DBMS - PROJECT

TEAM MATE 1

Name - DEVASNH PUROHIT
Sap Id - 590017030
Batch - 30
Course - DATABASE MANAGEMENT SYSTEM
Submitted To - MR. KALLURI SHREEF BABU

TEAM MATE 2

Name - SURAJ PUNDIR
Sap Id - 590012972
Batch - 30
Course - DATABASE MANAGEMENT SYSTEM
Submitted To - MR. KALLURI SHREEF BABU

INDEX

Smart Green Farm – Crop Recommendation

DBMS MINI PROJECT REPORT

1. Introduction

1.1 Background

Agriculture is one of the most important sectors of the Indian economy. Farmers often decide which crop to grow based on intuition, past experience or advice from local dealers. However, crop suitability actually depends on measurable parameters such as soil nutrients (N, P, K), pH, temperature, humidity and rainfall.

With the availability of agricultural datasets and AI/ML tools, it is now possible to build intelligent systems that suggest which crop will be most suitable under given conditions. At the same time, modern software engineering practices encourage end-to-end solutions: a user-friendly frontend, machine learning backend, and a database layer that stores all user interactions.

1.2 Motivation

Initially I built a simple AIML project on “Crop Recommendation System” that only ran locally and did not store user data anywhere. Later, in the DBMS course, our faculty asked us to extend our work into a full-fledged application with:

- A web frontend where a user can enter values,
- A backend model that performs prediction,
- And a database in which all user inputs and predictions are stored in a structured manner.

This motivated me to design Smart Green Farm, a web-based crop recommendation system that:

- Uses Machine Learning to recommend crops,

- Is implemented using Python and Streamlit,
- And stores every submission in a cloud database (Neon Postgres) with a local SQLite fallback.

1.3 Problem Statement

To design and implement a web-based AI/ML application that recommends the most suitable crop based on soil and weather conditions, and to store all user-submitted data in a structured SQL database which can be used for analysis and decision making.

1.4 Objectives

- To build a user-friendly web interface for entering agricultural parameters.
- To develop or reuse a trained machine learning model that recommends the most suitable crop.
- To store each prediction request (farmer details + conditions + recommended crop) in a relational database table.
- To deploy the system using GitHub + Streamlit Cloud, and to use Neon's cloud PostgreSQL as the primary database.
- To explore future extensions such as plant disease detection and crop yield prediction.

1.5 Scope of the Project

This project currently focuses on crop recommendation using standard parameters from the "Crop Recommendation" dataset. It supports:

- Different combinations of N, P, K, temperature, humidity, pH and rainfall.
- Farmer details such as name, city, and state.
- Cloud-based storage of all inputs and outputs.

Future versions can be extended to support sensor integration, disease detection, yield estimation and analytics dashboards.

2. LITERATURE SURVEY / EXISTING SYSTEMS

Traditionally, farmers rely on:

- Personal experience and trial-and-error,
- Advice from local experts or fertilizer sellers,
- Government advisory leaflets.

Several research papers and projects propose crop recommendation using decision trees, random forests, Naive Bayes, etc. Most of these solutions are:

- **Offline only** (run in Jupyter Notebook),
- Do not provide an interactive web frontend,
- Do not store user data for future analysis.

The **Smart Green Farm** project tries to bridge this gap by providing:

- A **modern web UI** using Streamlit,
- A **deployed model** accessible from browser,
- And a **cloud-backed database** to store all usage data.

3. SYSTEM ANALYSIS

3.1 Functional Requirements

1. The system shall allow the user to input:
    - Name, City, State

- Soil nutrients: Nitrogen (N), Phosphorus (P), Potassium (K)
- Temperature
- Humidity
- pH
- Rainfall

2. The system shall validate that all required fields are filled.
3. The system shall pass these values to a trained ML model and compute:
    - Recommended crop label,
    - Probability/score for all crops.
4. The system shall display:
    - The recommended crop in a highlighted card,
    - A table showing prediction probabilities for each crop.
5. The system shall store each submission in a database table with fields:
    - Farmer details, environmental conditions, predicted crop, probability vector, timestamp.
6. The system shall work both:
    - Locally using SQLite, and
    - In the cloud using Neon PostgreSQL.

3.2 Non-Functional Requirements

- **Usability:** Interface should be simple and intuitive for non-technical users.
- **Performance:** Prediction should be returned within a couple of seconds.
- **Reliability:** If cloud DB is not available, data should still be saved locally.
- **Scalability:** Database design should be flexible for adding new features (disease detection, yield, etc.).
- **Maintainability:** Code should be modular and version-controlled using GitHub.

3.3 Hardware & Software Requirements

Hardware:

- Any modern laptop/PC with at least 4 GB RAM.
- Internet connectivity for cloud deployment and Neon access.

Software:

- Operating System: Windows / Linux / macOS
- Programming Language: Python 3.x
- Libraries: `streamlit`, `pandas`, `numpy`, `joblib`, `sqlalchemy`, `psycopg2-binary`
- Database: Neon PostgreSQL (cloud) and SQLite (local file)

- Tools: Visual Studio Code, Git, GitHub, DB Browser for SQLite (for visualization).

4. SYSTEM DESIGN

4.1 Overall Architecture

The system follows a **three-layer architecture**:

1. **Presentation Layer (Frontend):**
    - Built using Streamlit.
    - Collects user inputs through sliders and text boxes.
    - Displays recommended crop and probabilities.
2. **Application Layer (Backend / ML Logic):**
    - Python functions that:
        - Load the trained model (`hybrid_crop_reco_model.pkl`),
        - Preprocess the input,
        - Call `model.predict()` and `model.predict_proba()`.
3. **Data Layer (Database):**
    - Primary: Neon PostgreSQL DB in the cloud.
    - Fallback: Local SQLite database `crop_recomendation.db`.

- Managed using SQLAlchemy ORM.

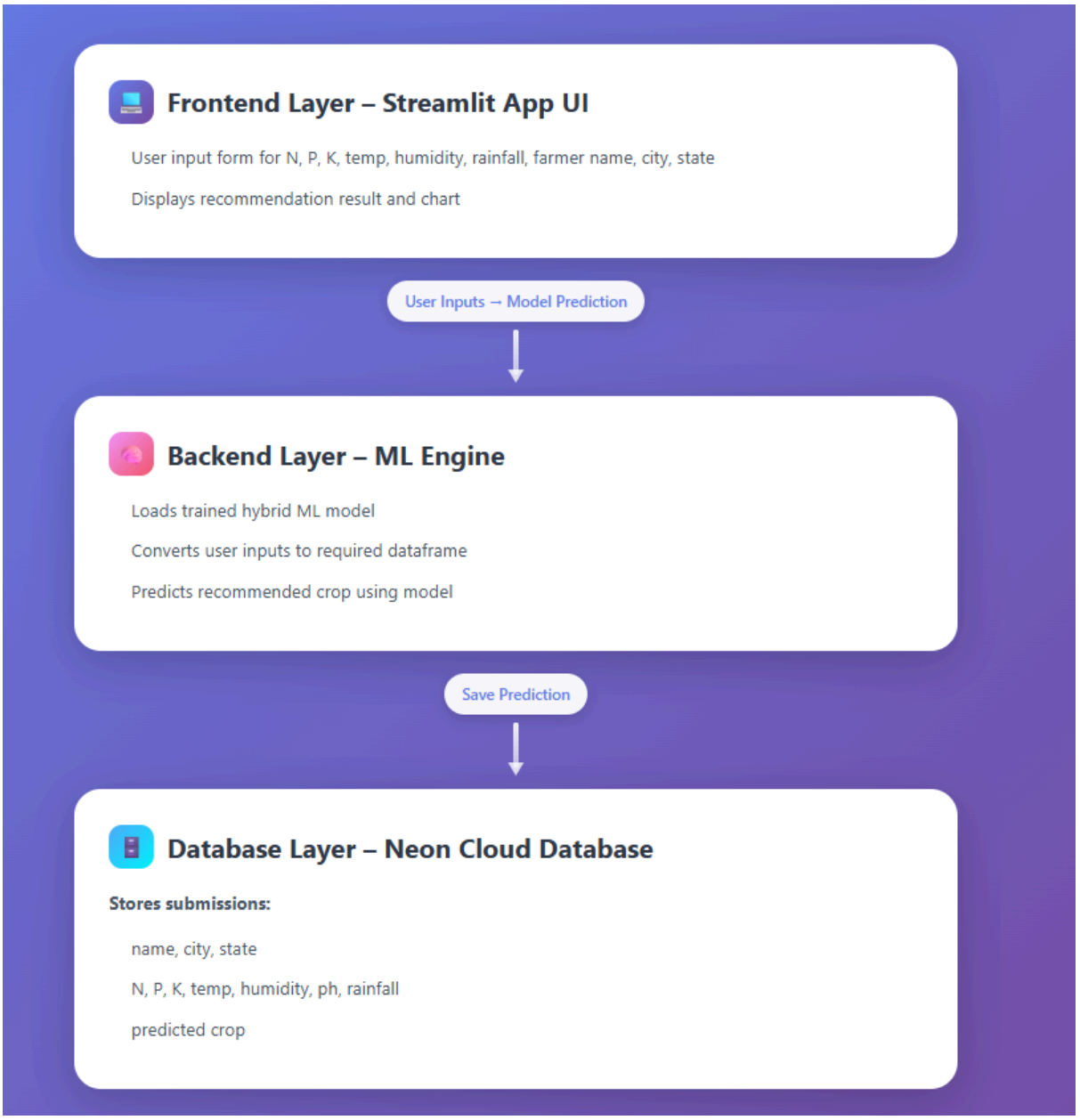


Figure 1: 3-Tier Architecture of Smart Green Farm

4.2 Data Flow

1. Farmer opens the web app.
2. Inputs “Name, City, State”.
3. Adjusts sliders for N, P, K, temperature, humidity, pH and rainfall.
4. Clicks on **“Recommend Crop”** button.
5. Values form a DataFrame row and are passed to the ML model.
6. Model predicts the best crop and probability distribution.
7. Frontend shows recommended crop + probability table.
8. Input values + result + timestamp are packaged into a dictionary.
9. SQLAlchemy’s `save_submission()` inserts this row into `crop_submissions` table on Neon/SQLite.

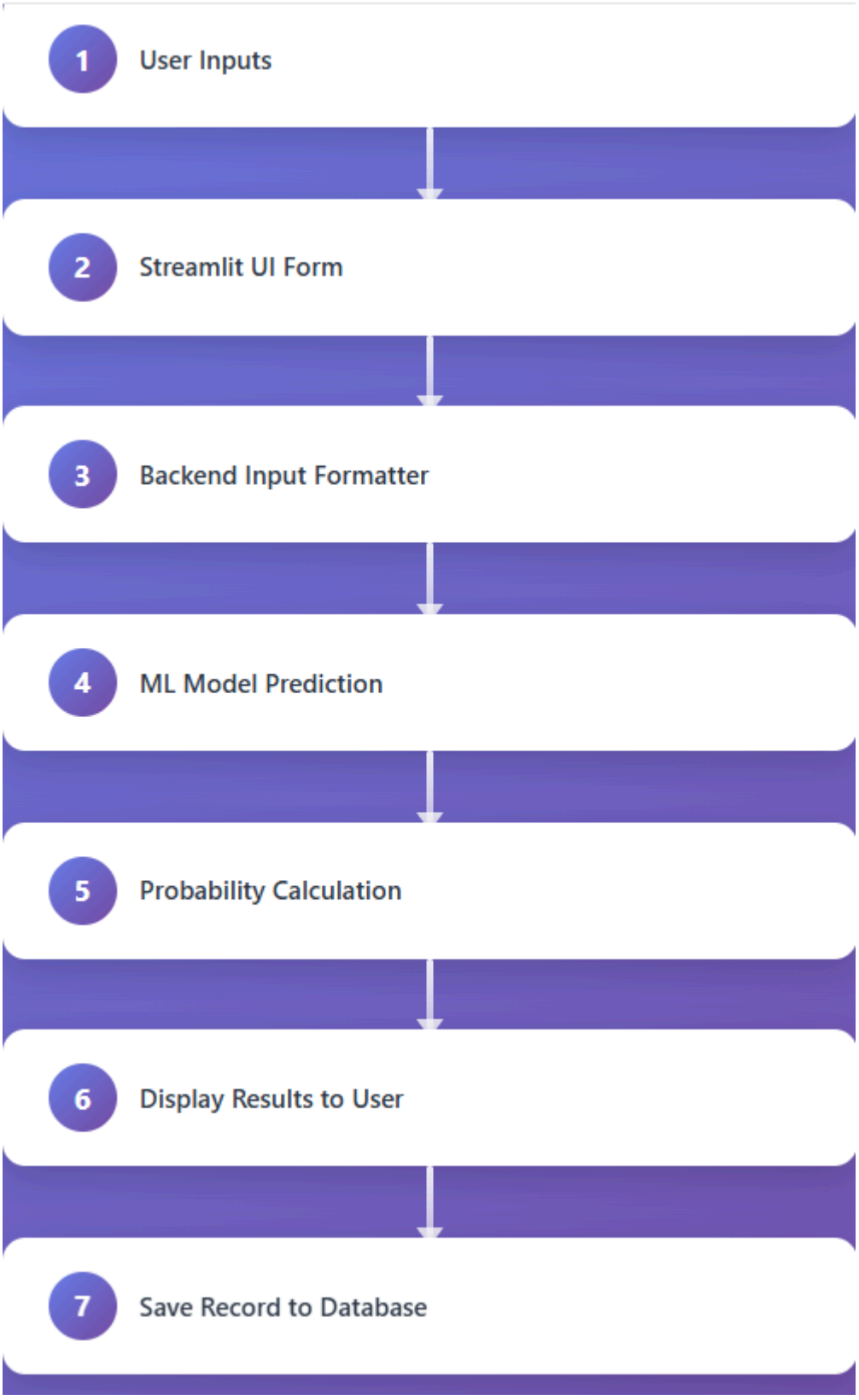


Figure 2: Data Flow Diagram for Crop Recommendation

4.3 Database Design (Schema)

Table Name: `crop_submissions`

Column Name	Data Type	Description
id	INTEGER (PK)	Auto-increment primary key
submitted_at	DATETIME	Timestamp (UTC) of submission
name	VARCHAR(100)	Farmer's name
city	VARCHAR(100)	City
state	VARCHAR(100)	State
N	FLOAT	Nitrogen value
P	FLOAT	Phosphorus value
K	FLOAT	Potassium value
temperature	FLOAT	Temperature in °C
humidity	FLOAT	Relative humidity (%)
ph	FLOAT	Soil pH
rainfall	FLOAT	Rainfall (mm)
predicted_crop	VARCHAR(256)	Crop recommended by model
predicted_proba	TEXT/JSON	Probability list for all crop classes

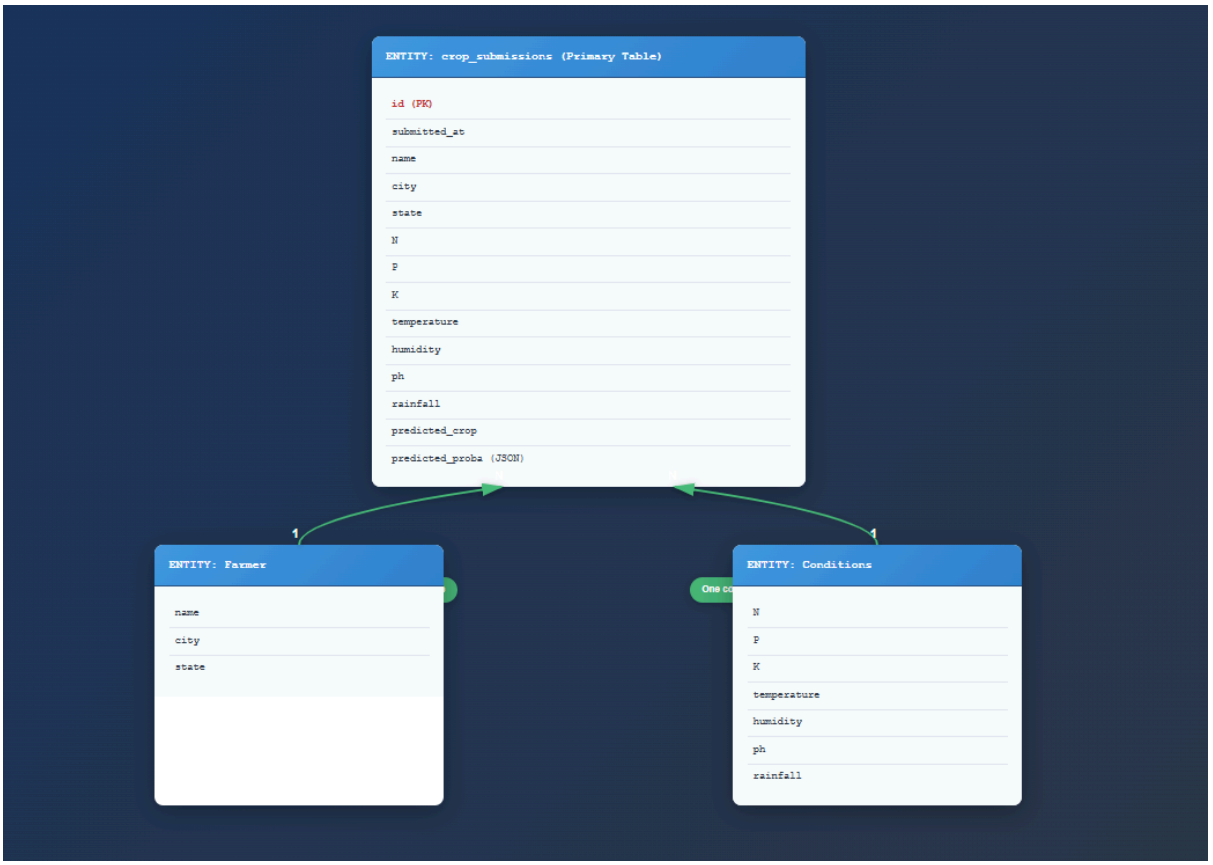


Figure 3: ER Diagram for crop_submissions

5. IMPLEMENTATION

5.1 Technology Stack

- ***Programming Language:*** *Python 3*
- ***Libraries:***
    - `pandas`, `numpy` *for data handling,*
    - `joblib` *for loading trained ML model,*
    - `streamlit` *for frontend UI,*
    - `sqlalchemy` *to interact with relational DB,*

    - `psycopg2-binary` *as Postgres driver for Neon.*

- ***Databases:***
    - *Neon PostgreSQL (cloud),*
    - *SQLite (local file).*
- ***Tools:***
    - *Visual Studio Code for development,*
    - *Git & GitHub for version control,*
    - *Streamlit Cloud for deployment.*

5.2 Dataset Description

The system uses the ***Crop Recommendation Dataset*** *(`Crop_recommendation.csv`). Main features:*

- ***Input Columns:*** *N, P, K, temperature, humidity, ph, rainfall*
- ***Output Column:*** *crop / label (e.g., rice, wheat, maize…)*

The dataset contains multiple rows where each row corresponds to environment conditions and the crop that grows well under those conditions.

5.3 Machine Learning Model

A hybrid ML model (`hybrid_crop_reco_model.pkl`) is pre-trained using scikit-learn (for example, RandomForest or ensemble). The training is done offline, and the final model is stored using `joblib.dump()`.

In the application:

@st.cache_resource(show_spinner=False)

def load_model():

return joblib.load(MODEL_PATH)

The loaded model supports:

- `model.predict(df_in)` → *recommended crop label.*
- `model.predict_proba(df_in)` → *probability scores for each crop class.*

You can briefly describe the training process (train/test split, accuracy) if you have those results.

5.4 Frontend – Streamlit Web Application

Streamlit allows building web apps with minimal code. Key UI components:

- `st.text_input` *for* ***Name, City, State****.*
- `st.slider` *for* ***N, P, K, temperature, humidity, pH, rainfall****.*
- `st.button("🌱 Recommend Crop")` *to trigger prediction.*
- `st.success`, `st.dataframe` *to show result and probability table.*
- *Custom CSS is injected using* `st.markdown(..., unsafe_allow_html=True)` *to give a* ***premium dark theme*** *look.*

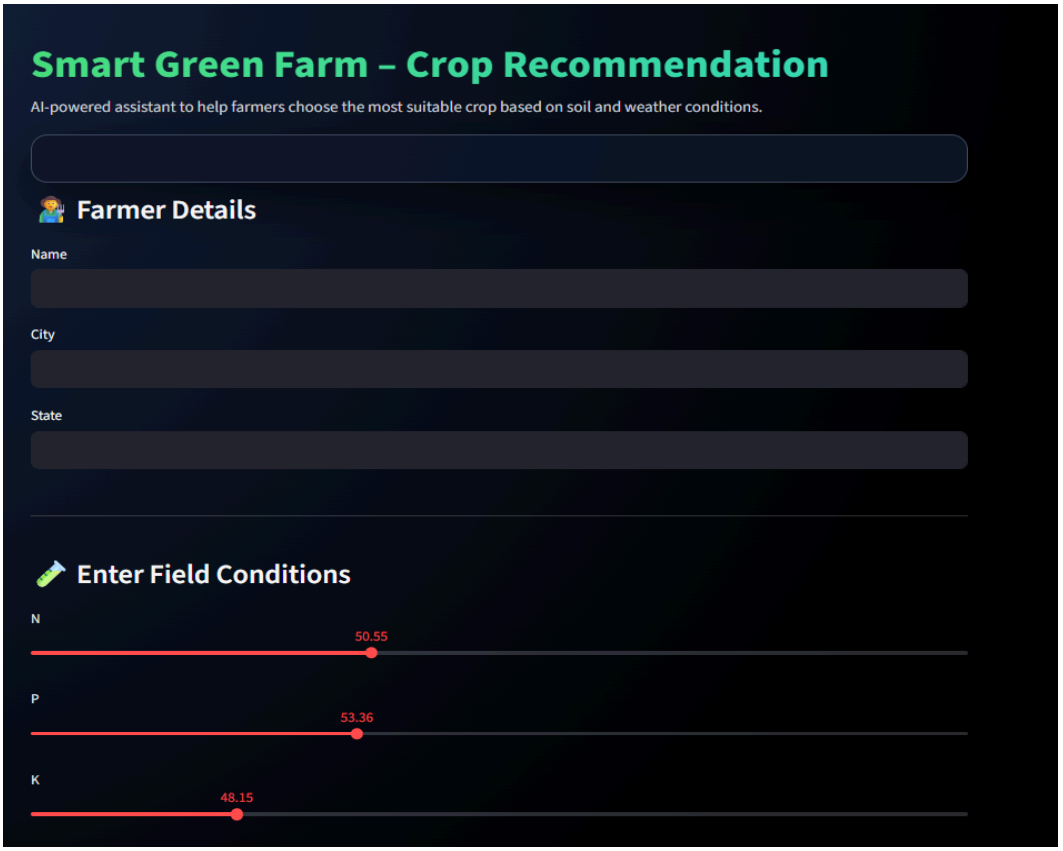


Figure 4: Streamlit Home Screen – Smart Green Farm

Figure 5: Sliders for Entering Field Conditions

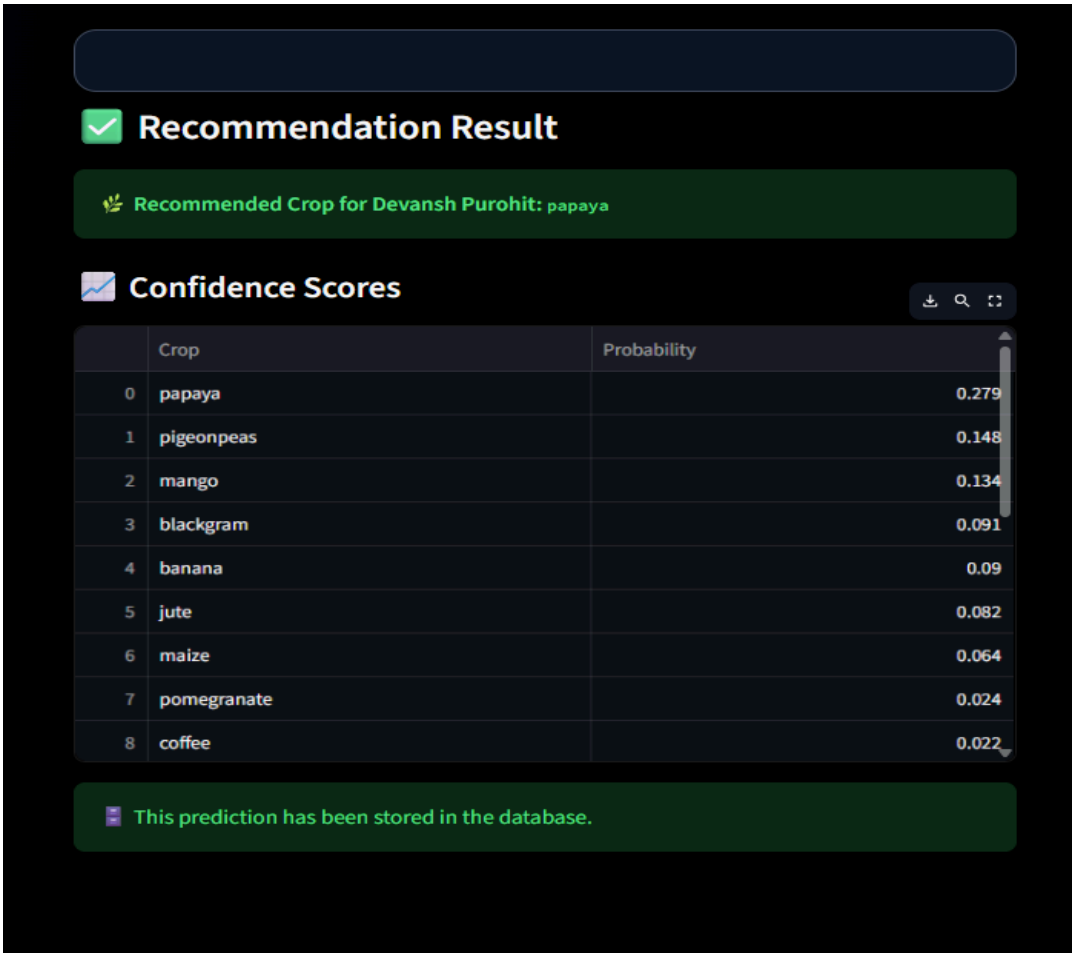


Figure 6: Recommendation Result and Confidence Scores

5.5 Backend Code Flow

Important Python functions:

`load_data()` *– reads CSV and calculates min/max/mean for each feature.*

`rng(col)` *– returns min, max, mean to set slider ranges.*

Button handler:

```
if predict_clicked:
    df_in = pd.DataFrame([vals], columns=feature_cols)
    crop = model.predict(df_in)[0]
    proba = model.predict_proba(df_in)[0]
```

```
# display result

# call save_submission(...)
```

The backend is tightly integrated with Streamlit; there is no separate REST API since Streamlit runs Python server itself.

5.6 Database Layer – Neon Cloud DB & SQLite Fallback

Using SQLAlchemy, the following steps are implemented:

1. ***Reading Neon URL from secrets:***

```
if "NEON_DB_URL" in st.secrets:
    DB_URL = st.secrets["NEON_DB_URL"]
```

2. ***Creating Engine and fallback:***

```
ENGINE = create_engine(DB_URL, pool_pre_ping=True, future=True)
# if that fails -> create SQLite engine with
sqlite_url = f"sqlite:///{DB_PATH.as_posix()}"
ENGINE = create_engine(sqlite_url, echo=False, future=True)
```

3. ***Defining table:***

```
submissions_table = Table(
    "crop_submissions",
    metadata,
    Column("id", Integer, primary_key=True, autoincrement=True),
    ...
)
```

```
metadata.create_all(ENGINE)
```

4. ***Insert using `save_submission`:***

```
with ENGINE.begin() as conn:
    conn.execute(submissions_table.insert().values(**ins))
```

*This ensures that **every prediction** with user details and environmental parameters is stored in the DB.*

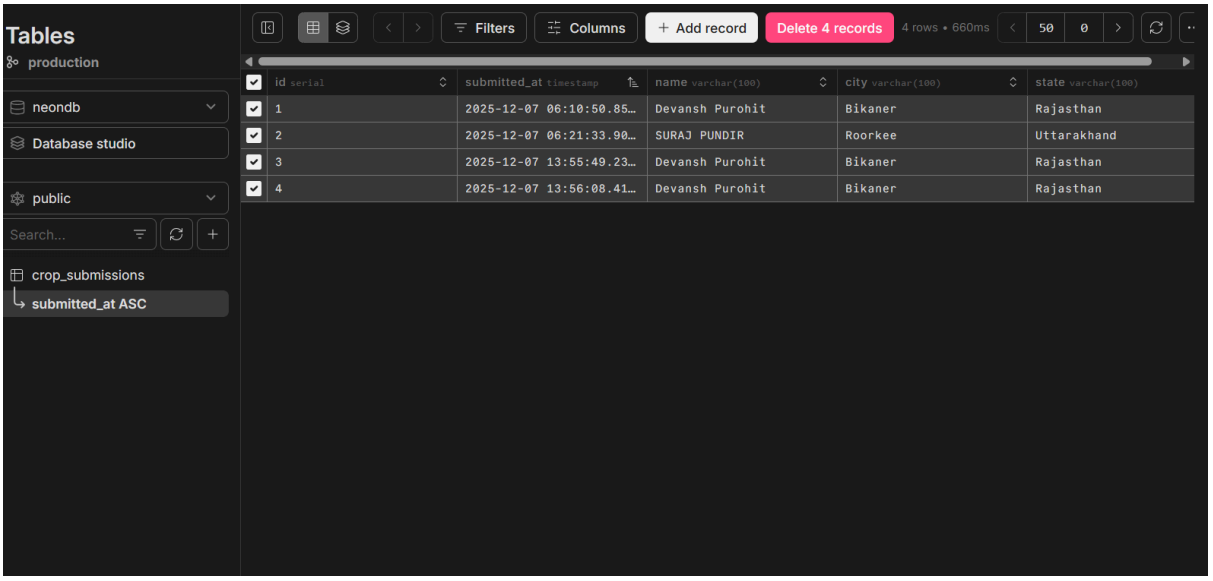


Figure 7: Neon Console – crop_submissions Table

5.7 GitHub & Streamlit Cloud Deployment

All project files are kept in a local folder:

`app.py`

`Crop_recommendation.csv`

`hybrid_crop_reco_model.pkl`

`requirements.txt`

1.

A Git repository is initialized:

```
git init

git add .

git commit -m "Initial commit - Smart Green Farm"

git remote add origin
https://github.com/<username>/smart-green-farm.git

git push -u origin main
```

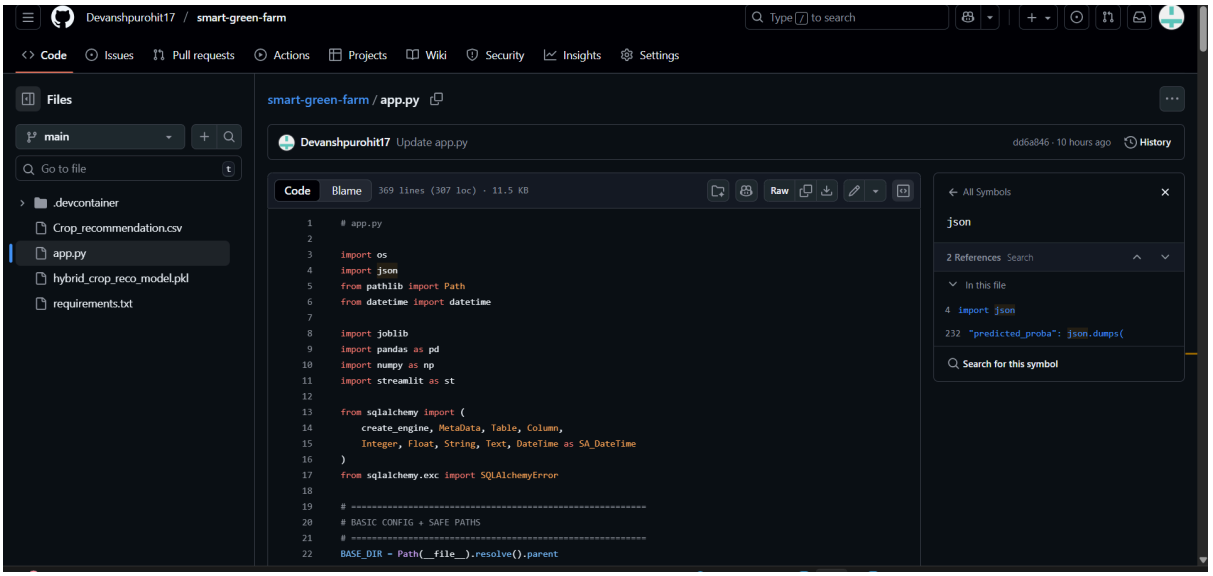


Figure 8: GitHub Repository Structure

2. *Deployment steps:*
    - *Open **Streamlit Community Cloud**.*
    - *Click **"New app"** → choose the GitHub repo and select `app.py`.*
    - *Set environment secrets (NEON_DB_URL).*
    - *Click **Deploy**.*
3. *Now the app is live on a `https://<your-app>.streamlit.app` URL and can be accessed from any browser.*

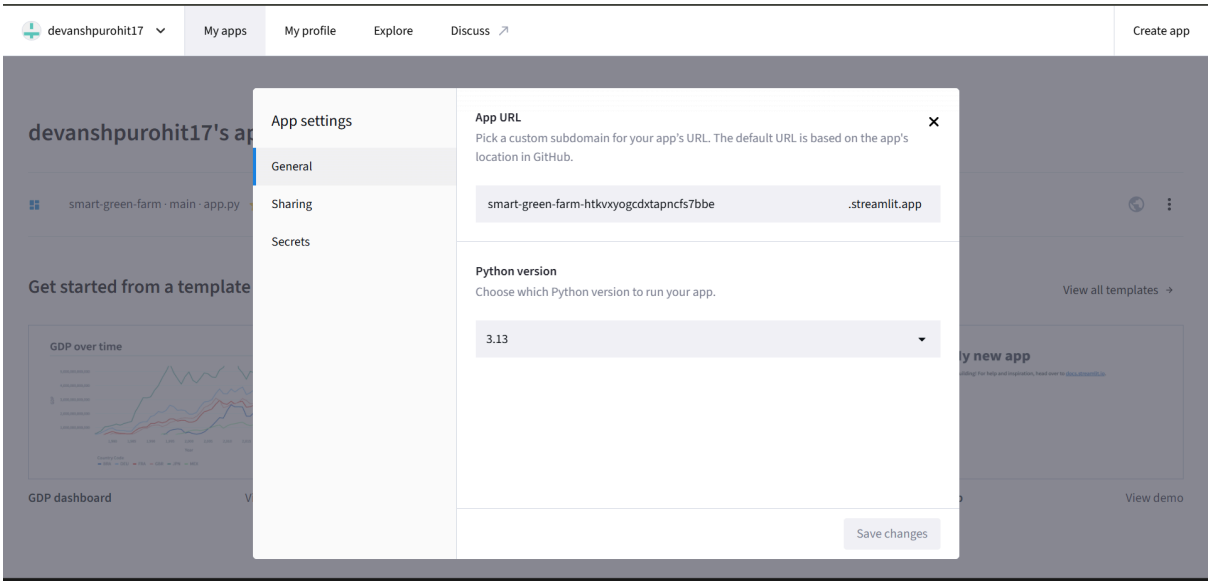


Figure 9: Streamlit Deployment Settings Screenshot

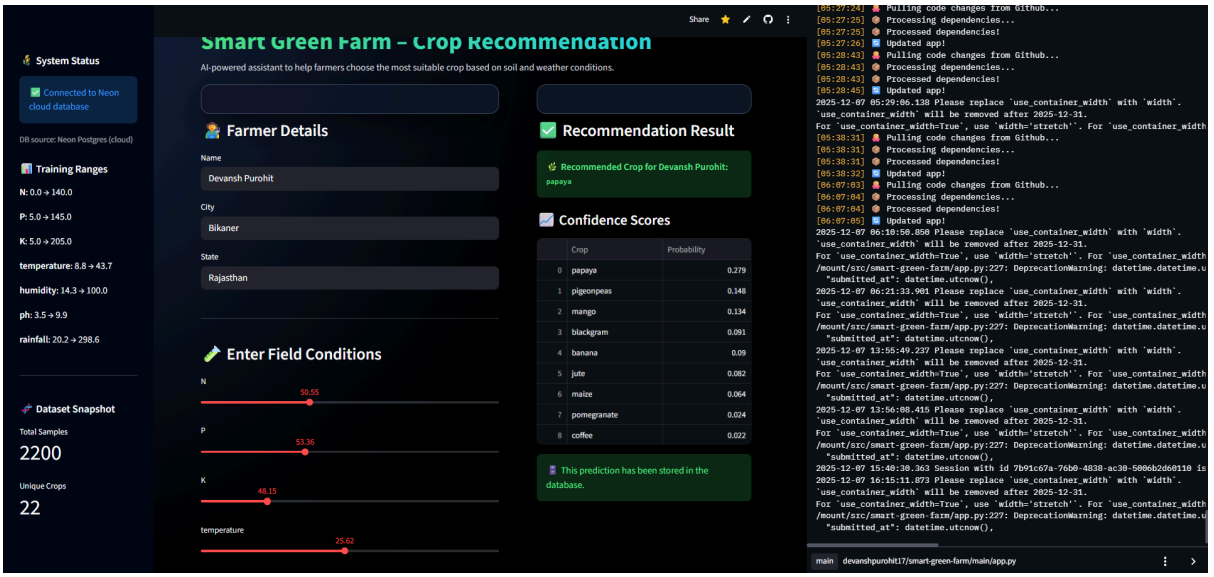


Figure 10: Live Deployed App Running in Browser

---

6. WORKING OF THE SYSTEM (STEP-BY-STEP)

1. User Access:

- *User opens the web URL in a browser.*

2. Farmer Details Input:

- *Enters **Name**, **City**, **State** in text fields.*

3. Soil & Weather Input:

- *Uses sliders to set values for N, P, K, temperature, humidity, pH, rainfall.*
- *Sliders are bounded using min/max from the training dataset.*

4. Recommendation Trigger:

- *User clicks **"🌱 Recommend Crop"** button.*

5. Prediction:

- *Inputs are assembled into a pandas DataFrame.*
- *Model loaded from `hybrid_crop_reco_model.pkl` runs `predict()` and `predict_proba()`.*

6. Result Display:

- *Frontend shows recommended crop in a green card.*
- *A table displays probability of each crop.*

7. Database Storage:

- *`save_submission()` is called.*
- *If Neon is reachable → row inserted into cloud PostgreSQL.*
- *If not → row inserted into local SQLite `crop_recomendation.db`.*

8. Teacher/Developer View:

- *In Neon console, teacher can see all entries with filters.*
- *Locally, DB Browser for SQLite can show the same table.*

Figure 11: Sequence Diagram of End-to-End Workflow

7. TESTING & RESULTS

7.1 Functional Testing

- *Tested with multiple valid ranges of N, P, K etc.*
- *Verified that:*
    - *Predictions are generated,*

- *Data is inserted into the database,*
- *Mandatory fields (Name, City, State) are enforced.*

7.2 Sample Test Case

name varchar(100)	city varchar(100)	state varchar(100)	N double precision	P double precision
Devansh Purohit	Bikaner	Rajasthan	50.551818181818184	53.36272727272727
SURAJ PUNDIR	Roorkee	Uttarakhand	50	53.36272727272727
Devansh Purohit	Bikaner	Rajasthan	50.551818181818184	53.36272727272727
Devansh Purohit	Bikaner	Rajasthan	32	20

K double precision	temperature double prec...	humidity double precision	ph double precision	rainfall double precision
48.14909090909091	25.616243851779544	71.48177921778637	6.469480065256364	103.46365541576817
48.14909090909091	25.616243851779544	60	6.469480065256364	103.46365541576817
48.14909090909091	25.616243851779544	71.48177921778637	6.469480065256364	103.46365541576817
23	25.616243851779544	29	6	74

predicted_crop varchar(...	predicted_proba text
papaya	[0.0, 0.08987192654839...
mango	[0.0, 0.05774229691876...
papaya	[0.0, 0.08987192654839...
mothbeans	[0.0, 0.0, 0.015163678...

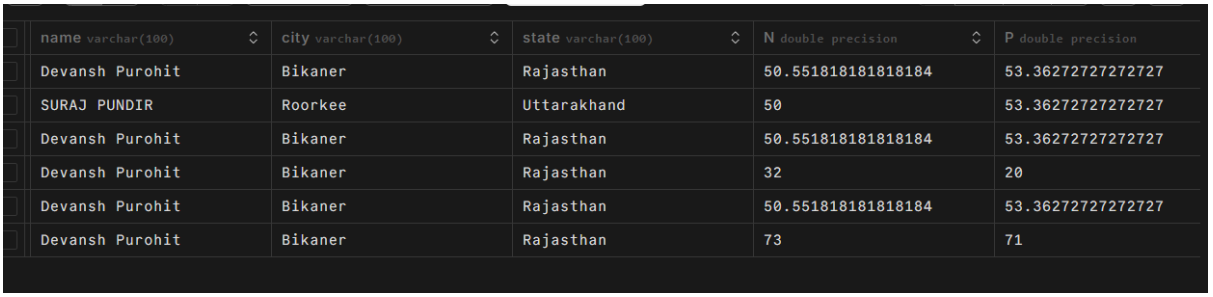
7.3 Screenshots

- Screenshot of different examples where crops like rice, wheat, maize, chickpea etc. are recommended.
- Screenshot of Neon DB table containing multiple farmer records.

Figure 12: Test Case 1 – Input Screenshot

	Crop	Probability
0	banana	0.321
1	coffee	0.236
2	jute	0.092
3	papaya	0.079
4	maize	0.075
5	chickpea	0.072
6	pigeonpeas	0.065
7	rice	0.035
8	mango	0.01

Figure 13: Test Case 1 – Output Screenshot



name varchar(100)	city varchar(100)	state varchar(100)	N double precision	P double precision
Devansh Purohit	Bikaner	Rajasthan	50.551818181818184	53.36272727272727
SURAJ PUNDIR	Roorkee	Uttarakhand	50	53.36272727272727
Devansh Purohit	Bikaner	Rajasthan	50.551818181818184	53.36272727272727
Devansh Purohit	Bikaner	Rajasthan	32	20
Devansh Purohit	Bikaner	Rajasthan	50.551818181818184	53.36272727272727
Devansh Purohit	Bikaner	Rajasthan	73	71

Figure 14: Neon Table Showing Inserted Rows

8. ADVANTAGES & APPLICATIONS

Advantages

- **User-friendly:** Simple web interface that even non-technical users can operate.
- **AI-driven:** Uses machine learning to provide data-driven crop recommendations.
- **Cloud-backed:** Submissions are stored in Neon cloud DB, enabling later analysis.
- **Portable:** Can run locally with SQLite or on cloud via Streamlit.
- **Modular:** Clear separation of frontend, ML logic and database layer.

Applications

- As a **decision-support tool** for small and medium farmers.
- Pilot project for **Krishi Vigyan Kendras** or agricultural universities.
- Teaching aid for **AI & DBMS courses** to demonstrate end-to-end system design.

---

9. LIMITATIONS

- Model accuracy is limited by the **quality and size of the training dataset**.
- Collecting real-time parameters (e.g., sensor data) is not yet integrated.
- Internet is required to access the deployed app and Neon DB.
- Currently recommends only **one crop at a time**, does not consider market prices or crop rotation.

---

10. FUTURE SCOPE

I plan to extend **Smart Green Farm** with the following features:

1. **Plant Disease Detection:**
    - Integrate a CNN model that detects plant leaf diseases from images.
    - User can upload a leaf photo; the system diagnoses disease and suggests treatment.
2. **Yield Prediction & Total Production Analytics:**
    - Use historical data plus user submissions to estimate yield statistics:
        - Predict **expected yield** for a particular crop in a given area.
        - Provide **total production estimates** for a particular crop in a taluka/district.
3. **Multi-criteria Recommendation:**
    - Combine soil suitability, market price and risk factors to give ranked recommendations.
4. **Farmer Dashboard:**
    - Each farmer can log in to see their past submissions, recommended crops, and outcomes.
5. **Mobile App Integration:**

- Wrap the Streamlit application using frameworks like Streamlit's mobile support or a React Native frontend calling the same backend.

---

11. CONCLUSION

The **Smart Green Farm – Crop Recommendation System** successfully demonstrates how **Artificial Intelligence, Web Technologies and Databases** can be integrated to solve a real-world agricultural problem.

The project started from a simple ML model and evolved into a complete application with:

- A **Streamlit web frontend**,
- A **trained crop recommendation model**,
- And a **Neon/SQLite database** that stores farmer inputs and predictions in a structured relational table.

From a DBMS perspective, the project implements:

- Table design, primary keys and data types,
- DDL (CREATE TABLE via SQLAlchemy),
- DML operations (INSERT, SELECT),
- And integration of application logic with back-end storage.

From an AIML perspective, it showcases how trained models can be deployed in production-like environments. The project can be enhanced into a full-fledged smart agriculture platform with disease detection, yield prediction and advanced analytics.

---

12. REFERENCES

1. Crop Recommendation Dataset (Kaggle / Source used).

2. Streamlit Documentation – https://docs.streamlit.io
3. SQLAlchemy Documentation – https://docs.sqlalchemy.org
4. Neon Postgres Documentation – https://neon.tech/docs

PROJECT LINK

https://smart-green-farm-htkvxyogcdxtapncfs7bbe.streamlit.app/