FARM TO FORUM

Introduction to Farm to Forum:

In a world where the agricultural industry is often plagued by inefficiencies and complex supply chains, Farm to Forum emerges as a revolutionary solution aimed at transforming the way farmers engage with the marketplace. This innovative platform harnesses the power of artificial intelligence to empower rice farmers, helping them enhance their profitability while ensuring that consumers receive high-quality products at reasonable prices. By directly connecting farmers to consumers, Farm to Forum eliminates the need for intermediaries—those middlemen who historically increase costs and complicate transactions.

At its core, Farm to Forum focuses on making the farming experience more transparent and rewarding. It employs advanced technologies to analyze rice samples. This analysis allows our sophisticated AI algorithms to assess quality attributes such as grain size, shape, and color, ultimately determining a fair market price that reflects current demand and quality. By providing farmers with immediate insights into the value of their crops, we aim to foster confidence and facilitate informed decision-making.

The impact of Farm to Forum extends beyond just economic transactions; it nurtures a sense of community and collaboration. Our vision is to create a vibrant marketplace that goes beyond mere sales—a place where education, sustainability, and local economies thrive.

Description:

The implementation of the Farm to Forum project is designed to create an intuitive and robust platform that connects rice farmers directly to consumers, leveraging technology to streamline operations and enhance the overall user experience. The architecture is built around core functionalities that facilitate image analysis of rice samples, pricing predictions, and marketplace interactions.

1. Core Components:

key components:

1.Image Processing with Computer Vision:

Utilizing the OpenCV library, the application enables farmers to upload images of their rice samples. The backend processes these images to extract relevant quality metrics, such as grain size, shape, and color. This analysis is pivotal in ensuring that pricing is fair and reflective of the product's quality.

2.AI Pricing Model:

The project employs machine learning techniques, facilitated by libraries like Scikit-learn, to create a predictive model that determines optimal pricing based on the analyzed attributes of the rice samples alongside historical pricing data. This model continually learns and adapts, ensuring accuracy as market trends evolve.

2. Workflow:

The implementation flows as follows:

1.Uploading Rice Samples:

Farmers start by uploading images of their rice. The system immediately processes these images, performing quality assessments using pre-defined metrics.

2.Pricing Suggestions:

Once the analysis is complete, the AI model calculates a suggested selling price, which is communicated back to the farmer. This transparency in pricing builds trust and ensures farmers feel confident in the value of their products.

3.How The Code Works:

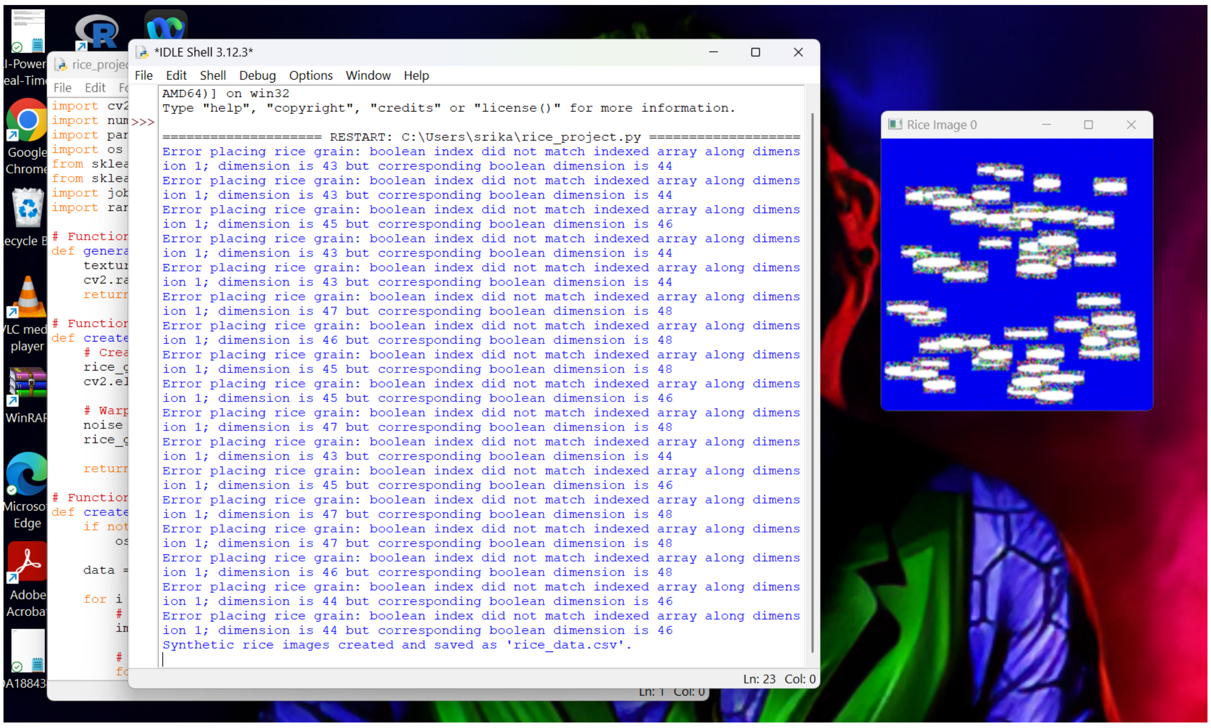
The code generates random images of rice and gets trained from the random data set. Later based on it another random image is given for which it produces the quality based on pre-exsisting set. We could also provide actual rice samples and test cases which would could deploy in real time.

Code:

```
import cv2
import numpy as np
import pandas as pd
import os
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import joblib
import random

# Function to generate a texture for the background
def generate_background_texture(size):
   texture = np.ones(size, dtype=np.uint8) * 230  # Light gray background
   cv2.randn(texture, 240, 10)  # Add some Gaussian noise
   return texture

# Function to create more complex rice grain shapes
def create_rice_grain(center, axes):
   # Create an empty mask with a transparent background
   rice_grain = np.zeros((axes[1] * 2, axes[0] * 2, 3), dtype=np.uint8)
   cv2.ellipse(rice_grain, (axes[0], axes[1]), (axes[0], axes[1] // 2), 0, 0, 360,
(255, 255, 255), -1)

   # Warp the shape a bit to make it less symmetrical
   noise = np.random.uniform(-5, 5, rice_grain.shape).astype(np.uint8)
   rice_grain = cv2.add(rice_grain, noise)

   return rice_grain

# Function to create synthetic rice-like images
def create_synthetic_rice_images(num_images=100, image_size=(300, 300)):
   if not os.path.exists('rice_images'):
      os.makedirs('rice_images')

   data = []

   for i in range(num_images):
    # Generate a background texture
    image = generate_background_texture((image_size[0], image_size[1], 3))

    # Add rice grain-like shapes
    for _ in range(random.randint(30, 70)):  # Random number of grains
        # Random position and size for the grains
        center = (np.random.randint(20, image_size[1] - 20),
np.random.randint(20, image_size[0] - 20))
        axes = (np.random.randint(15, 25), np.random.randint(7, 12))  # Wider
ellipses

        # Generate the rice grain
        rice_grain = create_rice_grain(center, axes)

        # Calculate the position for placing the grain
        x_offset = center[0] - axes[0]
        y_offset = center[1] - axes[1] // 2

        # Ensure that we do not go out of the image bounds
        if (0 <= x_offset < image_size[1]) and (0 <= y_offset < image_size[0]):
            # Get the region of interest in the original image
            try:
                roi = image[y_offset:y_offset + rice_grain.shape[0], x_offset:x_offset
+ rice_grain.shape[1]]
                # Create a mask from the rice grain
                mask = rice_grain != [0, 0, 0]

                # Overlay the rice grain onto the original image only where it is not
black
                roi[mask] = rice_grain[mask]
            except Exception as e:
                print(f"Error placing rice grain: {e}")

    # Apply a slight blur for realism
    image = cv2.GaussianBlur(image, (3, 3), 0)

    # Generate a filename
```

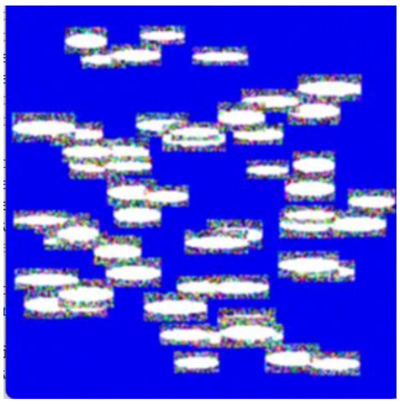
```
        filename = f'rice_images/rice_image_{i}.png'

        # Save the image
        cv2.imwrite(filename, image)

        # Assign a random price, for example, prices between $1 and $10
        price = np.random.uniform(1.0, 10.0)

        # Append to the list
        data.append({'image_path': filename, 'price': price})

    # Create a DataFrame and save it as a CSV file
    df = pd.DataFrame(data)
    df.to_csv('rice_data.csv', index=False)
    print("Synthetic rice images created and saved as 'rice_data.csv'.")

# Step 2: Extract Features from Images
def extract_features(image_path):
    image = cv2.imread(image_path)
    if image is None:
        return None
    image = cv2.resize(image, (300, 300))  # Match the size used in image
creation
    mean_color = image.mean(axis=(0, 1))
    histogram = cv2.calcHist([image], [0, 1, 2], None, [8, 8, 8], [0, 256, 0, 256, 0,
256])
    histogram = cv2.normalize(histogram, histogram).flatten()
    features = np.hstack((mean_color, histogram))
    return features

# Step 3: Display Images
def display_images(num_images=10):
    for i in range(num_images):
        image_path = f'rice_images/rice_image_{i}.png'
        image = cv2.imread(image_path)
        if image is not None:
            cv2.imshow(f'Rice Image {i}', image)
        cv2.waitKey(0)  # Wait for a key press to close the image
        cv2.destroyAllWindows()  # Close the window after key press

# Step 4: Train a Model
def train_model():
    # Load the synthetic dataset
    data = pd.read_csv('rice_data.csv')

    features = []
    prices = []

    # Extract features and prices from the dataset
    for index, row in data.iterrows():
        image_path = row['image_path']
        price = row['price']
        feature = extract_features(image_path)
        if feature is not None:
            features.append(feature)
            prices.append(price)

    X = np.array(features)
    y = np.array(prices)

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

    # Train a model
    model = LinearRegression()
    model.fit(X_train, y_train)

    # Save the model
    joblib.dump(model, 'rice_price_model.pkl')
    print("Model trained and saved as 'rice_price_model.pkl'.")

# Step 5: Predict Price for a New Image
def predict_rice_price(image_path):
    model = joblib.load('rice_price_model.pkl')
    features = extract_features(image_path)
    if features is None:
        return None
    features = features.reshape(1, -1)  # Reshape for the model
    predicted_price = model.predict(features)
    return predicted_price[0]

if __name__ == "__main__":
    # Create synthetic rice-like images
    create_synthetic_rice_images(num_images=100, image_size=(300, 300))

    # Display some images to visualize
    display_images(num_images=10)  # Adjust the number if needed

    # Train the model
    train_model()

    # Predict price for a sample image
    image_path = 'rice_images/rice_image_0.png'  # Change this if necessary
    price = predict_rice_price(image_path)
    print("Predicted Rice Price: $", price)
```

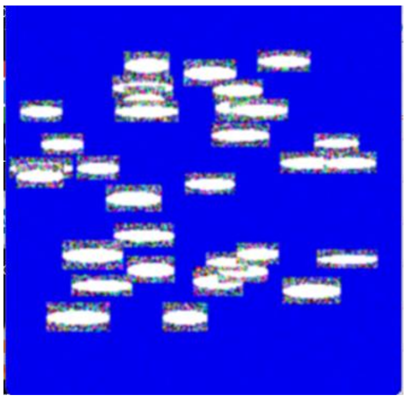
Output:

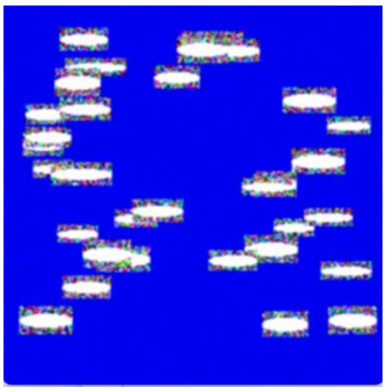


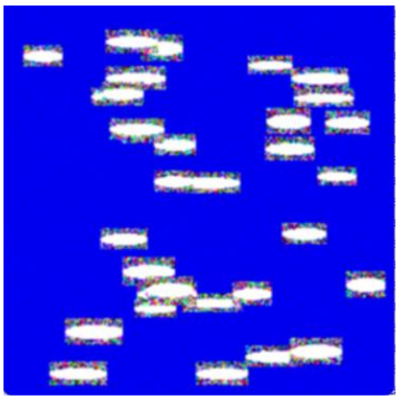
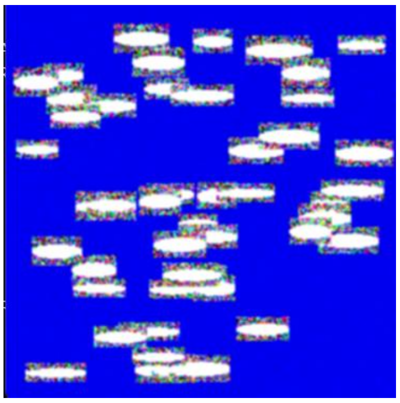
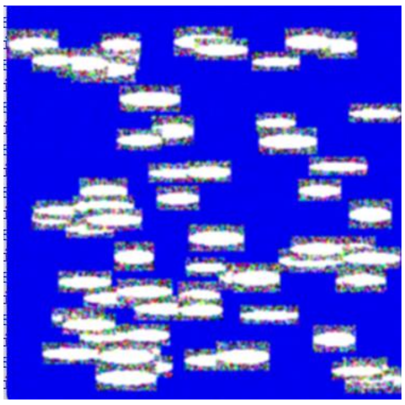
```
import cv2
import num
import par
import os
from sklea
from sklea
import job
import rar

# Function
def genera
    textur
    cv2.ra
    return

# Function
def create
    # Crea
    rice_g
    cv2.el

    # Warp
    noise
    rice_g

    return

# Function
def create
    if not
        os

    data =

    for i
        #
        in

        #
        fo
```

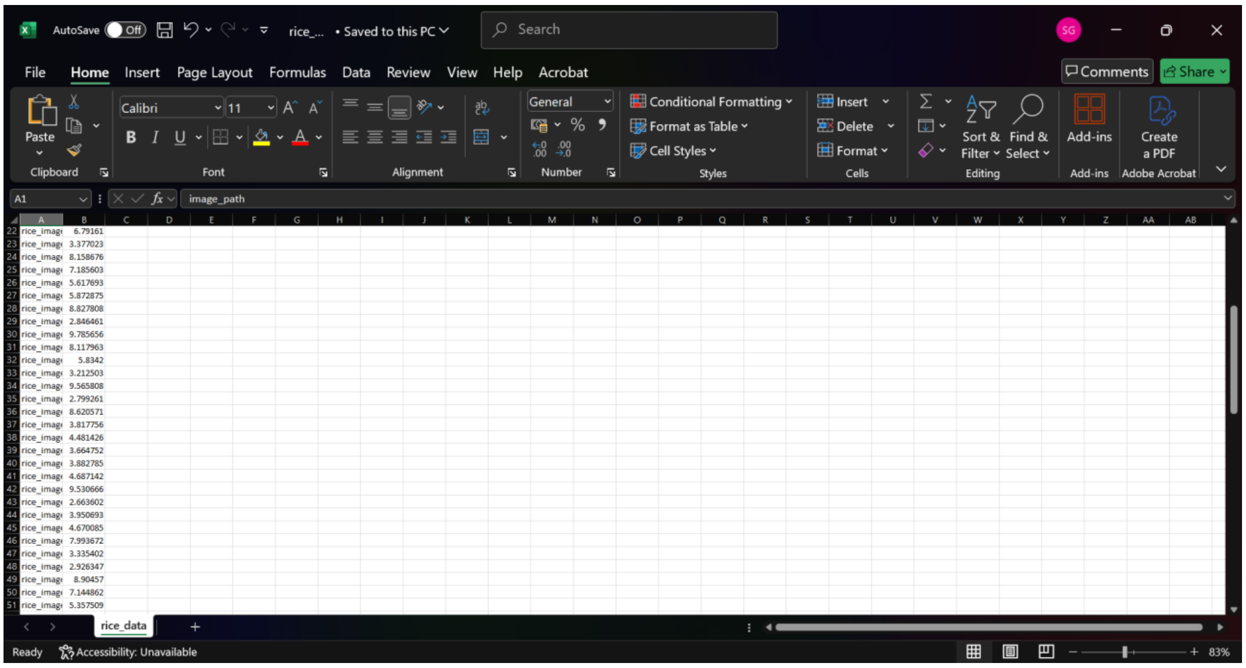
```
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:\Users\srika\rice_project.py ===================
Error placing rice grain: boolean index did not match indexed array along dimens
ion 1; dimension is 43 but corresponding boolean dimension is 44
Error placing rice grain: boolean index did not match indexed array along dimens
ion 1; dimension is 43 but corresponding boolean dimension is 44
Error placing rice grain: boolean index did not match indexed array along dimens
ion 1; dimension is 45 but corresponding boolean dimension is 46
Error placing rice grain: boolean index did not match indexed array along dimens
ion 1; dimension is 43 but corresponding boolean dimension is 44
Error placing rice grain: boolean index did not match indexed array along dimens
ion 1; dimension is 43 but corresponding boolean dimension is 44
Error placing rice grain: boolean index did not match indexed array along dimens
ion 1; dimension is 47 but corresponding boolean dimension is 48
Error placing rice grain: boolean index did not match indexed array along dimens
ion 1; dimension is 46 but corresponding boolean dimension is 48
Error placing rice grain: boolean index did not match indexed array along dimens
ion 1; dimension is 45 but corresponding boolean dimension is 48
Error placing rice grain: boolean index did not match indexed array along dimens
ion 1; dimension is 45 but corresponding boolean dimension is 46
Error placing rice grain: boolean index did not match indexed array along dimens
ion 1; dimension is 47 but corresponding boolean dimension is 48
Error placing rice grain: boolean index did not match indexed array along dimens
ion 1; dimension is 43 but corresponding boolean dimension is 44
Error placing rice grain: boolean index did not match indexed array along dimens
ion 1; dimension is 45 but corresponding boolean dimension is 46
Error placing rice grain: boolean index did not match indexed array along dimens
ion 1; dimension is 47 but corresponding boolean dimension is 48
Error placing rice grain: boolean index did not match indexed array along dimens
ion 1; dimension is 47 but corresponding boolean dimension is 48
Error placing rice grain: boolean index did not match indexed array along dimens
ion 1; dimension is 46 but corresponding boolean dimension is 48
Error placing rice grain: boolean index did not match indexed array along dimens
ion 1; dimension is 44 but corresponding boolean dimension is 46
Error placing rice grain: boolean index did not match indexed array along dimens
ion 1; dimension is 44 but corresponding boolean dimension is 46
Synthetic rice images created and saved as 'rice_data.csv'.
```









A1 image_path

	A	B
22	rice_image	6.79161
23	rice_image	3.377023
24	rice_image	8.158676
25	rice_image	7.185603
26	rice_image	5.617693
27	rice_image	5.872875
28	rice_image	8.827808
29	rice_image	2.846461
30	rice_image	9.785656
31	rice_image	8.117963
32	rice_image	5.8342
33	rice_image	3.212503
34	rice_image	9.565808
35	rice_image	2.799261
36	rice_image	8.620571
37	rice_image	3.817756
38	rice_image	4.481426
39	rice_image	3.664752
40	rice_image	3.882785
41	rice_image	4.687142
42	rice_image	9.530666
43	rice_image	2.663602
44	rice_image	3.950693
45	rice_image	4.670085
46	rice_image	7.993672
47	rice_image	3.335402
48	rice_image	2.926347
49	rice_image	8.90457
50	rice_image	7.144862
51	rice_image	5.357509

rice_data

```
Synthetic rice images created and saved as 'rice_data.csv'.
Model trained and saved as 'rice_price_model.pkl'.
Predicted Rice Price: $ 4.723450104332642
```

Conclusion:

In summary, the Farm to Form is highly useful for farmer where a major of profit is lost through middle while reaching the customer. Hence pre-determining the rice quality and assigning a price will and creating direct connection through the customer and farmer leads to less amount of burden on farmer and customer can purchase quality rice also through this project.