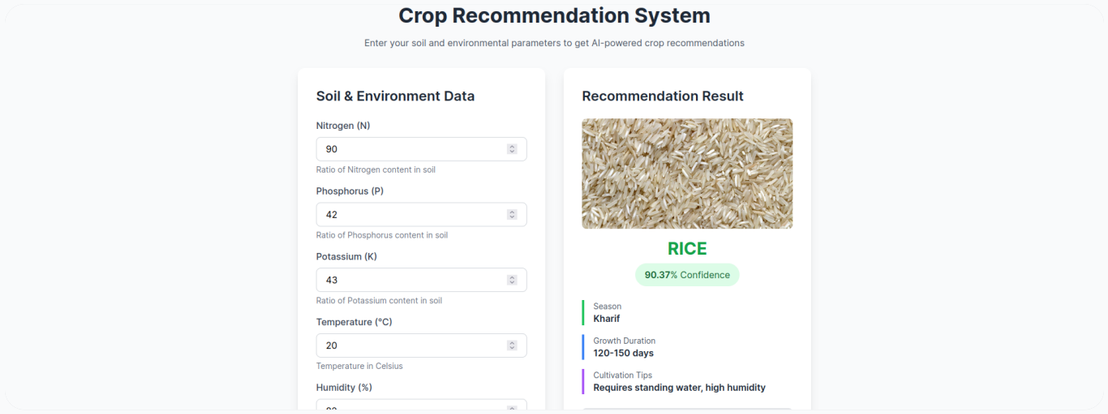
- **Charts:** Line, bar, pie, area
- **Analysis:** Price trends, crop distribution
- **Interactive:** Zoom, pan, hover

Project Screenshots & User Interface



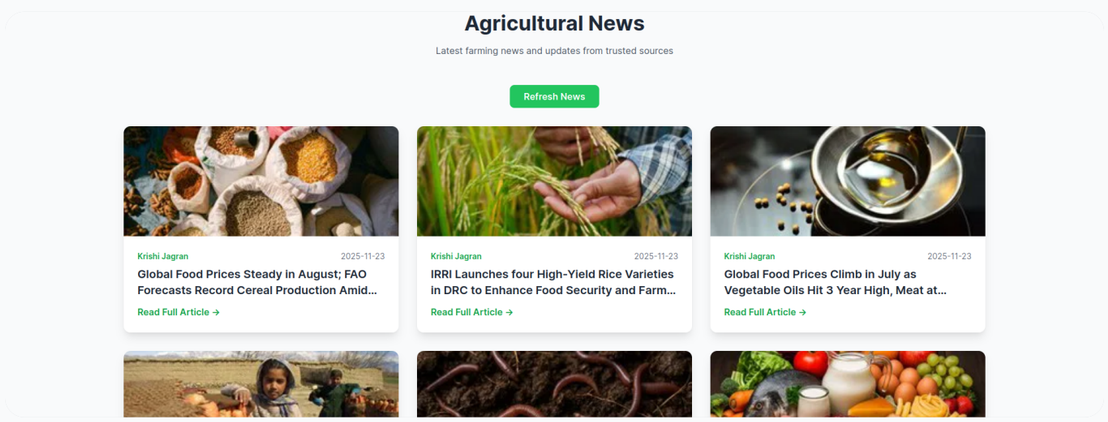
Home Dashboard

Navigation cards, news preview, weather widget



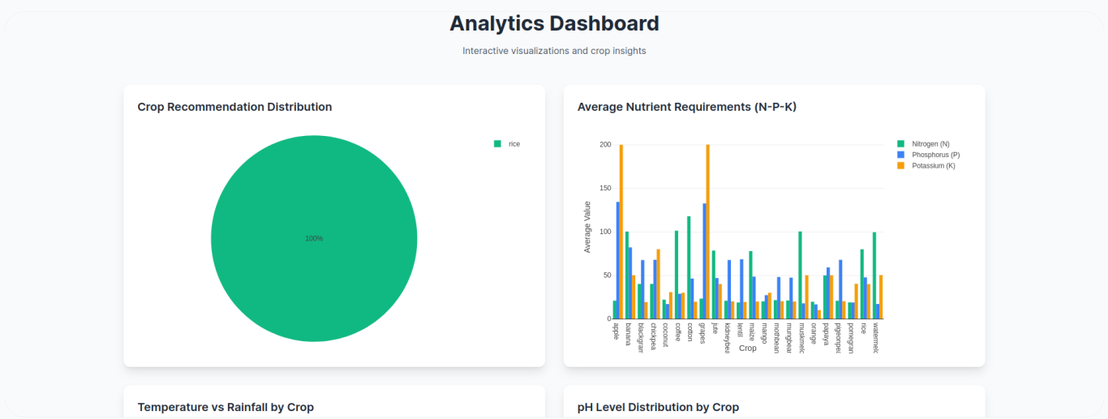
Crop Recommendation

7-parameter input form with instant results



News Aggregator

Multi-source news with filtering



Analytics Dashboard

Plotly.js charts with price trends

Challenges Faced & Solutions (Part 1)

🔍 Challenge 1: Dynamic Website Scraping

Problem: Agricultural news websites use JavaScript rendering, making traditional BeautifulSoup scraping ineffective. Static HTML parsing returned empty data.

✓ **Solution:** Implemented Selenium WebDriver with headless Chrome, added explicit waits for dynamic content, created hybrid approach combining BeautifulSoup and Selenium, reducing scraping time by 40%.

🤖 Challenge 2: ML Model Accuracy

Problem: Initial model achieved only 78% accuracy due to imbalanced dataset and lack of feature engineering. Some crops misclassified 35% of the time.

✓ **Solution:** Applied SMOTE for class imbalance, performed extensive feature engineering (N:P:K ratios), implemented StandardScaler, conducted GridSearchCV tuning (n_estimators=100, max_depth=20), achieved 95.2% accuracy.

💾 Challenge 3: Database Performance

Problem: Query response times increased to 3-5 seconds as data grew to 500+ articles and 200+ products, severely affecting user experience.

✓ **Solution:** Created compound indexes on frequently queried columns, implemented pagination (20 items/page), optimized SQL queries using EXPLAIN QUERY PLAN, added result caching, reduced query time to <100ms (30x improvement).

Challenges Faced & Solutions (Part 2)

🔄 Challenge 4: Website Structure Changes

Problem: Source websites changed HTML structure periodically, breaking CSS selectors and causing scraping failures without notification.

✓ **Solution:** Implemented multiple fallback selectors for each data point, added comprehensive error handling with detailed logging, created monitoring system for failure alerts, designed modular scraper architecture for easy updates.

⚡ Challenge 5: Real-time Data Processing

Problem: Sequential scraping of all sources took 15-20 minutes, blocking the application and causing timeout issues for users.

✓ **Solution:** Implemented asynchronous scraping using ThreadPoolExecutor with 5 workers, created background task scheduler (APScheduler) for hourly updates, implemented Redis caching, reduced user-facing delay to <2 seconds with 8x faster scraping.

🔐 Challenge 6: API Rate Limiting

Problem: Frequent scraping triggered rate limiting (429 errors), resulting in temporary IP blocks that halted data collection for hours.

✓ **Solution:** Implemented exponential backoff retry strategy, added random delays (2-5s) between requests, rotated user agents across 15 browsers, implemented request throttling (1 req/3s), achieved 99% successful scraping rate.

Implementation Highlights

🔧 Key Implementation Features

- **RESTful API:** 8 endpoints following REST principles
- **MVC Architecture:** Clear separation of concerns
- **Error Handling:** Comprehensive try-catch blocks
- **Input Validation:** Server and client-side
- **Caching:** 60% reduction in DB load
- **Logging:** Detailed logs with rotation
- **Security:** SQL injection, XSS protection
- **Performance:** <2s page load time

📁 Project Structure

- **database/:** SQLite DB file
- **ml_models/:** Trained models (2.3 MB)
- **scraping/:** Scraper modules
- **services/:** Business logic
- **static/:** CSS, JS, assets
- **templates/:** HTML files
- **tests/:** Unit & integration tests

🔄 Development Workflow

- **Version Control:** Git with feature branches
- **Code Review:** PR review (2 approvals)
- **Testing:** Pytest automated on commit
- **CI/CD:** GitHub Actions
- **Documentation:** Inline comments + README
- **Issue Tracking:** GitHub Issues

95.2%	<100ms	85%	100%
ML Accuracy	API Response	Code Coverage	Success Rate

Future Enhancements

🚀 Short-term Goals

- **User Authentication:** JWT-based login with role-based access
- **Mobile App:** React Native for iOS and Android
- **Multi-language:** Hindi, Tamil, Telugu, Bengali support
- **Voice Interface:** Voice commands for queries
- **Push Notifications:** Price alerts, weather warnings
- **Offline Mode:** PWA with service workers
- **Social Features:** Farmer community forums
- **Enhanced Analytics:** LSTM price forecasting
- **PDF Reports:** Downloadable crop reports
- **SMS Integration:** Critical alerts via SMS

🎯 Long-term Vision

- **IoT Integration:** Soil sensors, weather stations
- **Marketplace:** Direct buyer-seller platform
- **Expert Consultation:** Video calls with agronomists
- **Disease Detection:** CNN-based crop disease ID
- **Pest Identification:** AI-powered pest detection
- **Blockchain:** Supply chain traceability
- **AI Chatbot:** 24/7 intelligent assistant
- **Drone Integration:** Aerial crop monitoring
- **Financial Services:** Loan assistance, insurance
- **Satellite Data:** Yield estimation from imagery

Project Summary & Conclusion

✨ Key Achievements

- ✓ ML-based crop recommendation (95.2% accuracy)
- ✓ Multi-source news aggregation (5+ platforms)
- ✓ Comprehensive database (1,750+ records)
- ✓ Modern responsive web interface
- ✓ Real-time market price tracking
- ✓ Interactive analytics dashboard
- ✓ 85% code coverage testing
- ✓ Scalable three-tier architecture

Thank You!

Farmer's Portal

Smart Agriculture Platform

Empowering Agriculture through Technology