# Project Samarth

### Intelligent Q&A for India's Agricultural & Climate Data

Project Samarth is a functional prototype of an intelligent Q&A system that answers complex, natural language questions about India's agricultural economy and its relationship with climate patterns.

It's designed to reason across multiple, disparate government datasets in real-time, providing traceable, data-backed insights.

*(Add a screenshot of your running application here)*

## Core Mission

Government portals like data.gov.in host thousands of valuable datasets. However, this data exists in varied formats (e.g., annual crop reports vs. daily rainfall data), making cross-domain analysis difficult.

Project Samarth solves this by:

1. **Integrating** disparate data sources into a single, analysis-ready dataset.
2. **Understanding** natural language questions from a user.
3. **Generating & Executing** data analysis code on the fly.
4. **Synthesizing** a clear, citable answer based on the computed data.

## How It Works: System Architecture

This prototype uses a "Generative AI Data Agent" architecture.

1. **Phase 1: Data Integration (On Startup)**
   * The Python Flask server starts and loads three source CSVs:
     + agri.csv: Annual district-level crop production.
     + rain.csv: Daily district-level rainfall.
     + soil.csv: Daily district-level soil moisture.
   * A Pandas script cleans, aggregates (converts daily climate data to annual averages/totals), and merges these sources into a single master\_df (DataFrame) in memory.
2. **Phase 2: Intelligent Q&A (On-Demand)**
   * **1. User Asks:** A user types a question (e.g., "What was the total RICE PRODUCTION in Maharashtra in 2010?") into the web UI.
   * **2. Code Generation:** The question is sent to the Gemini API with a system prompt instructing it to generate Python/Pandas code to answer that question using the master\_df.
   * **3. Code Execution:** The Flask server receives the generated code and **executes it** (exec()) in a secure context against the master\_df.
   * **4. Answer Synthesis:** The *data result* from the code (e.g., 21345.87) is sent back to the Gemini API.
   * **5. Final Answer:** Gemini synthesizes this raw data into a natural language answer (e.g., "The total RICE PRODUCTION in Maharashtra in 2010 was 21,345.87 (1000 tons).") and displays it to the user, providing full traceability.

## Tech Stack

* **Backend:** Python, Flask
* **Data Analysis:** Pandas
* **Generative AI:** Google Gemini API
* **Frontend:** HTML, Tailwind CSS, JavaScript

## Setup & Installation

Follow these steps to run the prototype on your local machine.

### 1. Clone the Repository

git clone [https://github.com/your-username/project-samarth.git](https://github.com/your-username/project-samarth.git)  
cd project-samarth

### 2. Install Dependencies

Install the required Python libraries.

pip install -r requirements.txt

### 3. Add Data Files

This prototype does not include the data. You must download the source CSVs and place them in the root directory with the following names:

* agri.csv (ICRISAT District Level Data)
* rain.csv (IMD Rainfall Data)
* soil.csv (IMD Soil Moisture Data)

### 4. Add Your API Key

The application requires a Google Gemini API key to function.

1. Get your key from [**Google AI Studio**](https://www.google.com/search?q=https://aistudio.google.com/app/keys).
2. Open the app.py file.
3. Paste your key into the API\_KEY variable on line 16:  
   API\_KEY = "PASTE\_YOUR\_GEMINI\_API\_KEY\_HERE"

### 5. Run the Application

python app.py

The server will start, load and process the data files (this may take a moment), and then be available.

Open **http://127.0.0.1:5000** in your browser to use Project Samarth.

## Security Note

This prototype uses exec() to run AI-generated code. This is a significant security risk in a production environment. For this demonstration, it is used to prove the "code generation" architecture. A production-grade system would require a heavily sandboxed environment (e.g., Docker container, RestrictedPython) to execute the code safely.