**CROP**

**RECOMMENDATION**

The Crop Recommendation Flask Web App is a web application that recommends the best crop to grow based on soil and climate conditions. The project involves building a machine learning model that can predict the crop yield based on several parameters such as soil pH, temperature, rainfall, humidity, and crop type. The machine learning model is then integrated into a Flask web application to provide farmers with a simple and easy-to-use tool for crop selection.

**Functional requirements:**

**2.1 User Registration and Authentication:**

* The system should allow users to register with their email address and password.
* Users should be able to log in securely to access the system.

**2.2 Input Data Collection:**

* The system should provide forms or interfaces for users to input data such as temperature, humidity, pH, rainfall, and soil nutrient levels (N, P, K).
* Input forms should validate data entries and provide appropriate error messages for invalid inputs.

**2.3 Crop Recommendation Algorithm:**

* The system should implement a crop recommendation algorithm based on input data provided by the user.
* The algorithm should analyze the input data, including temperature, humidity, pH, rainfall, and soil nutrient levels, to recommend suitable crops for cultivation.
* Recommendations should consider crop-specific requirements for optimal growth conditions.

**2.4 Crop Information Display:**

* Upon receiving input data and processing recommendations, the system should display recommended crops to the user.
* Crop recommendations should include details such as crop name, growth cycle, water requirements, soil pH preference, and nutrient needs.
* Users should be able to view additional information about recommended crops, including cultivation practices and potential yields.

**2.5 User Feedback and Adjustment:**

* The system should allow users to provide feedback on recommended crops.
* Users should be able to indicate whether they accept or reject a recommended crop based on their preferences or constraints.
* The system should use user feedback to refine future recommendations and improve the accuracy of crop suggestions over time.

**3 . Non functional requirements:**

**3.1 Security:**

* User data should be protected against unauthorized access, modification, or disclosure.

**3.2 Performance:**

* The system should process recommendations quickly and efficiently, even with large datasets.
* Response times for user interactions should be minimal to provide a seamless user experience.

**3.3 Usability:**

* Input forms should be user-friendly and easy to navigate.
* The system should provide clear and understandable recommendations to users.

**3.4 Reliability:**

* The system should be reliable and available for use at all times, with minimal downtime for maintenance or updates.

**3.5 Scalability:**

* The system should be able to handle an increasing number of users and data inputs without compromising performance**.**

**Ensuring security in our website (future approach)**

1. **Access Control:**
   * Implement role-based access control to restrict unauthorized access to different parts of the website.
   * Ensure that only authenticated users can access sensitive functionalities.
2. **Secure Authentication:**
   * Enforce strong authentication methods such as multi-factor authentication (MFA) to verify user identities.
   * Safeguard user credentials and session management processes.
3. **Input Validation:**
   * Validate and sanitize all user input to prevent injection attacks such as SQL injection and cross-site scripting (XSS).
4. **Data Encryption:**
   * Encrypt sensitive data both at rest and in transit, including user credentials and payment information.
5. **Security Headers:**
   * Utilize security headers like Content Security Policy (CSP) and HTTP Strict Transport Security (HSTS) to enhance website security.
6. **Regular Security Testing**:
   * Conduct routine vulnerability assessments and penetration testing to identify and address security weaknesses.
7. **Secure Hosting Environment:**
   * Choose a reputable hosting provider and ensure that the hosting infrastructure is securely configured and regularly updated.
8. **Logging and Monitoring:**
   * Implement logging and monitoring mechanisms to track user activity and detect suspicious behavior.
   * Monitor access logs and system logs for potential security incidents.
9. **User Education:**
   * Provide users with security awareness training, including guidance on creating strong passwords and recognizing phishing attempts.

**Unit testing:**

**Testing predict.html:**

**import unittest**

**from app import app**

**class FlaskTest(unittest.TestCase):**

**# Ensure that the welcome page returns a 200 status code**

**def test\_welcome\_page(self):**

**tester = app.test\_client(self)**

**response = tester.get('/')**

**statuscode = response.status\_code**

**self.assertEqual(statuscode, 200)**

**# Ensure that the predict route returns a 200 status code**

**def test\_predict\_route(self):**

**tester = app.test\_client(self)**

**response = tester.post('/predict', data=dict(Temperature=30, Humidity=60, PH=6.5, rain\_fall=50, n=10, p=15, k=20))**

**statuscode = response.status\_code**

**self.assertEqual(statuscode, 200)**

**# Ensure that the prediction form contains necessary elements**

**def test\_prediction\_form\_elements(self):**

**tester = app.test\_client(self)**

**response = tester.get('/')**

**self.assertIn(b'Temperature:', response.data)**

**self.assertIn(b'Humidity:', response.data)**

**self.assertIn(b'PH in the soil:', response.data)**

**self.assertIn(b'Rain Fall:', response.data)**

**self.assertIn(b'Potassium (K) in the Soil:', response.data)**

**self.assertIn(b'Nitrogen (N) in the Soil:', response.data)**

**self.assertIn(b'Phosphorous (P) in the Soil:', response.data)**

**self.assertIn(b'Predict', response.data)**

**# Ensure that the prediction form is submitted correctly**

**def test\_prediction\_form\_submission(self):**

**tester = app.test\_client(self)**

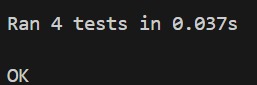
**response = tester.post('/predict', data=dict(Temperature=30, Humidity=60, PH=6.5, rain\_fall=50, n=10, p=15, k=20))**

**self.assertIn(b'Recomended Crop is MOTH BEANS', response.data)**

**if \_\_name\_\_ == '\_\_main\_\_':**

**unittest.main()**

**Output:**

****

**Integration and system testing:**

**Test case:** if the predict page loads successfully

    # Ensure that the predict route returns a 200 status code

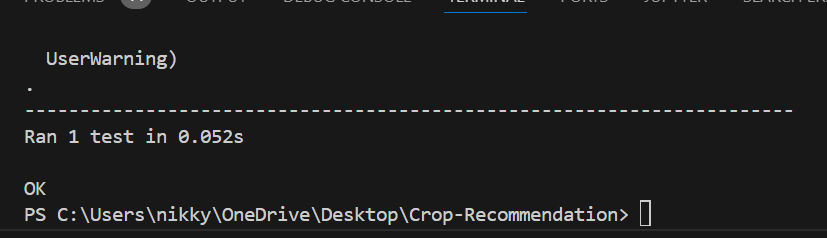
    def test\_predict\_route(self):

        tester = app.test\_client(self)

        response = tester.post('/predict', data=dict(Temperature=30, Humidity=60, PH=6.5, rain\_fall=50, n=10, p=15, k=20))

        statuscode = response.status\_code

        self.assertEqual(statuscode, 200)



**Output**

**Test case:** if response page loads successfully

class FlaskTest(unittest.TestCase):

     # Ensure that the welcome page returns a 200 status code

    def test\_welcome\_page(self):

        tester = app.test\_client(self

)

        response = tester.get('/')

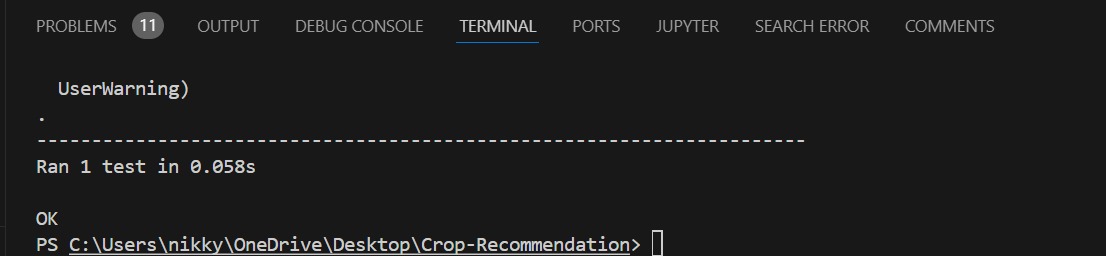
        statuscode = response.status\_code

        self.assertEqual(statuscode, 200)

if \_\_name\_\_ == '\_\_main\_\_':

    unittest.main()

**Output:**



**Testcase:** if expected predictions loads successfully

       # Ensure that the predict route returns the expected prediction

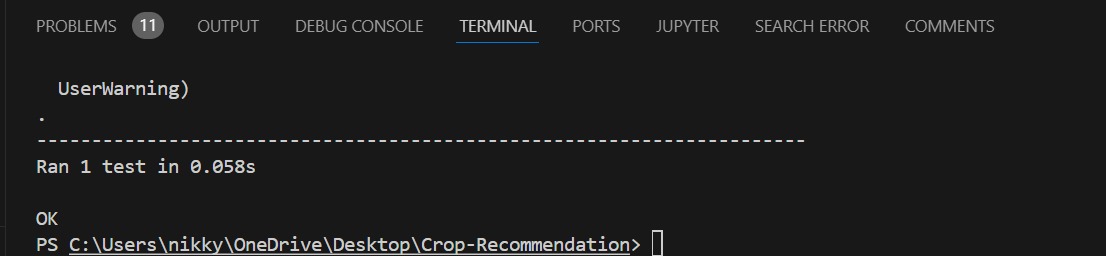
    def test\_predict\_output(self):

        tester = app.test\_client(self)

        response = tester.post('/predict', data=dict(Temperature=30, Humidity=60, PH=6.5, rain\_fall=50, n=10, p=15, k=20))

        self.assertIn(b'Recommended Crop is', response.data)

**Output:**



**Description:**

**1.test\_welcome\_page:** This test ensures that the welcome page of the application returns a status code of 200, indicating that the page is successfully loaded.

**2.test\_predict\_route:** This test ensures that the prediction route of the application returns a status code of 200 when posting valid input data. It checks whether the prediction route is functioning correctly.

**3.test\_prediction\_form\_elements:** This test verifies that the prediction form on the welcome page contains all the necessary elements such as input fields for temperature, humidity, pH, rainfall, potassium (K), nitrogen (N), phosphorous (P), and a predict button.

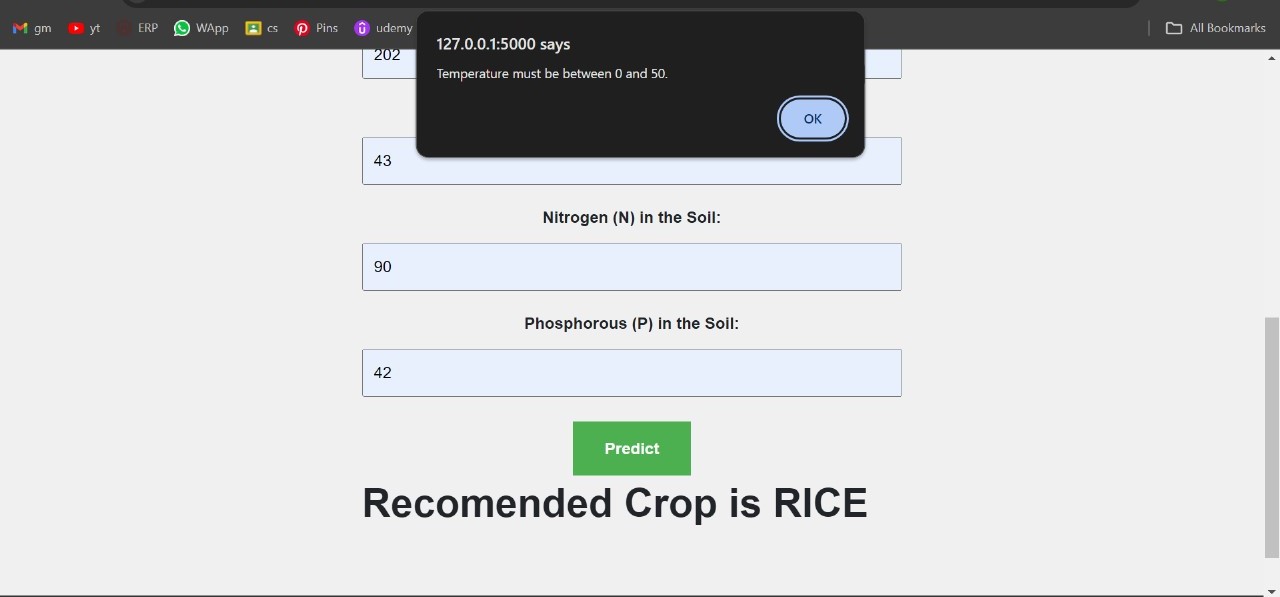
**4.test\_prediction\_form\_submission:** This test checks whether the prediction form is submitted correctly. It posts sample input data to the prediction route and verifies that the response contains the expected recommendation for the crop, which in this case is "MOTH BEANS".

**Invalid cases:**

Users are restricted choosing invalid cases by providing options to the inputs

but still if they enter the system takes the boundary case and tells the

prediction



import unittest

from app import app

class FlaskTest(unittest.TestCase):

    # Ensure that the welcome page returns a 200 status code

    def test\_welcome\_page(self):

        tester = app.test\_client(self)

        response = tester.get('/')

        statuscode = response.status\_code

        self.assertEqual(statuscode, 200)

    # Ensure that the predict route returns a 200 status code

    def test\_predict\_route(self):

        tester = app.test\_client(self)

        response = tester.post('/predict', data=dict(Temperature=30, Humidity=60, PH=6.5, rain\_fall=50, n=10, p=15, k=20))

        statuscode = response.status\_code

        self.assertEqual(statuscode, 200)

    # Ensure that the prediction form contains all necessary elements

    def test\_prediction\_form\_elements(self):

        tester = app.test\_client(self)

        response = tester.get('/')

        self.assertIn(b'Temperature:', response.data)

        self.assertIn(b'Humidity:', response.data)

        self.assertIn(b'PH in the soil:', response.data)

        self.assertIn(b'Rain Fall:', response.data)

        self.assertIn(b'Potassium (K) in the Soil:', response.data)

        self.assertIn(b'Nitrogen (N) in the Soil:', response.data)

        self.assertIn(b'Phosphorous (P) in the Soil:', response.data)

        self.assertIn(b'Predict', response.data)

    # Ensure that the prediction form is submitted correctly

    def test\_prediction\_form\_submission(self):

        tester = app.test\_client(self)

        response = tester.post('/predict', data=dict(Temperature=30, Humidity=60, PH=6.5, rain\_fall=50, n=10, p=15, k=20))

        self.assertIn(b'<h1><b>RECOMENDED CROP IS MOTH BEANS</b></h1>', response.data)

if \_\_name\_\_ == '\_\_main\_\_':

    unittest.main()

**Output:**

****

**SRS fulfillment:**

|  |  |  |
| --- | --- | --- |
| Functional Requirement | Actions Needed to Meet Requirement | Met (Yes/No) |
| 2.1 User Registration and Authentication | Implement user registration functionality where users can register using their email address and password. | NO |
|  | Develop secure login functionality for registered users to securely access the system. | NO |
| 2.2 Input Data Collection | Design forms or interfaces for users to input data, including temperature, humidity, pH, rainfall, and soil nutrient levels (N, P, K). | Yes |
|  | Implement data validation checks in input forms to ensure data accuracy and provide error messages for invalid inputs. | Yes |
| 2. 3 Crop Recommendation Algorithm | Develop a crop recommendation algorithm that analyzes input data to suggest suitable crops for cultivation. | Yes |
|  | Ensure the algorithm considers crop-specific requirements for optimal growth conditions. | Yes |
| 2. 4 Crop Information Display | Create a display interface to show recommended crops to the user, including details like crop name, growth cycle, water requirements, soil pH preference, and nutrient needs. | Yes |
|  | Provide additional information about recommended crops, including cultivation practices and potential yields. | Yes |
| 2.5 User Feedback and Adjustment | Enable users to provide feedback on recommended crops, indicating acceptance or rejection based on preferences or constraints. | Yes |
|  | Utilize user feedback to refine future recommendations and enhance the accuracy of crop suggestions over time. | Yes |

|  |  |  |
| --- | --- | --- |
| Non-Functional Requirement | Actions Needed to Meet Requirement | Met (Yes/No) |
| Security | Implement robust authentication mechanisms and data encryption protocols to protect user data from unauthorized access. | No |
|  | Regularly update and patch system vulnerabilities to mitigate potential security risks and ensure data integrity. | No |
| Performance | Optimize algorithm efficiency and database queries to ensure fast processing of recommendations, even with large datasets. | No |
|  | Utilize caching mechanisms and load balancing techniques to improve system responsiveness and handle increased user loads. | No |
| Usability | Design intuitive and user-friendly input forms with clear instructions to streamline data input processes. | No |
|  | Provide user-friendly interfaces for displaying recommendations, ensuring that information is presented in a clear manner. | No |
| Reliability | Implement redundancy and failover mechanisms to ensure high availability and minimize downtime for maintenance activities. | No |
|  | Conduct regular system backups and perform thorough testing to identify and address potential points of failure. | No |
| Scalability | Design the system architecture to be scalable, allowing it to accommodate growing user bases and data volumes. | No |
|  | Implement horizontal scaling strategies to distribute workload across multiple servers and maintain performance. | No |

**Conclusion:**The system is almost ready for deployment

Here's a general overview of the project:

**1.Data collection:** Collect soil and climate data from reliable sources such as the National Soil Information System and the National Oceanic and Atmospheric Administration (NOAA).

**2.Data preprocessing:** Clean and prepare the data for use in the machine learning model.

**3.Feature selection:** Select the most important features that can affect the crop yield, such as soil pH, temperature, rainfall, humidity, and crop type.

**4.Model training:** Train a machine learning model using the preprocessed data and the selected features.

**5.Model evaluation:** Evaluate the performance of the machine learning model to ensure it can accurately predict the crop yield.

**6.Flask app development:** Develop a Flask web application that allows users to input soil and climate parameters and get a recommendation for the best crop to grow.

**7.Deployment:** Deploy the web application to a server so that it can be accessed by users.

Overall, the Crop Recommendation Flask Web App project can be a valuable tool for farmers to increase their crop yield and improve their farming practices.

**Algorithms used:**

Random Forest Classifier is used for development of models.

Only three algorithms are used to predict the output. They are Logistic Regression, XGBoost and Random Forest.\

1. Accuracy of the model using Logistic Regression is 95%.

2. Accuracy of the model using Random Forest Classifier is 99%.

3. Accuracy of the model using XGBoost Classifier is 99%.

**Compatibility:**

We have performed on google chrome, Microsoft edge and Mozilla Firefox .It runs successfully .and the running time is best in mozilla firefox.

**Scalability and security:**

scalability and security is still pending and we need to create a login or signup page for security purposes.

**Usability:**

our user interface is aesthetically pleasing  and simple .users may like our design

Thank you

     Nikshiptha.Sonajoke(B22CS050)

     Banoth.Sri Kowshika raj(B22CS018)