# YIELD PREDICTION SYSTEM

## Technical Answers for Real World Problems

## CSE1901

**Team Members:**

Megha Nath (20BCE1581)

Anegha Jain (20BCE1547)

Aana Kakroo (20BAI1138)

Shashank Sharma (20BCE1978)

**ML Models**

```
[41] #Linear regression model
     from sklearn.linear_model import LinearRegression

     # create linear regression object
     linreg = LinearRegression()

     # train the model using the training sets
     linreg.fit(X_train, y_train_le)

     #prediction score
     print("Training score: {:.3f}".format(linreg.score(X_train,y_train_le)))
     print("Testing score: {:.3f}".format(linreg.score(X_test,y_test_le)))

     Training score: 0.299
     Testing score: 0.275
```

```
#Decision tree model
from sklearn.tree import DecisionTreeRegressor

# create a regressor object
tree = DecisionTreeRegressor(random_state = 0)

# fit the regressor with X and Y data
tree.fit(X_train, y_train_le)

#prediction score
print("Training score: {:.3f}".format(tree.score(X_train,y_train_le)))
print("Testing score: {:.3f}".format(tree.score(X_test,y_test_le)))
```

```
Training score: 1.000
Testing score: 0.940
```

```
#Support Vector Classifier model
from sklearn.svm import SVC

clf = SVC(kernel='linear')

# fitting x samples and y classes
clf.fit(X_train, y_train_le)

#prediction score
print("Training score: {:.3f}".format(clf.score(X_train,y_train_le)))
print("Testing score: {:.3f}".format(clf.score(X_test,y_test_le)))
```

```
Training score: 0.992
Testing score: 0.993
```

```python
#Random Forest Classifier model
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators = 10)

# Training the model on the training dataset
rf.fit(X_train, y_train_le)

# performing predictions on the test dataset
y_pred = rf.predict(X_test)

#prediction score
print("Training score: {:.3f}".format(rf.score(X_train,y_train_le)))
print("Testing score: {:.3f}".format(rf.score(X_test,y_test_le)))
```

```
Training score: 1.000
Testing score: 0.989
```

```python
#Gradient Booster classifier model
from sklearn.ensemble import GradientBoostingClassifier

#create the object and fit the model on the training dataset
gbc = GradientBoostingClassifier(n_estimators=10, learning_rate=1.0, max_depth=1, random_state=0).fit(X_train, y_train_le)

#prediction score
print("Training score: {:.3f}".format(gbc.score(X_train,y_train_le)))
print("Testing score: {:.3f}".format(gbc.score(X_test,y_test_le)))
```

```
Training score: 0.095
Testing score: 0.075
```

```python
#Logisitc Regression model
from sklearn.linear_model import LogisticRegression

#create the object and fit the model on the training dataset
logreg = LogisticRegression(random_state=0).fit(X_train, y_train_le)

#prediction score
print("Training score: {:.3f}".format(logreg.score(X_train,y_train_le)))
print("Testing score: {:.3f}".format(logreg.score(X_test,y_test_le)))
```

```
Training score: 0.968
Testing score: 0.961
```

```
#MLP classifier
from sklearn.neural_network import MLPClassifier

#scaling the data
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
X_train_scaled = scale.fit_transform(X_train)

#create object and fit the model on training dataset
mlp = MLPClassifier(random_state=1, max_iter=300).fit(X_train_scaled, y_train_le)

#prediction score
print("Training score: {:.3f}".format(mlp.score(X_train_scaled,y_train_le)))
print("Testing score: {:.3f}".format(mlp.score(X_test,y_test_le)))
```

```
Training score: 0.993
Testing score: 0.116
```

```
[ ] from sklearn.externals import joblib
    joblib.dump(rf, 'model.pkl')
```

```
[ ] ['model.pkl']
```

```
[ ] rf = joblib.load('model.pkl')
```

```
[ ] model_columns = list(X_train.columns)
    joblib.dump(model_columns, 'model_columns.pkl')
    ['model_columns.pkl']
```

```
[ ]  # Dependencies
     from flask import Flask, request, jsonify
     import traceback

     # Your API definition
     app = Flask(__name__)

     @app.route('/predict', methods=['POST'])
     def predict():
         if lr:
             try:
                 json_ = request.json
                 print(json_)
                 query = pd.get_dummies(pd.DataFrame(json_))
                 query = query.reindex(columns=model_columns, fill_value=0)

                 prediction = list(rf.predict(query))

                 return jsonify({'prediction': str(prediction)})

             except:

                 return jsonify({'trace': traceback.format_exc()})
         else:
             print ('Train the model first')
             return ('No model here to use')

     if __name__ == '__main__':
         try:
             port = int(sys.argv[1]) # This is for a command-line input
         except:
             port = 12345 # If you don't provide any port the port will be set to 12345
```

```
     lr = joblib.load("model.pkl") # Load "model.pkl"
     print ('Model loaded')
     model_columns = joblib.load("model_columns.pkl") # Load "model_columns.pkl"
     print ('Model columns loaded')

     app.run(port=port, debug=True)
```

## Website

## Code:

*<!DOCTYPE html>*

*<html>*

*<head>*

*<title>Login Form</title>*

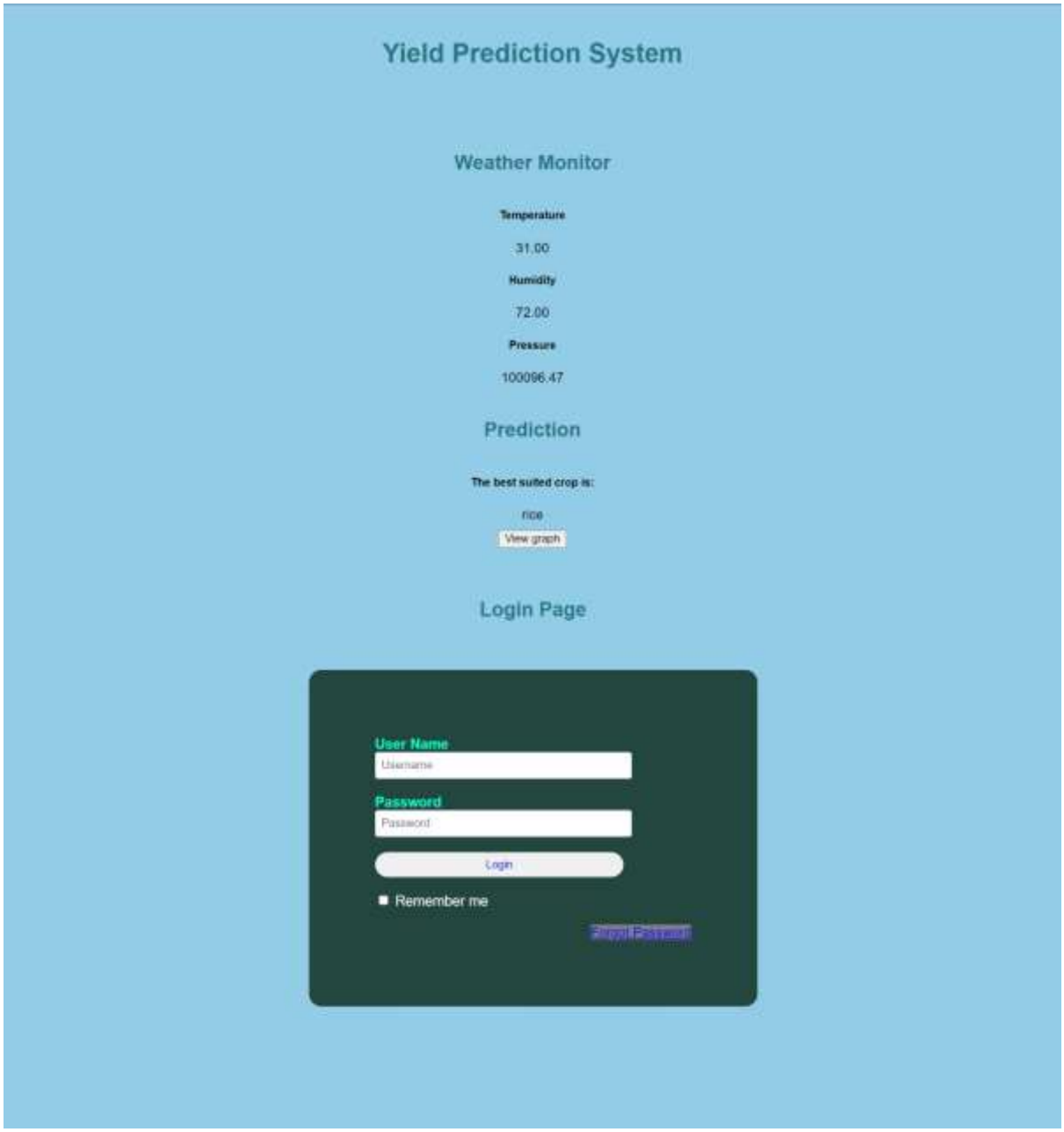
*<link rel="stylesheet" type="text/css" href="C:/Users/anegh/OneDrive/Desktop/tarp/style.css">*

```
</head>

<body>

   <h2>Login Page</h2><br>

   <div class="login">

   <form id="login" method="get" action="yield.html">

      <label><b>User Name</b>

      </label>

      <input type="text" name="Uname" id="Uname" placeholder="Username">

      <br><br>

      <label><b>Password</b>

      </label>

      <input type="Password" name="Pass" id="Pass" placeholder="Password">

      <br><br>

      <input type="submit" name="log" id="log" value="Login">

      <br><br>

      <input type="checkbox" id="check">

      <span>Remember me</span>

      <br><br>

      <a href="#">Forgot Password</a>

   </form>

</div>

</body>

</html>

<!DOCTYPE html>
<html>
<head>
   <title>Yield prediction</title>
   <link rel="stylesheet" type="text/css" href="C:/Users/anegh/OneDrive/Desktop/tarp/style.css">
</head>
<body>
   <h1>Yield Prediction System</h1><br>
   <h2> Weather Monitor </h2>
   <h5>Temperature</h5>
   <p>31.00</p>
   <h5>Humidity</h5>
   <p>72.00</p>
   <h5>Pressure</h5>
```

```
    <p>100096.47</p>
    <h2>Prediction</h2>
    <h5>The best suited crop is:</h5>
    <p>rice</p>
    <div class="btn">
        <button type="button">View graph</button>
    </div>

</body>
</html>
</body>
</html>
[21:35, 30/03/2023] Nimbu□: body
{
    margin: 0;
    padding: 0;
    background-color:#6abadeba;
    font-family: 'Arial';
}
.btn{
    margin: 0;
    position: absolute;
    top: 95%;
    left: 50%;
    -ms-transform: translate(-50%, -50%);
    transform: translate(-50%, -50%);
  }
.login{
        width: 382px;
        overflow: hidden;
        margin: auto;
        margin: 20 0 0 450px;
        padding: 80px;
        background: #23463f;
        border-radius: 15px ;

}
h5{
    text-align: center;
}
p{
    text-align: center;
}
h1{
    text-align: center;
    color: #277582;
    padding: 20px;
}
h2{
    text-align: center;
    color: #277582;
    padding: 20px;
}
label{
    color: #08ffd1;
    font-size: 17px;
}
#Uname{
    width: 300px;
    height: 30px;
```

```
    border: none;
    border-radius: 3px;
    padding-left: 8px;
}
#Pass{
    width: 300px;
    height: 30px;
    border: none;
    border-radius: 3px;
    padding-left: 8px;

}
#log{
    width: 300px;
    height: 30px;
    border: none;
    border-radius: 17px;
    padding-left: 7px;
    color: blue;

}
span{
    color: white;
    font-size: 17px;
}
a{
    float: right;
    background-color: grey;
}

import streamlit as st
import pickle
from tensorflow import keras
from keras.models import load_model
import numpy
import joblib
import os
path=os.getcwd()
path=os.path.join(path,'model2.pkl')
# with open(path, 'rb') as file:
#     model2 = pickle.load(file)
model2=joblib.load('model2.pkl')
print(type(model2))
#model=pickle.load(r"C:\Users\anegh\OneDrive\Desktop\proj\model2.pkl")
st.header("YIELD PREDICTION SYSTEM")
pressure=st.text_input("ENTER PRESSURE VALUE:")
temp=st.text_input("ENTER TEMPERATURE VALUE:")
humid=st.text_input("ENTER HUMID VALUE:")
if (pressure is not None) and (temp is not None) and (humid is not None):
    if st.button("predict"):
        inp = numpy.array([[float(pressure), float(temp), float(humid)]])
        out = model2.predict(inp)
        st.subheader(out[0])
```

## Output:




## Arduino UNO

## Code:

*// Include Libraries*

*#include "Arduino.h"*

*#include "SFE_BMP180.h"*

*#include "DHT.h"*

*// Pin Definitions*

```
#define DHT_PIN_DATA  2
#define SIM800L_SOFTWARESERIAL_PIN_TX        1
#define SIM800L_SOFTWARESERIAL_PIN_RX        0



// Global variables and defines

// object initialization
SFE_BMP180 bmp180;
DHT dht(DHT_PIN_DATA);



// define vars for testing menu
const int timeout = 60000;       //define timeout of 10 sec
char menuOption = 0;
long time0;

// Setup the essentials for your circuit to work. It runs first every time your circuit is powered with electricity.
void setup()
{
   // Setup Serial which is useful for debugging
   // Use the Serial Monitor to view printed messages
   Serial.begin(9600);
   while (!Serial) ; // wait for serial port to connect. Needed for native USB
   Serial.println("start");

   //Initialize I2C device
   bmp180.begin();
   dht.begin();
```

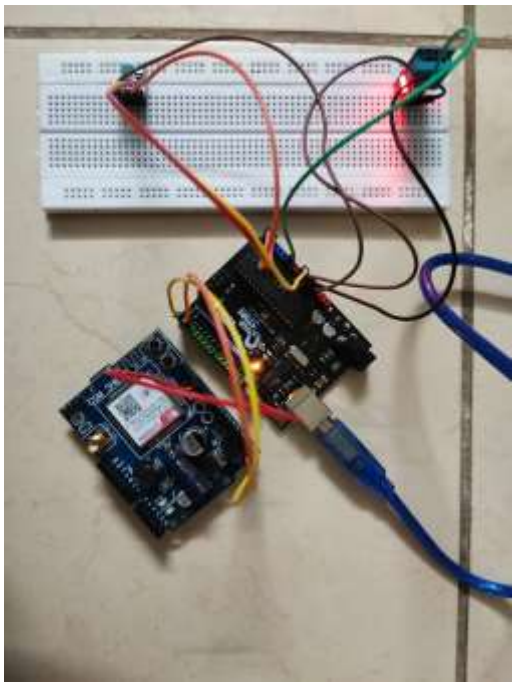
```
    menuOption = menu();


}


// Main logic of your circuit. It defines the interaction between the components you selected. After setup, it runs over and over again, in an eternal loop.

void loop()

{



    if(menuOption == '1') {

    // BMP180 - Barometric Pressure, Temperature, Altitude Sensor - Test Code

    // Read Altitude from barometric sensor, note that the sensor is 1m accurate

    double bmp180Alt = bmp180.altitude();

    double bmp180Pressure = bmp180.getPressure();

    double bmp180TempC = bmp180.getTemperatureC();     //See also bmp180.getTemperatureF() for Fahrenheit

    Serial.print(F("Altitude: ")); Serial.print(bmp180Alt,1); Serial.print(F(" [m]"));

    Serial.print(F("\tpressure: ")); Serial.print(bmp180Pressure,1); Serial.print(F(" [hPa]"));

    Serial.print(F("\tTemperature: ")); Serial.print(bmp180TempC,1); Serial.println(F(" [°C]"));


    }

    else if(menuOption == '2') {

    // DHT22/11 Humidity and Temperature Sensor - Test Code

    // Reading humidity in %

    float dhtHumidity = dht.readHumidity();

    // Read temperature in Celsius, for Fahrenheit use .readTempF()

    float dhtTempC = dht.readTempC();

    Serial.print(F("Humidity: ")); Serial.print(dhtHumidity); Serial.print(F(" [%]\t"));

    Serial.print(F("Temp: ")); Serial.print(dhtTempC); Serial.println(F(" [C]"));
```

```
    }
    // else if(menuOption == '3')
    // {
    // // Disclaimer: The QuadBand GPRS-GSM SIM800L is in testing and/or doesn't have code, therefore it may be buggy. Please be kind and report any bugs you may find.
    // }

    if (millis() - time0 > timeout)
    {
        menuOption = menu();
    }

}

// Menu function for selecting the components to be tested
// Follow serial monitor for instrcutions
char menu()
{

    Serial.println(F("\nWhich component would you like to test?"));
    Serial.println(F("(1) BMP180 - Barometric Pressure, Temperature, Altitude Sensor"));
    Serial.println(F("(2) DHT22/11 Humidity and Temperature Sensor"));
    Serial.println(F("(3) QuadBand GPRS-GSM SIM800L"));
    Serial.println(F("(menu) send anything else or press on board reset button\n"));
    while (!Serial.available());

    // Read data from serial monitor if received
    while (Serial.available())
```

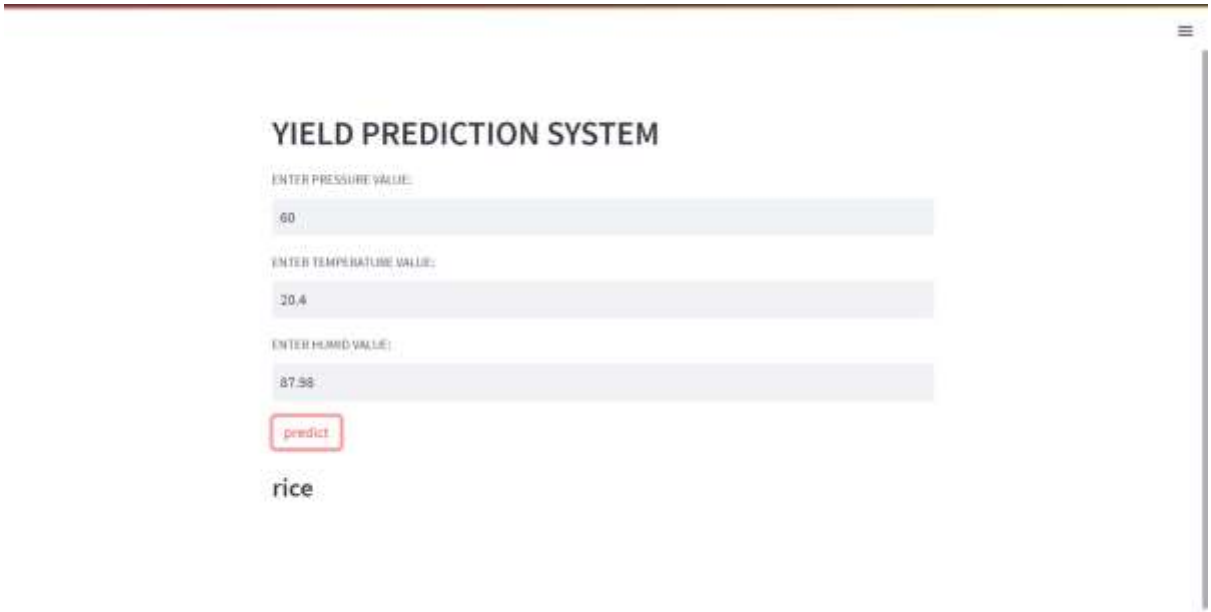
```
  {
    char c = Serial.read();
    if (isAlphaNumeric(c))
    {

      if(c == '1')
                    Serial.println(F("Now Testing BMP180 - Barometric Pressure, Temperature, Altitude Sensor"));
              else if(c == '2')
                    Serial.println(F("Now Testing DHT22/11 Humidity and Temperature Sensor"));
              else if(c == '3')
                    Serial.println(F("Now Testing QuadBand GPRS-GSM SIM800L - note that this component doesn't have a test code"));
      else
      {
        Serial.println(F("illegal input!"));
        return 0;
      }
      time0 = millis();
      return c;
    }
  }
}
```

**Deployment:**

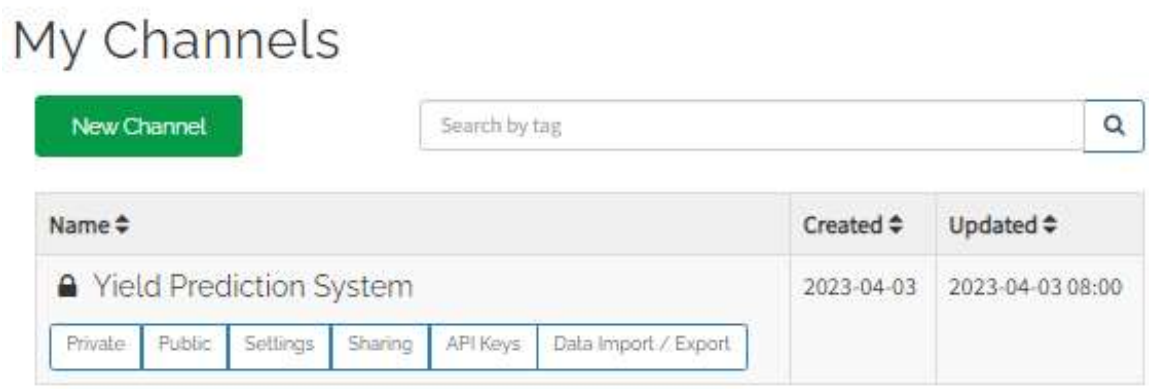
Connections:

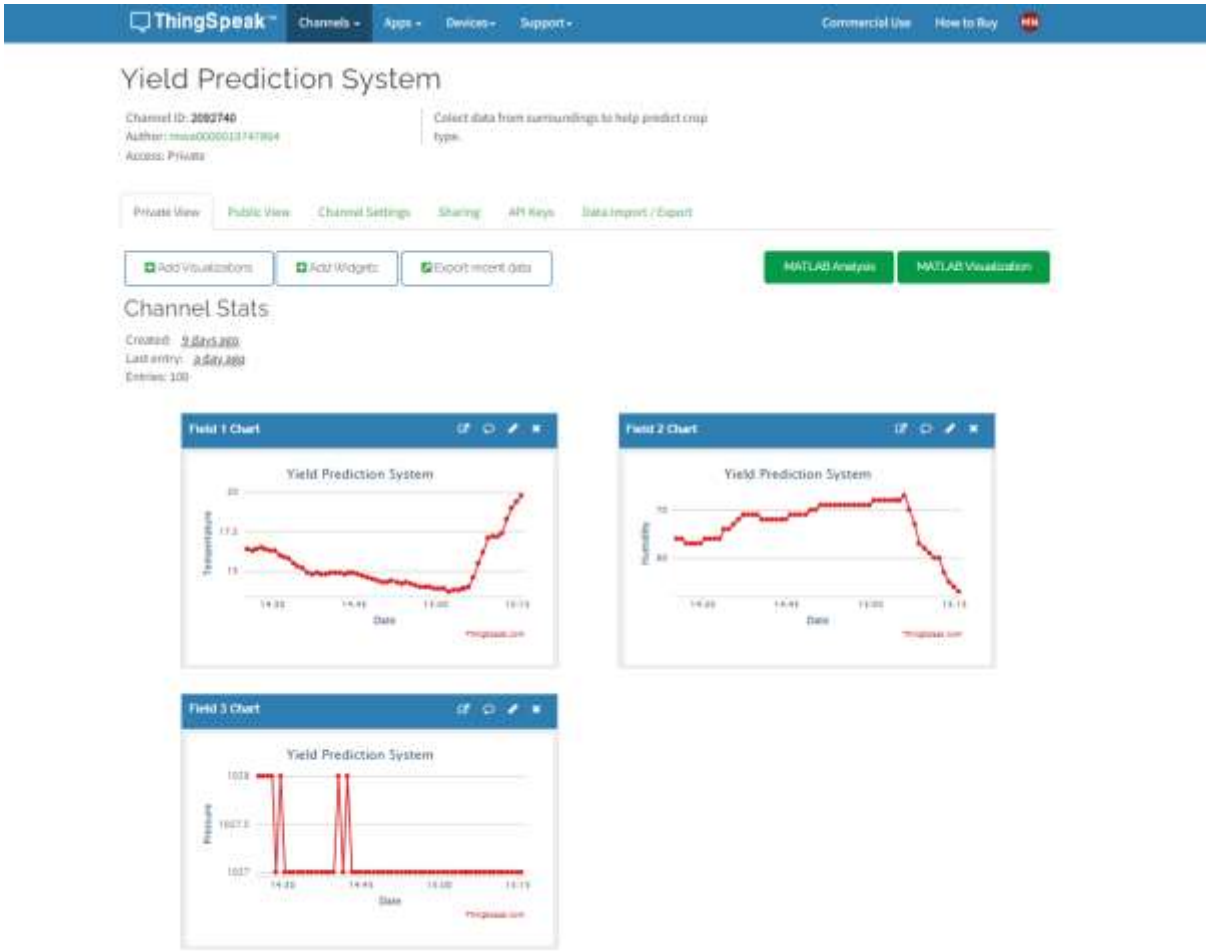


Website



IoT Cloud:




**Readme File:**

# Yield Prediction System

This project is a prototype of a Yield Prediction System developed as a part of a course taught in our school. This prototype is meant to be employed in the Agriculture Sector, specifically targeting farmers.

This project employs cloud technology to store data, Machine Learning and REST apis on a website to help farmers get the correct crop to grow. We use sensors and Arduino Uno to get real time data that is uploaded to cloud to later improve our ML model to help it give better and more accurate results.

## Authors

- [@megha70265](https://github.com/megha70265)
- [@AneghaJain](https://github.com/AneghaJain)
- [@aana0308](https://github.com/aana0308)
- [Shashank7000](https://github.com/Shashank7000)

## Acknowledgements

- [Arduino Documentation](https://docs.arduino.cc/hardware/uno-rev3)
- [Arduino Cloud Blog](https://blog.arduino.cc/2022/10/14/use-your-phone-as-an-iot-device-in-the-arduino-cloud/)
- [ThingSpeak](https://in.mathworks.com/help/thingspeak/getting-started-with-thingspeak.html)
- [Streamlit Web App Blog](https://medium.com/@u.praneel.nihar/streamlit-for-building-web-apps-9b8ab6b829fb)

## Packages

The following packages must be installed in your virtual environment for running the Streamlit web app:

- tensorflow

- streamlit

- numpy

- pandas

- pickle

- joblib

The following libraries to be installed in Arduino IDE:

- SoftwareSerial.h

- DHT.h

- SFE_BMP180.h

- Arduino.h

## Tech Stack

**Client:** HTML, CSS

**Server:** Streamlit, ThingSpeak

**ML Model:** RandomForestClassifer

## Deployment

To deploy this project download zip file.

Make a Thingspeak account and make a channel to deploy this project on cloud. Make the following appropriate fields on the channel:

- field1: Temperature

- field2: Humidity

- field3: Pressure

Warning: ThingSpeak assumes a created_by(timestamp for upload) field.

Open '.ino' file on Arduino IDE. Change Write API key to your own Write API key found on your ThingSpeak channel. To test connected devices run the '.ino' and manually check each device using the option menu.

Plug model.pickle file into the REST API and in your CLI type the following:

```bash
  python app.py
```

Website will open in active web browser.

## Report

[Access Report](https://drive.google.com/file/d/1F9Y1nY4nf8aHOwjQXIzeL5H8GYLGPEKR/view?usp=sharing)