Product Requirements Document (PRD) for FIRE\_Version\_1

1. Executive Summary

FIRE\_Version\_1 (Field Insight & Reporting Engine) is a web-based data analysis and reporting application targeted at non-technical field staff and program managers. The application allows users to upload field data (CSV/Excel files), ask natural language questions, and receive both textual insights and interactive visualizations. Powered by AI (via an open router API for Deepseek) and supported by a robust backend (FastAPI) and an intuitive frontend (Streamlit), FIRE\_Version\_1 democratizes data analysis and reporting for users who may not have deep technical expertise.

2. Objectives

• Ease of Use:

Allow non-technical users to upload data and get insights using natural language queries without needing to understand underlying data processes.

• Rapid Insight Generation:

Enable quick responses to data queries by integrating AI capabilities that transform natural language requests into data analysis operations.

• Interactive Reporting:

Provide clear visualizations (using Plotly) that help users quickly understand trends and insights in their data.

• Modular & Scalable Architecture:

Build the system using modular components so that it can be easily maintained, scaled, and extended with new features.

• Enhanced Context through RAG:

Integrate supporting documents (policies, reports) to augment the analysis results with additional context.

⸻

3. Target Users

• Field Program Staff:

Individuals working in the field who collect data on beneficiaries, services, and outcomes. They need a tool to quickly process and understand the data without extensive technical training.

• Program Managers:

Managers who require consolidated reports and visualizations to monitor performance across regions or programs.

• Data Analysts (Non-Technical):

Users who can leverage natural language to run data queries and gain insights without writing code or complex queries.

4. User Personas

• Alice, the Field Officer:

Profile: Works on the ground collecting data.

Needs: Quickly upload field data, run simple queries (e.g., “How many beneficiaries in region X?”), and share visual summaries with her supervisor.

Challenges: Limited technical skills, time constrained.

• Brian, the Program Manager:

Profile: Oversees multiple field operations and needs high-level summaries to make decisions.

Needs: Access historical data, compare program performance across regions, and export detailed reports.

Challenges: Balancing multiple data sources, requires clear and actionable insights.

• Catherine, the Data Champion:

Profile: A non-technical user responsible for reporting and data cleaning.

Needs: Tools for basic data cleaning and standardized analysis templates (e.g., demographic breakdowns).

Challenges: Ensuring data integrity and consistency before analysis.

5. User Journey

5.1 Onboarding and Login

1. User Registration/Authentication:

• Users log in through a simple authentication system (initially simulated, later integrated with Supabase).

• Upon successful login, users access their personalized dashboard where they can view past uploads and reports.

2. Dashboard Overview:

• The dashboard displays a welcome message, recent uploads, and quick actions (e.g., “Upload New File”, “Run Query”).

5.2 Data Upload Flow

1. File Upload:

• The user selects one or more CSV/Excel files using an intuitive file uploader.

• Uploaded files are previewed (first few rows) so that users can confirm the data format.

2. Data Validation & Cleaning:

• The system automatically checks for missing values or outliers.

• Suggestions for data cleaning (e.g., imputing missing values) are provided with an option to apply them.

5.3 Data Analysis via Natural Language

1. Query Input:

• A chat box allows the user to type natural language questions (e.g., “What is the total number of beneficiaries in region A?”).

• Users can toggle between “Quick Mode” for simple queries and “Advanced Mode” for complex requests.

2. AI Processing:

• The user’s query is sent to the backend where the analysis engine (using the Llama 4 API) interprets the question.

• The query is translated into data operations using Pandas and then processed.

3. Results Display:

• The result is returned as text and, if applicable, accompanied by interactive visualizations (e.g., bar charts, line graphs).

• Users can click on charts for further drill-down or explore additional data insights.

5.4 Reporting and Export

1. Template-based Analysis:

• Users can select from a dropdown of predefined templates (e.g., demographic summary, regional service report).

• The system processes the selected template and generates a detailed report.

2. Export Options:

• Users have the option to export analysis results in multiple formats (PDF, DOCX, CSV).

• Exported reports include both textual summaries and embedded charts.

5.5 Augmented Analysis (RAG Integration)

1. Supporting Documents Upload:

• Users can upload additional documents (e.g., policies, M&E reports) for contextual analysis.

• The system uses LangChain to integrate relevant excerpts from these documents into the analysis results.

2. Enhanced Query Response:

• When a user asks a question, the backend combines insights from the data with contextual information extracted from supporting documents.

6. Detailed Feature Requirements

6.1 Frontend Features

• User Login and Dashboard:

• Simple login interface.

• Personalized dashboard showing upload history and previous analyses.

• File Uploader:

• Supports CSV and Excel files.

• Previews file content immediately after upload.

• Chat Interface:

• Text input for natural language queries.

• Toggle between “Quick” and “Advanced” query modes.

• Visualization Display:

• Interactive charts using Plotly.

• Ability to drill down into data visualizations on click.

• Export Functionality:

• Options to export reports in PDF, DOCX, and CSV formats.

6.2 Backend Features

• FastAPI Endpoints:

• /analyze: Accepts natural language queries, returns analysis results and chart data.

• /uploadfile: Handles file uploads and processes them for analysis.

• AI-Driven Analysis Engine:

• Integrates with Deepseek (via open router API) for interpreting natural language.

• Uses Pandas for data manipulation and analysis.

• Data Cleaning Module:

• Automatically detects and fills missing values.

• Provides options for outlier detection and handling duplicates.

• Template Engine:

• Runs pre-defined analysis templates (demographic, regional service, gender/age analysis).

• RAG Module:

• Integrates external document context using LangChain.

• Enhances responses by combining data insights with contextual document excerpts.

6.3 Utility and Configuration

• File Management:

• Handles saving of uploaded files and maintaining user file history.

• Chart Preparation:

• Prepares and formats data for visualization.

• Export Utilities:

• Functions to generate DOCX and PDF reports.

• Configuration File:

• Stores API endpoints, keys, and credentials.

• Easily adjustable for different deployment environments.

7. Non-Functional Requirements

• Performance:

• Fast response times for both API endpoints and UI interactions.

• Scalable architecture to handle increasing data loads and simultaneous users.

• Usability:

• Intuitive and minimalistic interface for non-technical users.

• Clear error messages and guidance when issues occur.

• Security:

• Secure user authentication (to be integrated with Supabase).

• Proper handling and storage of sensitive data.

• Secure API communications (HTTPS, API keys).

• Maintainability:

• Modular code structure to facilitate updates and integration of new features.

• Comprehensive documentation and in-code comments for ease of development.

• Portability:

• Designed to run on multiple platforms (local development, cloud deployment, Docker containerization).

8. Technical Architecture Overview

8.1 Frontend

• Tool: Streamlit

• Language: Python

• Functionality:

• Renders the user interface.

• Provides interactive components (login, file upload, chat, visualizations).

8.2 Backend

• Tool: FastAPI

• Language: Python

• Functionality:

• Hosts API endpoints for analysis and file processing.

• Integrates with third-party APIs (Deepseek, LangChain).

8.3 Data Processing

• Libraries: Pandas, Plotly, PandasAI (for natural language integration)

• Functionality:

• Cleans, processes, and analyzes uploaded data.

• Generates visualizations and report outputs.

8.4 Storage & Configuration

• Storage: Supabase (for authentication, file storage, and database management)

• Configuration:

• Central configuration file (config.py) for API keys and endpoints.

9. Milestones and Roadmap

1. Phase 1 – Foundation Setup (Weeks 1-2):

• Set up project structure and basic configuration.

• Implement user login and file upload functionalities.

2. Phase 2 – Core Analysis (Weeks 3-4):

• Develop and integrate the AI-driven analysis engine.

• Build the backend FastAPI endpoints.

3. Phase 3 – UI/UX and Visualization (Weeks 5-6):

• Develop the frontend components (chat interface, visualization with Plotly).

• Integrate interactive elements and error handling.

4. Phase 4 – Advanced Features and RAG Integration (Weeks 7-8):

• Implement template-based analysis.

• Integrate external document context (RAG module using LangChain).

5. Phase 5 – Testing, Optimization, and Deployment (Weeks 9-10):

• Conduct unit and integration testing.

• Optimize performance and prepare for production deployment.

10. Risks and Mitigation

• Data Privacy and Security:

• Mitigation: Use secure authentication, encrypt sensitive data, and follow best practices for API security.

• Performance Bottlenecks:

• Mitigation: Optimize data processing pipelines, implement caching, and plan for scalable infrastructure.

• User Adoption:

• Mitigation: Engage with early users to gather feedback, iterate on UI/UX, and provide comprehensive support documentation.