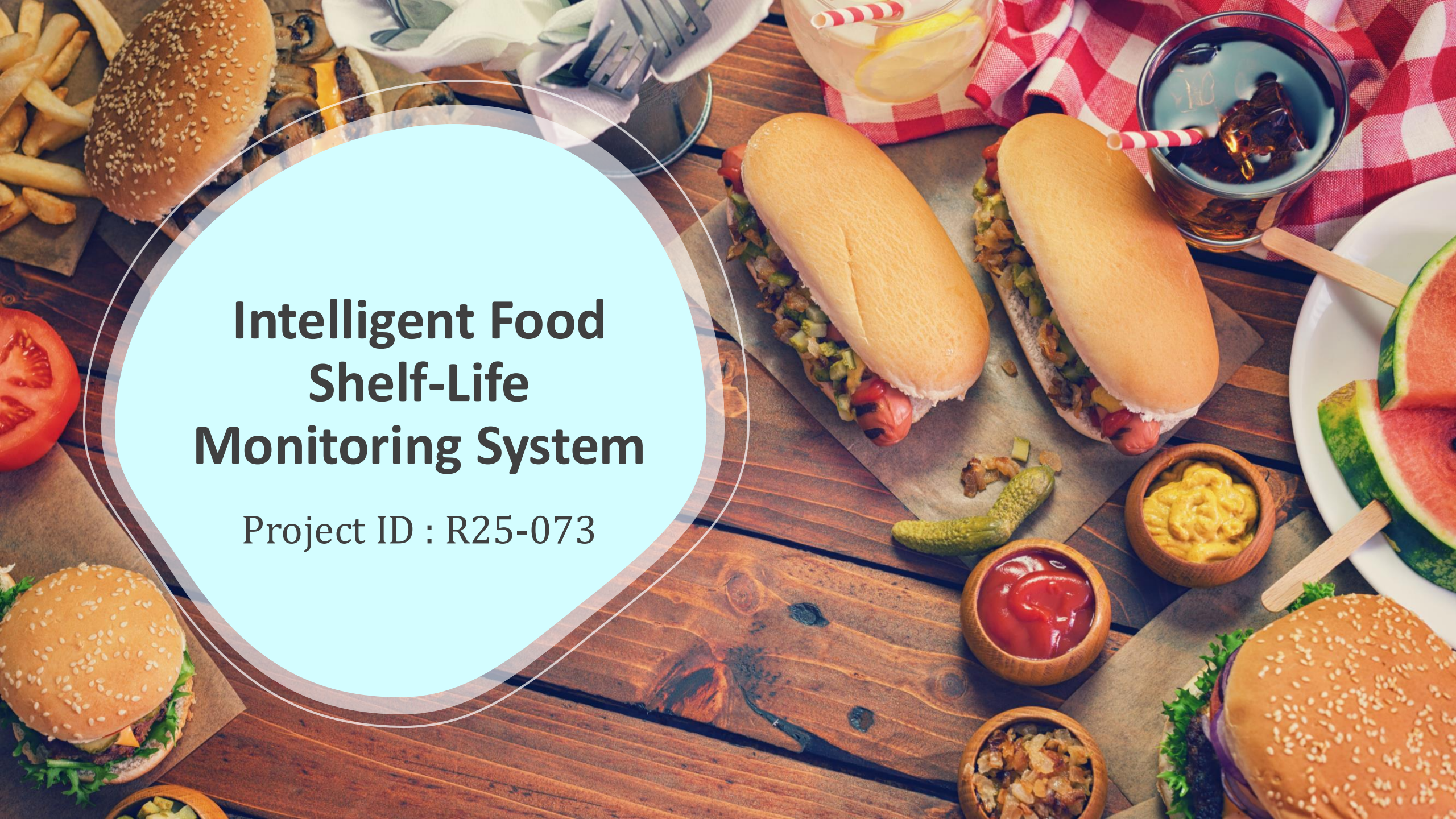
Intelligent Food Shelf-Life Monitoring System

Project ID : R25-073



Our Team

PERERA B.C.V
IT20196110

KULARATHNA E.M.S.D
IT21828720

FERNANDO M.G.S.S.A
IT19970882

PERERA P.A.E.U.K.
IT21829710

Introduction

This is a mobile application that is used to identify the expirations of the food that are stored in the fridge in home.

There are four main function

- Real time food identification and expiry detection.
- Estimating remaining shelf life of food items
- Managing food storage compatibility
- Suggest a recipe from expiring food

RESEARCH QUESTION

Food waste and spoilage continue to be significant challenges due to
- improper storage,
- lack of real-time freshness monitoring,
- the absence of personalized food management systems.

Objectives

Main Objective

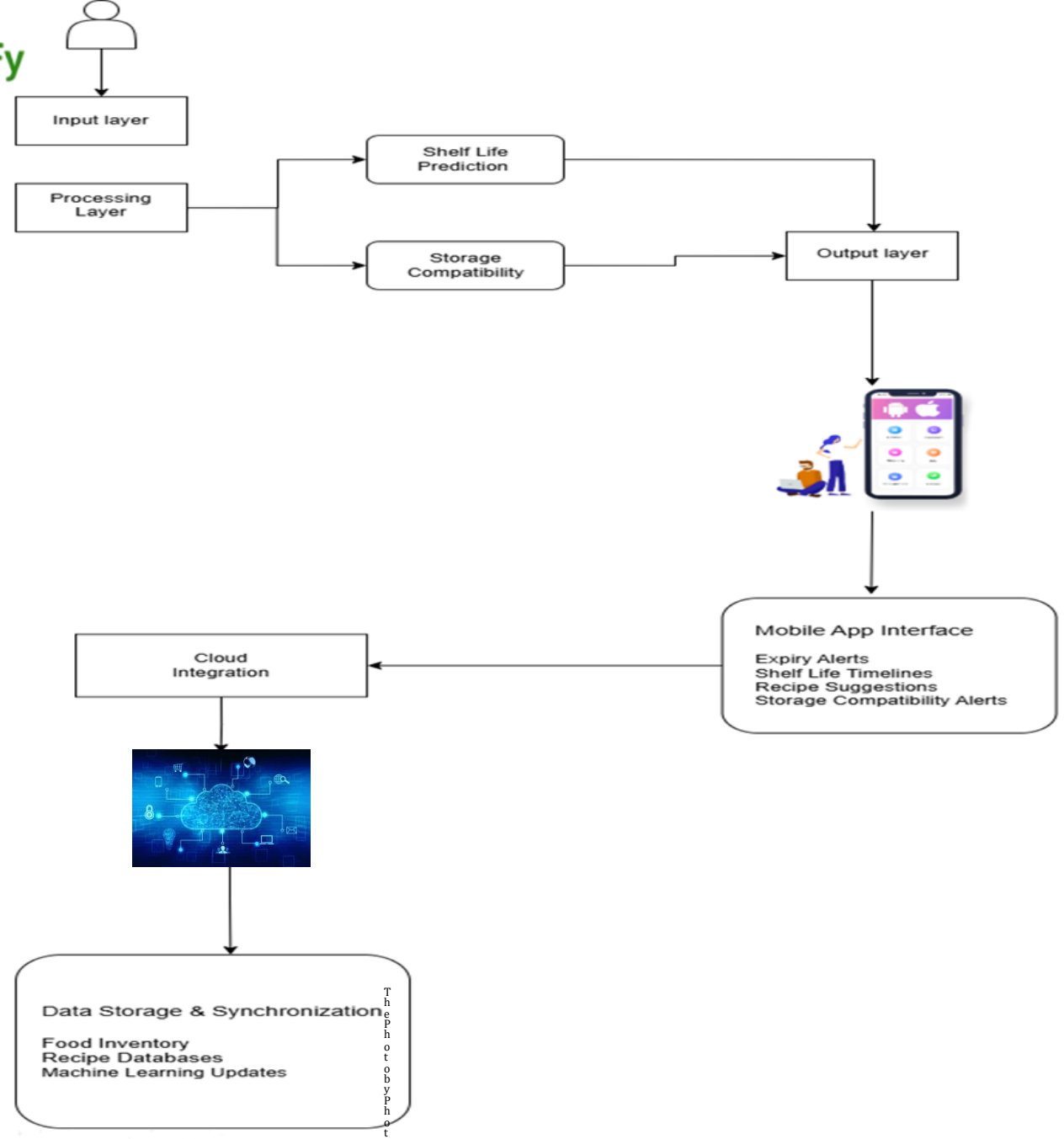
▪To develop an Intelligent Food Shelf-Life Monitoring System that leverages AI, IoT, and machine learning to reduce food waste, enhance storage compatibility, and improve household food management efficiency.

Sub Objective

- Develop a real-time food identification and spoilage detection system using image recognition and gas sensors.
- Implement a shelf-life prediction module to estimate remaining freshness for packaged, fresh, and homemade food items.

FreshiFy

SYSTEM DIAGRAM



The Photo by PhotoAuthor

IT20196110 – PERERA B.C.V

BSc (Hons) Degree in Information Technology
specialization in Information Technology

Real-Time Food Identification and Expiry Detection

INTRODUCTION

- HELPS USERS IDENTIFY AND TRACK FOOD TYPES TO MANAGE THEIR MEALS MORE EFFECTIVELY.
- ELIMINATES MANUAL SPOILAGE CHECKS BY AUTOMATING THE IDENTIFICATION AND NOTIFICATION PROCESS.
- DETECTS GAS USING SENSORS TO DETERMINE IF FOOD IS SPOILED, HELPING USERS MAKE SAFE DECISIONS.

Research Question

- **"How can a combined system of gas sensors and image processing provide reliable, real-time detection of homemade food spoilage to reduce manual check time and health risks?"**

Specific & Sub Objectives

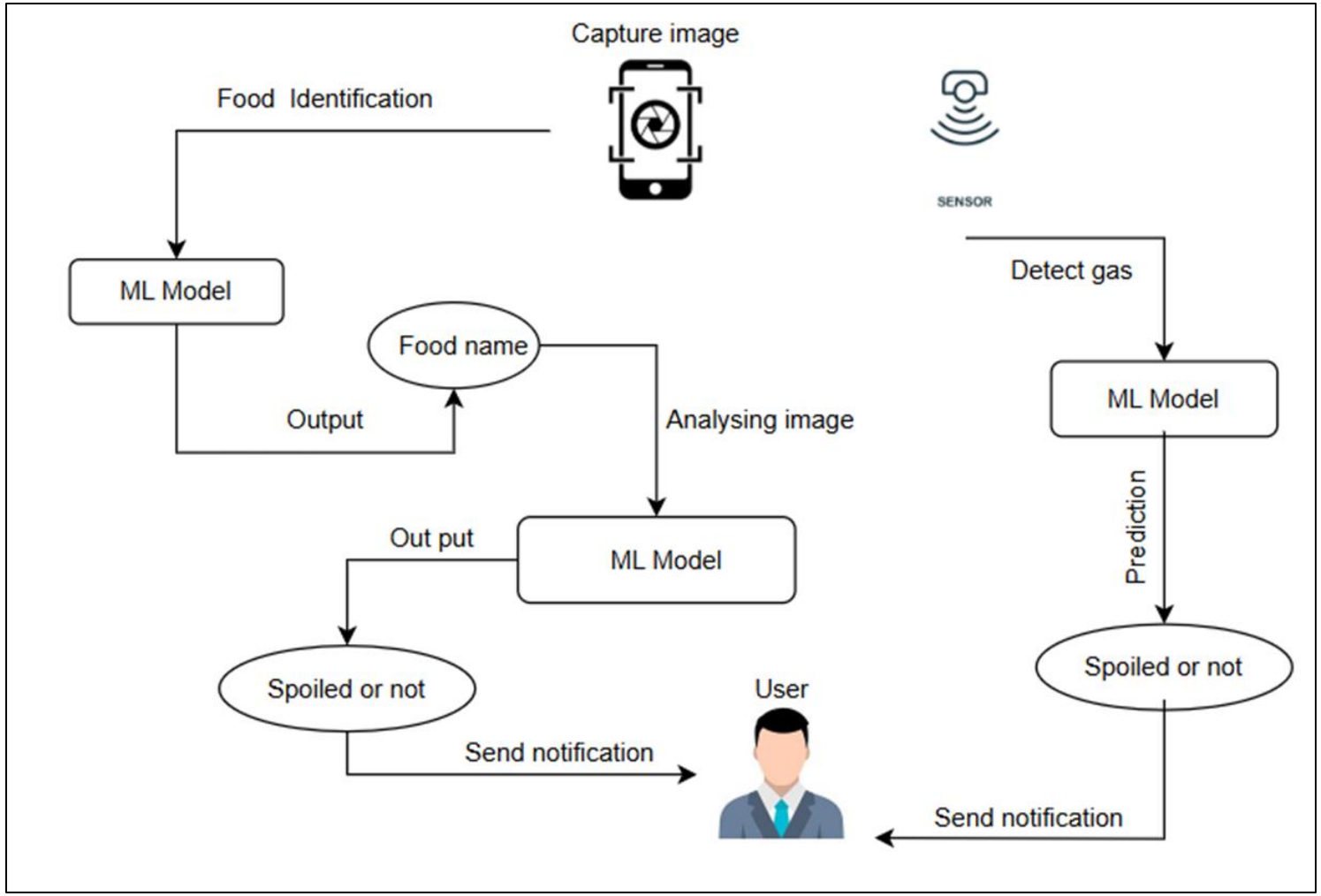
Specific

To develop a smart system that combines gas sensors and image processing to detect spoilage in homemade food accurately and in real-time

Sub

- To collect and preprocess datasets of spoiled and fresh homemade foods
- Identify and classify different food types.
- Analyze and detect the spoilage of homemade foods.
- Implement a notification system to inform users of spoiled food immediately.
- Enhance system reliability for accurate spoilage detection and timely alerts.

System Diagram



Tools & Technologies

- **Programming Language :** Python, Kotlin
- **Libraries** : TensorFlow, Keras, NumPy, Matplotlib, PIL, Tkinter
- **Model** : MobileNetV2
- **Frontend :** React Native
- **Sensors** : MQ-135

Requirements

Functional Requirements

- Image Capture and Analysis
- Sensor Integration
- Spoilage Detection via Machine Learning
- Mobile Application Interface

Non-Functional Requirements

- Performance
- Scalability
- Reliability
- Usability
- Maintainability

Coding Screenshot

Image Selection

```
# -------------------------------
# 1. Data Loading & Image Selection
# -------------------------------
data_root = 'fruits'
image_exts = ['jpeg', 'jpg', 'bmp', 'png']
max_images_per_class = 1000  # Limit for each class
```

Define Classes

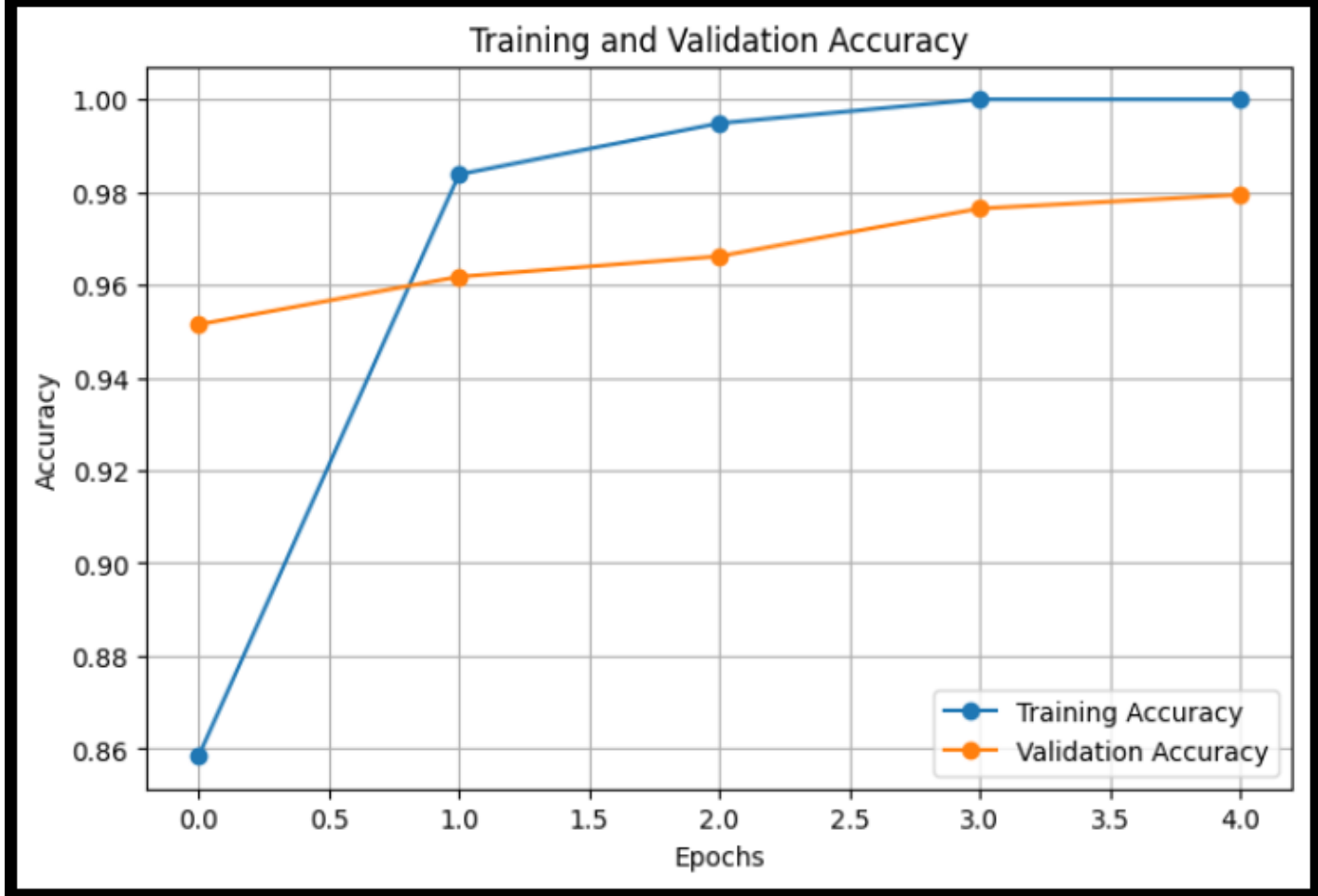
```
# -------------------------------
# 2. Combine Images & Create Labels
# -------------------------------
all_image_paths = []
all_labels = []

# Define the desired order of classes
desired_order = [
    'apple_fresh', 'apple_spoiled',
    'banana_fresh', 'banana_spoiled',
    'carrot_fresh', 'carrot_spoiled',
    'orange_fresh', 'orange_spoiled',
    'tomato_fresh', 'tomato_spoiled'
]
```

Model Build & Loading

```
# -------------------------------
# 5. Build & Train the Model
# -------------------------------
# Load MobileNetV2 as base model
base_model = tf.keras.applications.MobileNetV2(
    input_shape=(224, 224, 3),
    include_top=False,
    weights='imagenet',
    pooling='avg'
)
```

Coding Screenshot

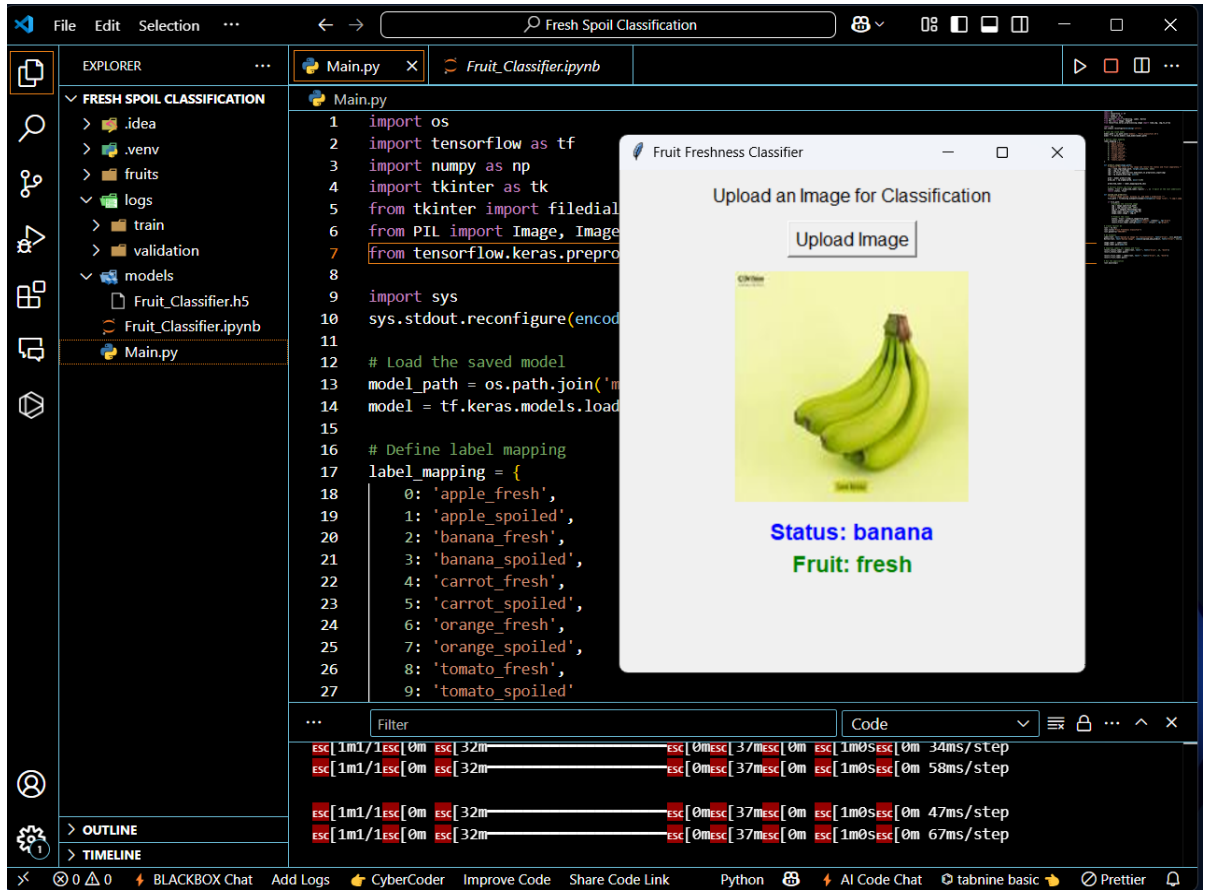
```
Model Accuracy on Validation Set: 0.9765
```

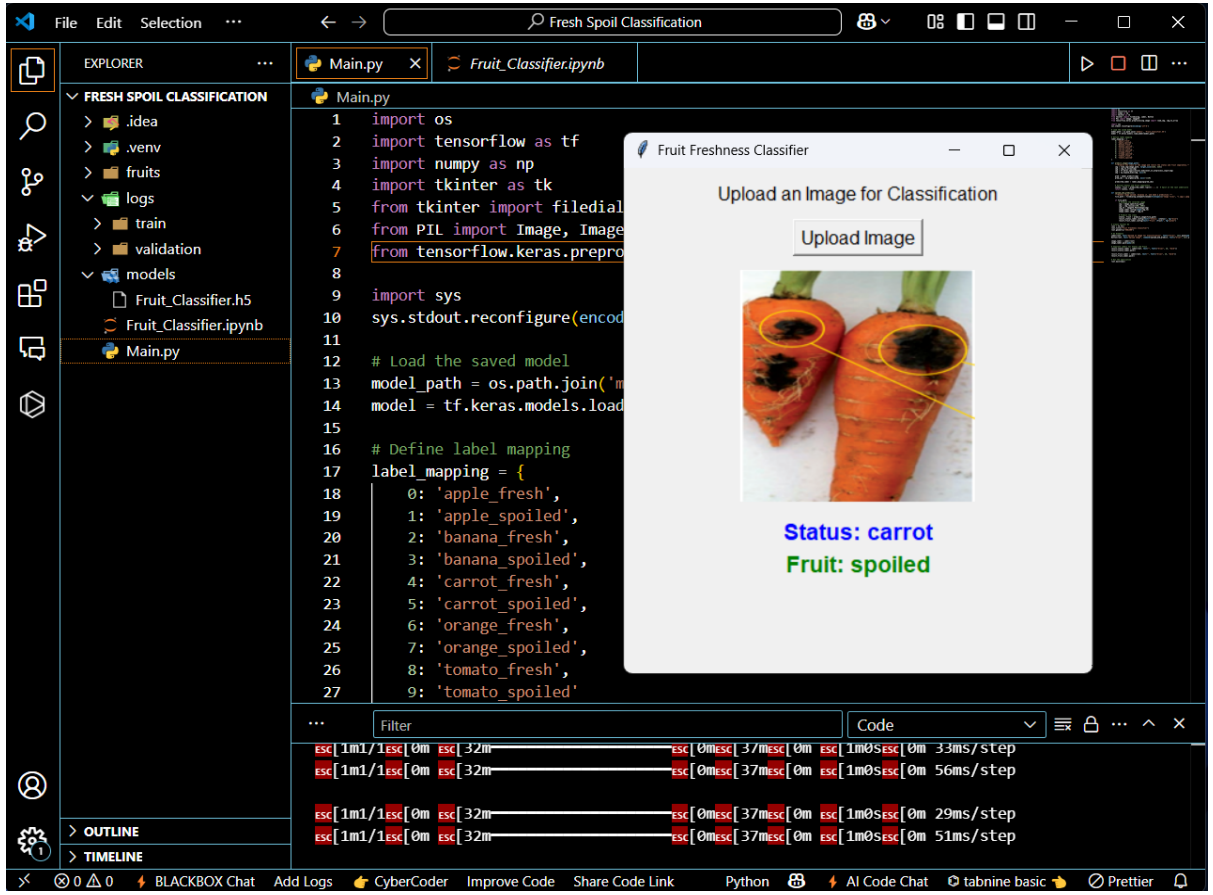



```
# 80% for training, 20% for validation
split_index = int(len(all_image_paths) * 0.8)
train_paths = all_image_paths[:split_index]
train_labels = all_labels[:split_index]
val_paths = all_image_paths[split_index:]
val_labels = all_labels[split_index:]
```

```
# ------------------------------
# 3. Manual Train/Validation Split (80/20)
# ------------------------------
# Combine paths and labels into a list of tuples and shuffle
combined = list(zip(all_image_paths, all_labels))
random.shuffle(combined)
all_image_paths, all_labels = zip(*combined)
all_image_paths = list(all_image_paths)
all_labels = list(all_labels)
```

Coding Screenshot





Completion and Future works

- **Completion of the components**
  - Dataset Collection & Preparation
  - Model Training
  - Temporary User Interface (UI)

- **Future Implementations**
  - Sensor Integration
  - UI/UX Improvement
  - Expand Dataset to Homemade Food

FreshiFy

IT21828720 | KULARATHNA E.M.S.D

SPECIALIZING IN INFORMATION TECHNOLOGY

COMPONENT -2

Estimating Remaining Shelf Life of Food Items

INTRODUCTION BACKGROUND

- Reducing food waste is essential for health and sustainability.
- Packaged foods often spoil due to poor freshness tracking.
- My task: Train a model that predicts how fresh a food item is, based on images.
- This can be later integrated into a mobile app.

"How can I build a smart system to predict food freshness and extract expiration dates from package labels to help users reduce food waste through timely alerts?"

Research Question

Specific & Sub Objectives

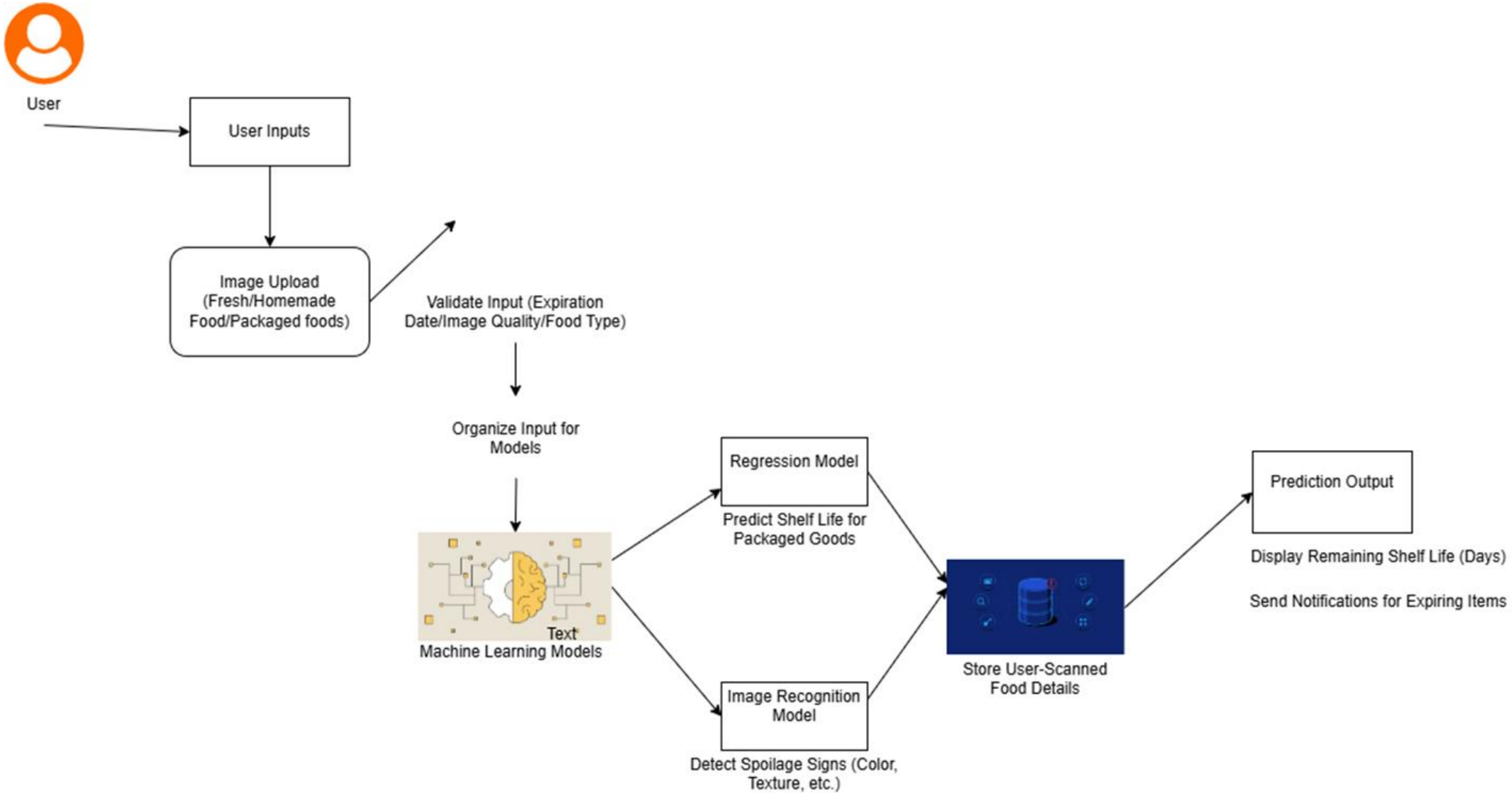
Main Objective

Train an ML model to estimate freshness stage and remaining shelf life.

Sub objectives

- **Create dataset of food images with freshness labels.**
- **Train and test a model to classify freshness levels.**
- **Extract and store expiration dates from food labels using image recognition (OCR).**
- **Generate alerts before expiration.**

System Diagram



Tools & Technologies

- **Programming Language** – Python, Java (for Android), Kotlin (UI), JavaScript
- **Libraries** – TensorFlow, Keras, NumPy, Matplotlib, PIL, Tkinter
- **Database** – Firebase
- **Models** – Custom CNN Model (MobileNetV2)
- **API** – Flask (for serving predictions)
- **Frontend** – Android Studio (XML UI), React Native (planned)

System, Personal and Software Specification Requirement

Functional Requirements

- Accept image input from camera/gallery.
- Send image to backend API for freshness prediction.
- Return freshness
- Store expiration prediction locally or in cloud database.
- (Future) Alert user before predicted expiry date.

Non - Functional Requirements

- User-friendliness
- Accessibility
- Usability
- Reliability
- Scalability

Personal Requirements

- Basic knowledge of Python (for model development).
- Familiarity with TensorFlow/Keras for CNN modeling.
- Android Studio usage experience.
- Ability to handle datasets and do data preprocessing.
- Team coordination for integration with UI/backend.

Software Requirement

- Python 3.10+
- TensorFlow 2.0+
- Flask, Pillow, Tesseract
- Android Studio + SDK
- Windows 10

Coding Screenshot

Main_All.py 1 ● | Fruit_Classifier_All.ipynb ×

Fruit_Classifier_All.ipynb > # ----------

+ Code + Markdown | Run All ···

Image loading

```
# ------------------------------
# 1. Data Loading & Image Selection
# ------------------------------
data_root = 'fruits'
image_exts = ['jpeg', 'jpg', 'bmp', 'png']
max_images_per_class = 1000  # Limit for each class
```

[5]

+ Code + Markdown

```
# Dictionary to store selected image paths for each class
selected_images = {}
```

[6]

```
# Loop through each main folder (e.g., Apple, Banana, etc.) inside "Fruits"
for fruit in os.listdir(data_root):
    fruit_dir = os.path.join(data_root, fruit)
    if not os.path.isdir(fruit_dir):
        continue  # Skip if not a directory

    # Get valid class names inside the fruit folder using the desired prefixes
    valid_prefixes = ("_fresh", "_moderately_fresh", "_near_spoilage", "_spoiled")
    valid_classes = [d for d in os.listdir(fruit_dir) if d.endswith(valid_prefixes)]
    print(f"\nProcessing fruit: {fruit}")
    print(f"Valid classes: {valid_classes}")
```

[7]

Build model

```
# ------------------------------
# 5. Build & Train the Model
# ------------------------------
# Load MobileNetV2 as base model
base_model = tf.keras.applications.MobileNetV2(
    input_shape=(224, 224, 3),
    include_top=False,
    weights='imagenet',
    pooling='avg'
)
```

```
base_model.trainable = False
```

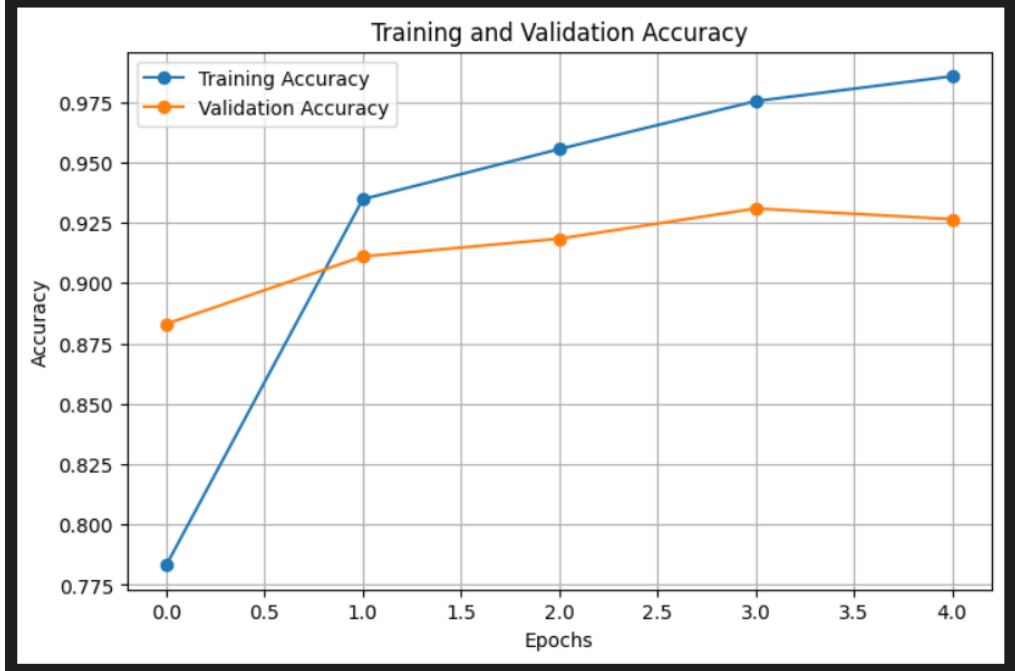
```
inputs = base_model.input
x = tf.keras.layers.Dense(128, activation='relu')(base_model.output)
```

```
history = model.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=5,
    callbacks=[
        tf.keras.callbacks.EarlyStopping(
            monitor='val_loss',
            patience=2,
            restore_best_weights=True
        )
    ]
)
```

[4]

```
Epoch 1/5
170/170 ━━━━━━━━━━━━━━━━━━━━ 100s 568ms/step - accuracy: 0.6146 - loss: 1.3261 - val_accuracy: 0.8831 - val_loss: 0.3796
Epoch 2/5
170/170 ━━━━━━━━━━━━━━━━━━━━ 89s 523ms/step - accuracy: 0.9258 - loss: 0.2537 - val_accuracy: 0.9110 - val_loss: 0.2802
Epoch 3/5
170/170 ━━━━━━━━━━━━━━━━━━━━ 96s 563ms/step - accuracy: 0.9509 - loss: 0.1571 - val_accuracy: 0.9184 - val_loss: 0.2713
Epoch 4/5
170/170 ━━━━━━━━━━━━━━━━━━━━ 88s 516ms/step - accuracy: 0.9767 - loss: 0.0854 - val_accuracy: 0.9309 - val_loss: 0.2537
Epoch 5/5
170/170 ━━━━━━━━━━━━━━━━━━━━ 98s 574ms/step - accuracy: 0.9865 - loss: 0.0542 - val_accuracy: 0.9265 - val_loss: 0.2844
```

```
43/43 ━━━━━━━━━━━━━━━━━━━━ 18s 427ms/step - accuracy: 0.9388 - loss: 0.2441

Model Accuracy on Validation Set: 0.9309
```




```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(m
```

```python
def predict_image(image_path):
    img = load_img(image_path, target_size=(224, 224))
    img = img_to_array(img)
    img = tf.keras.applications.mobilenet_v2.preprocess_input(img)
    img = np.expand_dims(img, axis=0)
    pred = model.predict(img)
    pred_idx = np.argmax(pred, axis=-1)[0]
    return label_mapping[pred_idx]
```

```python
predicted_label = predict_image(r"fruits\orange\orange_spoiled\orange_spoiled_2.jpg")
print("Predicted label:", predicted_label)
```

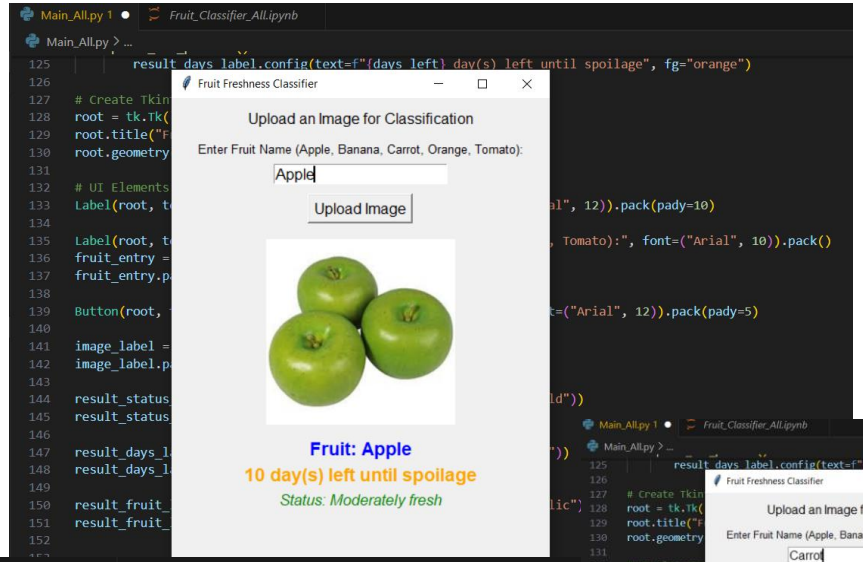
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 788ms/step
Predicted label: orange_spoiled
```

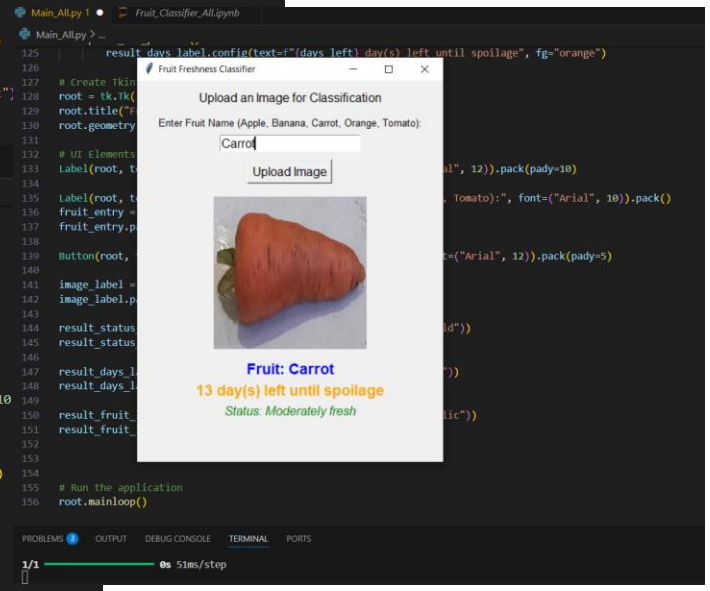
```python
# ------------------------------
# 6. Evaluate Model & Plot Accuracy Curves
# ------------------------------

# Print final accuracy
test_loss, test_acc = model.evaluate(val_dataset)
print(f"\nModel Accuracy on Validation Set: {test_acc:.4f}")

# Plot training & validation accuracy curves
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 5))
plt.plot(history.history['accuracy'], label='Training Accuracy', marker='o')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy', marker='o')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.grid()
plt.show()
```

```
43/43 ━━━━━━━━━━━━━━━━━━━━ 18s 427ms/step - accuracy: 0.9388 - loss: 0.2441
```

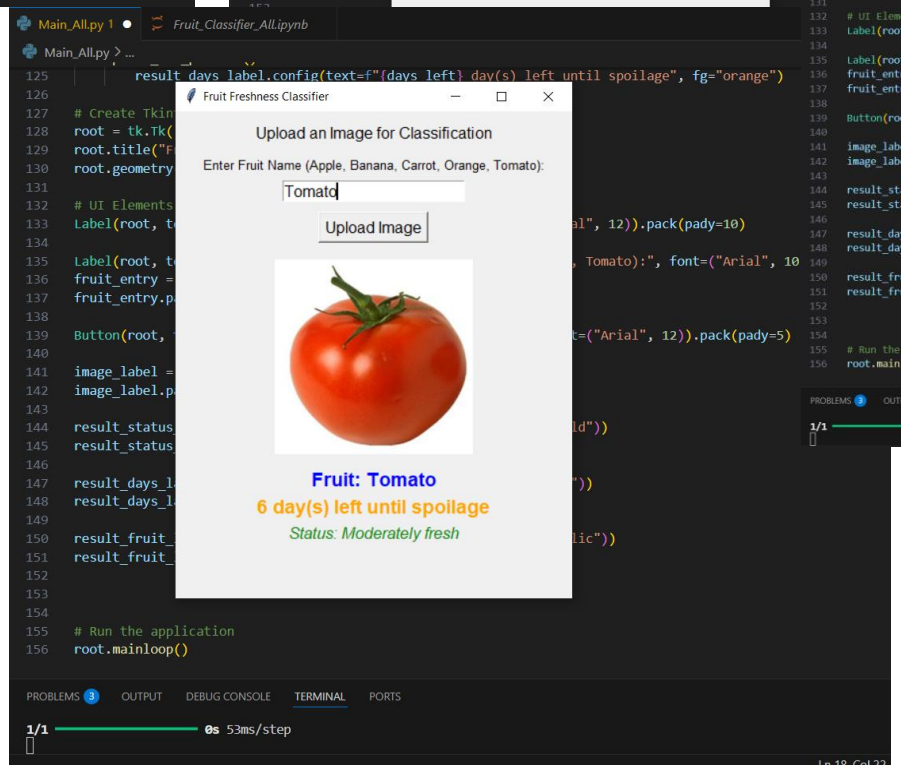
FreshiFy

```python
def get_days_until_spoiled(fruit, status):
    """Return the number of days remaining for the fruit to spoil."""
    return days_to_spoilage.get(fruit, {}).get(status, "Unknown")

def predict_image(image_path, user_fruit):
    """Predict the class of the image and return the fruit, status, and days left."""
    img = load_img(image_path, target_size=(224, 224))
    img = img_to_array(img)
    img = tf.keras.applications.mobilenet_v2.preprocess_input(img)
    img = np.expand_dims(img, axis=0)

    pred = model.predict(img)
    pred_idx = np.argmax(pred, axis=-1)[0]
    predicted_label = label_mapping[pred_idx]

    fruit, status = predicted_label.split('_', 1)
    days_left = get_days_until_spoiled(fruit, status)

    return fruit.capitalize(), status.replace('_', ' ').capitalize(), days_left

def upload_and_predict():
    """Handle image upload, display it, and make a prediction."""
    file_path = filedialog.askopenfilename(filetypes=[("Image files", ".jpg;.jpeg;.png;.bmp")])
    user_fruit = fruit_entry.get().strip().capitalize()

    if not user_fruit:
        result_status_label.config(text="Error: Enter a fruit name", fg="red")
        result_fruit_label.config(text="")
        result_days_label.config(text="")
        return
```






```python
import sys
sys.stdout.reconfigure(encoding='utf-8')

# Load the saved model
model_path = os.path.join('models', 'Fruit_Classifier_ALL.h5')
model = tf.keras.models.load_model(model_path)

# Define label mapping
label_mapping = {
    0: 'apple_fresh',
    1: 'apple_moderately_fresh',
    2: 'apple_near_spoilage',
    3: 'apple_spoiled',
    4: 'banana_fresh',
    5: 'banana_moderately_fresh',
    6: 'banana_near_spoilage',
    7: 'banana_spoiled',
    8: 'carrot_fresh',
    9: 'carrot_moderately_fresh',
    10: 'carrot_near_spoilage',
    11: 'carrot_spoiled',
    12: 'orange_fresh',
    13: 'orange_moderately_fresh',
    14: 'orange_near_spoilage',
    15: 'orange_spoiled',
    16: 'tomato_fresh',
    17: 'tomato_moderately_fresh',
    18: 'tomato_near_spoilage',
    19: 'tomato_spoiled'
}

# Allowed fruit names
allowed_fruits = {"Apple", "Banana", "Carrot", "Orange", "Tomato"}
```



Completion and Future works

- Dataset Created ✔
- Model Trained ✔
- Remaining Days Logic ✔

Completion of work

- Complete label OCR & integrate into mobile app
- Add alert scheduling functionality
- Extend to homemade foods

Future works

FreshiFy

Managing Food Storage Compatibility

IT19970882 | FERNANDO M.G.S.S.A

BSc (Hons) Degree in Information Technology specialization in Information Technology

INTRODUCTION BACKGROUND

- Optimize the storage of homemade and fresh foods by identifying and avoiding harmful proximities between incompatible items.

- The system ensures that foods are stored under ideal conditions to extend freshness and minimize waste.

- This innovative solution addresses the challenges of improper food storage and promotes better household food management.

Research Question

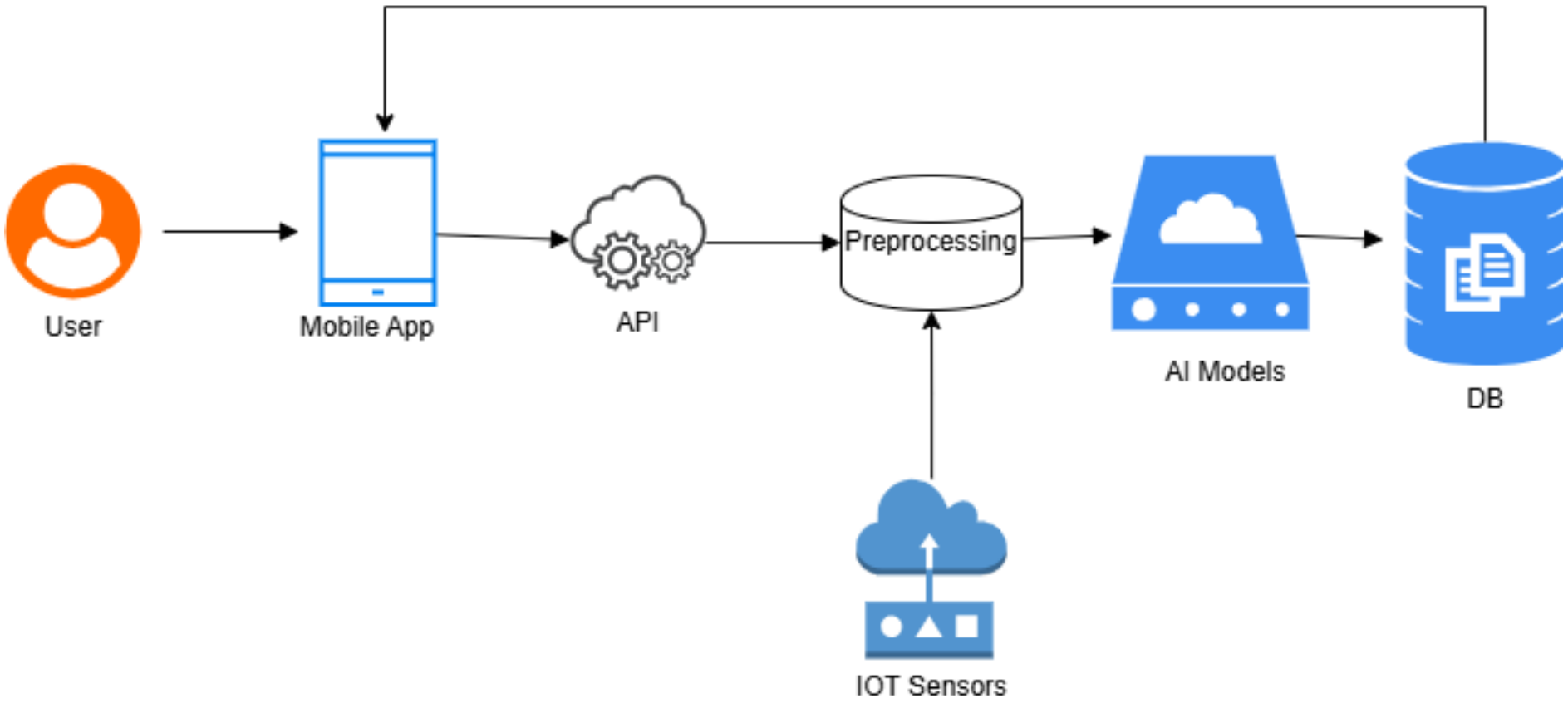
- Improper food storage leads to spoilage, reduced freshness, and increased waste.
- Some foods, like fruits (e.g., apples, bananas), emit ethylene gas, which can accelerate spoilage in nearby sensitive foods like vegetables or dairy.
- Existing systems don’t adjust to changing conditions or specific food interactions.

Specific & Sub Objectives

To develop a system that identifies incompatible food pairings and optimizes storage arrangements to extend freshness.

- Design a system to detect known storage incompatibilities
- Implement a model to categorize foods based on compatibility and freshness.
- Leverage IoT sensors to monitor real-time storage conditions

System Diagram



Tools & Technologies

- **Programming Language :** Python , Node.js
- **Python dependencies :** Scikit-learn, TensorFlow / PyTorch,Python-based rule engines
- **Al Algorithm :** Clustering, Time-series, Rule-based decision logic
- **API :** Flask, Firebase Cloud Messaging (FCM)
- **Backend :** MongoDB, Firebase, AWS
- **Frontend :** React Native
- **Sensors :** DHT22, MQ Series Sensors

Functional & Non-Functional Requirements

Functional Requirements

- Identify incompatible items
- Monitor temperature, humidity, and ethylene gas in real time.
- Notify users of harmful storage and suggest rearrangements.
- Assign foods to zones (e.g., crisper, shelves) based on compatibility.
- Allow manual input for foods and suggest optimal storage.

❑ Non-Functional Requirements

- Handle increasing users and data.
- Deliver alerts within 2-3 seconds.
- Work seamlessly with features like expiration tracking.
- Use low-power IoT sensors for long-lasting operation.

Coding Screenshots

Spoil_Prediction.ipynb > import numpy as np

```python
import pandas as pd
import numpy as np

# Set random seed for reproducibility
np.random.seed(42)

# Generate synthetic data
temperature = np.random.uniform(0, 40, 100)  # Temperature in Celsius
humidity = np.random.uniform(20, 90, 100)  # Humidity percentage
ethylene_gas = np.random.uniform(0, 10, 100)  # Ethylene gas concentration

# Generate labels based on conditions (higher temp, humidity, and ethylene favor spoilage)
labels = np.where(
    (temperature > 25) & (humidity > 60) & (ethylene_gas > 5),
    'Will Spoil',
    'Will not Spoil'
)

# Create DataFrame
df = pd.DataFrame({
    'Temperature': temperature,
    'Humidity': humidity,
    'Ethylene Gas': ethylene_gas,
    'Label': labels
})
```

[1]

	Temperature	Humidity	Ethylene Gas	Label
0	14.981605	22.200043	6.420316	Will not Spoil
1	38.028572	64.548729	0.841400	Will not Spoil
2	29.279758	42.004919	1.616287	Will not Spoil
3	23.946339	55.599948	8.985542	Will not Spoil
4	6.240746	83.529653	6.064291	Will not Spoil

```python
# Replace labels with numerical values
df['Label'] = df['Label'].replace({'Will Spoil': 1, 'Will not Spoil': 0})
```

[2]

```
<ipython-input-2-c1087700380a>:2: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a future version. To retain the old
  df['Label'] = df['Label'].replace({'Will Spoil': 1, 'Will not Spoil': 0})
```

```python
import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf
from sklearn import metrics
```

[3]

```python
df.describe()
```

[6]

	Temperature	Humidity	Ethylene Gas	Label
count	100.000000	100.000000	100.000000	100.000000
mean	18.807230	54.848221	5.176013	0.060000
std	11.899576	20.517788	2.934262	0.238683
min	0.220885	20.486649	0.050616	0.000000
25%	7.728030	36.940317	2.768799	0.000000
50%	18.565698	55.393740	5.625549	0.000000
75%	29.208125	73.632852	7.523669	0.000000
max	39.475477	88.995532	9.900539	1.000000

```python
# See distribution of target class
df['Label'].value_counts().plot.bar(figsize=(5,4), xlabel='Label', ylabel='Count', title='Count of Labels')
plt.show()
```

[7]

Count of Labels

FreshiFy

Coding Screenshots

Spoil_Prediction.ipynb × Spoil_Sensors.py

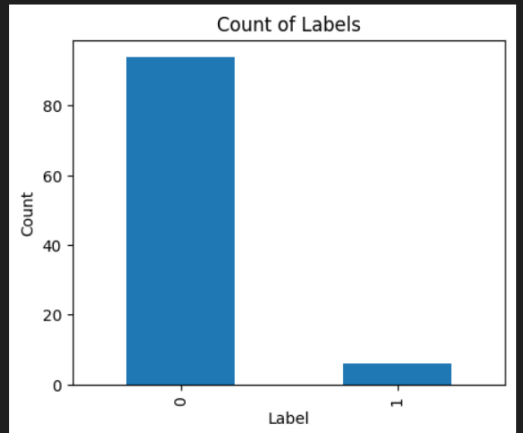
Spoil_Prediction.ipynb > import numpy as np

+ Code + Markdown | Run All ···

kera

```
# See distribution of target class
df['Label'].value_counts().plot.bar(figsize=(5,4), xlabel='Label', ylabel='Count', title='Count of Labels')
plt.show()
```

[7]



Spoil_Prediction.ipynb × Spoil_Sensors.py

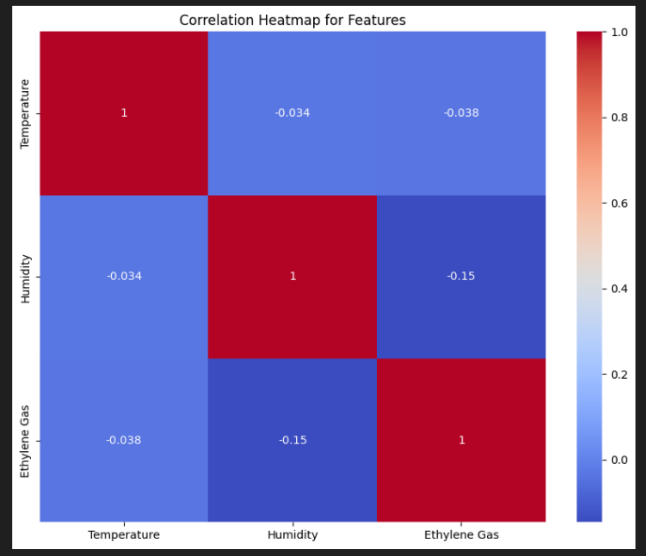
Spoil_Prediction.ipynb > import numpy as np

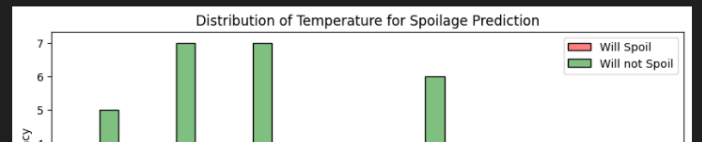
+ Code + Markdown | Run All ···

```
# Example usage:
analyze_spoilage_data(df)
```

[9]

```
Number of samples that will spoil = 6 (6.000%)
Number of samples that will not spoil = 94 (94.000%)
```





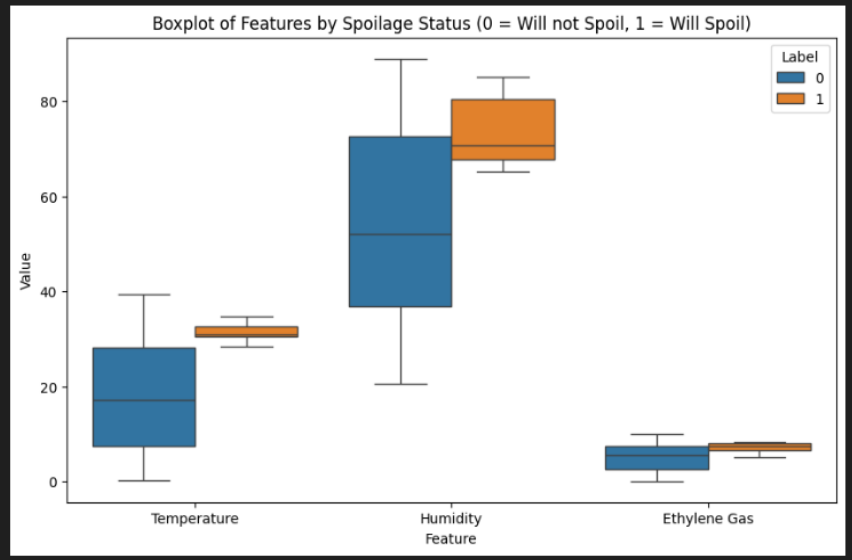
Spoil_Prediction.ipynb × Spoil_Sensors.py

Spoil_Prediction.ipynb > import numpy as np

+ Code + Markdown | Run All ···

```
Descriptive statistics for Humidity (samples that will spoil):
count     6.000000
mean     73.658987
std       8.459689
min      65.162095
25%      67.663539
50%      70.627111
75%      80.494093
max      85.078836
Name: Humidity, dtype: float64

Descriptive statistics for Ethylene Gas (samples that will spoil):
count    6.000000
mean     7.095104
...
75%      8.065947
max      8.226006
Name: Ethylene Gas, dtype: float64
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...



FreshiFy

Coding Screenshots

Spoil_Prediction.ipynb Spoil_Sensors.py

Spoil_Prediction.ipynb > import numpy as np

+ Code + Markdown | Run All ···

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.9783 - loss: 0.1438 - val_accuracy: 0.9259 - val_loss: 0.1888
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
# Evaluate the model on the test set
loss, accuracy = rnn_model.evaluate(X_test_scaled, y_test, verbose=0)
print("RNN Test Accuracy:", accuracy)

# Generate predictions (probabilities) and convert them to binary predictions
y_pred_prob = rnn_model.predict(X_test_scaled)
y_pred = (y_pred_prob > 0.5).astype(int).reshape(-1)

# Print the classification report
print("\nRNN Classification Report:")
print(classification_report(y_test, y_pred))
```

[22]

```
RNN Test Accuracy: 0.9298245906829834
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 218ms/step

RNN Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.87      0.93        31
           1       0.87      1.00      0.93        26

    accuracy                           0.93        57
   macro avg       0.93      0.94      0.93        57
weighted avg       0.94      0.93      0.93        57
```

```
# Save the RNN model in HDF5 (.h5) format
rnn_model.save('rnn_model.h5')
print("RNN model saved in 'rnn_model.h5'.")
```

[23]

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native keras format, e.g. `model.save('my
RNN model saved in 'rnn_model.h5'.
```

```
# -------------------------
# Load the model from the .h5 file
rnn_loaded = load_model('rnn_model.h5')
```

[24]

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
```

View Go Run Terminal Help Sensors

Spoil_Prediction.ipynb Spoil_Sensors.py

Spoil_Prediction.ipynb > import numpy as np

+ Code + Markdown | Run All ···

```
print("RNN model saved in 'rnn_model.h5'.")
```

[23]

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native keras format, e.g. `
RNN model saved in 'rnn_model.h5'.
```

```
# -------------------------
# Load the model from the .h5 file
rnn_loaded = load_model('rnn_model.h5')
```

[24]

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
```

```
# Define sample inputs as variables (example values)
sample_temperature = 30.0  # in Celsius
sample_humidity = 65.0     # in percentage
sample_ethylene = 6.0      # ethylene gas concentration

# Combine the sample inputs into a NumPy array (2D shape)
sample_input = np.array([[sample_temperature, sample_humidity, sample_ethylene]])

# Scale the sample input using the same scaler
sample_input_scaled = scaler.transform(sample_input)
# Reshape to 3D (samples, timesteps, features)
sample_input_scaled = sample_input_scaled.reshape((sample_input_scaled.shape[0], 1, sample_input_scaled.shape[1]))

# Make a prediction using the loaded RNN model
sample_pred_prob = rnn_loaded.predict(sample_input_scaled)
sample_pred = (sample_pred_prob > 0.5).astype(int).reshape(-1)

# Define mapping from numeric to text labels
label_mapping = {1: 'Will Spoil', 0: 'Will not Spoil'}

# Convert numeric prediction to text
text_prediction = [label_mapping[pred] for pred in sample_pred]

print("Textual prediction for the sample input:", text_prediction)
```

[25]

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 304ms/step
Textual prediction for the sample input: ['Will Spoil']
```

FreshiFy

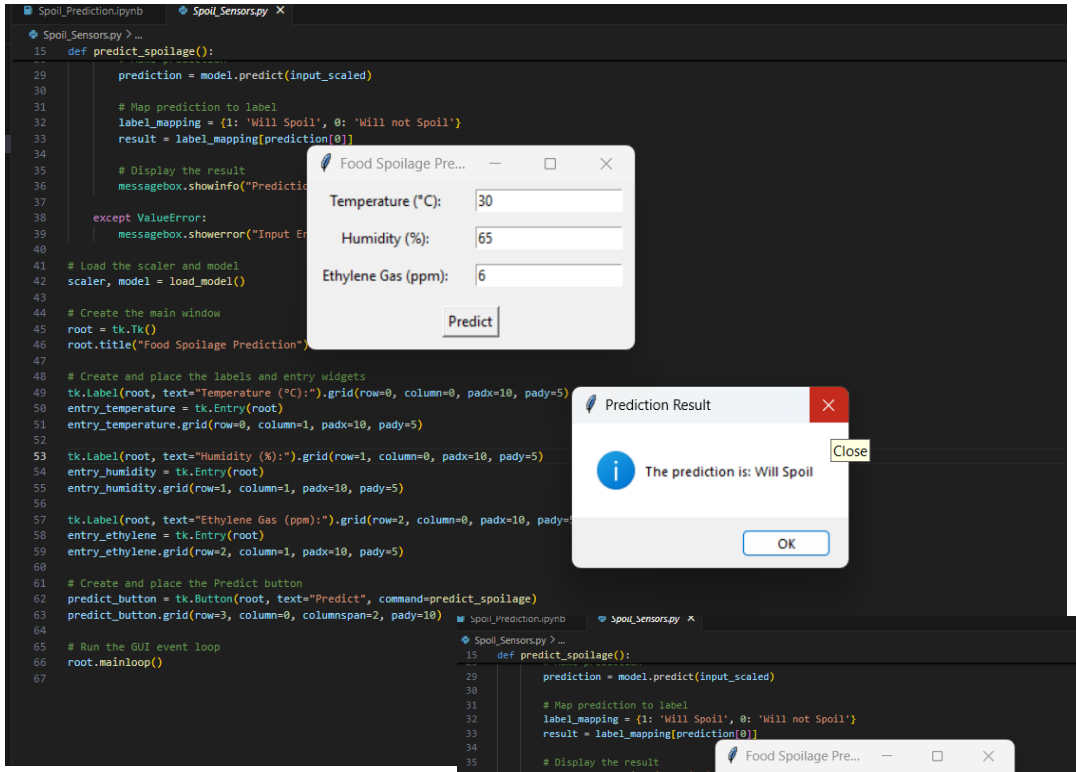
Coding Screenshots

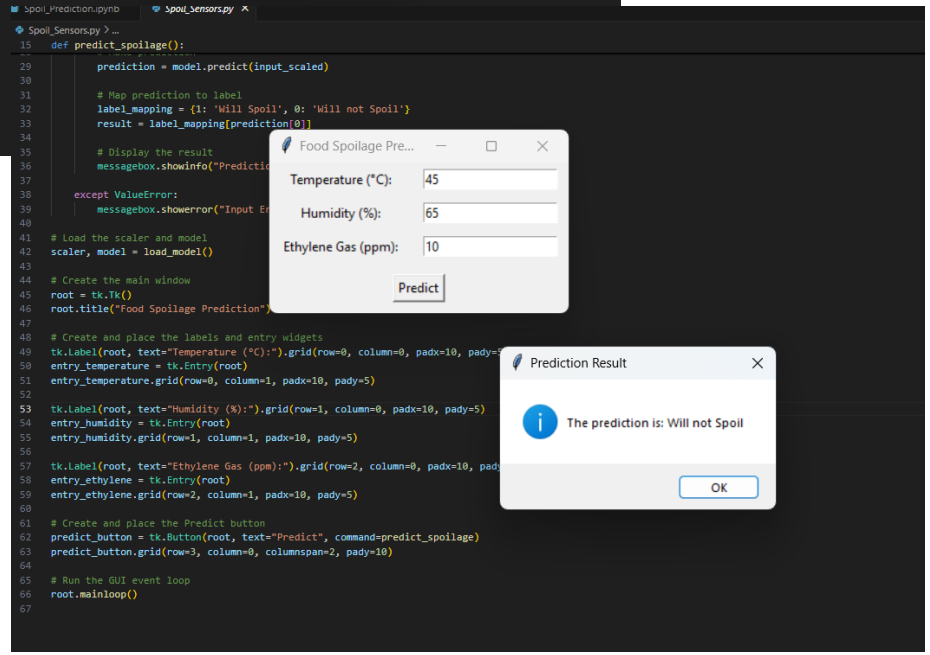
View Go Run Terminal Help Sensors

Spoil_Prediction.ipynb | Spoil_Sensors.py

Spoil_Sensors.py > ...

```python
import tkinter as tk
from tkinter import messagebox
import numpy as np
import pickle

# Function to load the scaler and model
def load_model():
    with open('scaler.pkl', 'rb') as scaler_file:
        scaler = pickle.load(scaler_file)
    with open('svm_model.pkl', 'rb') as model_file:
        model = pickle.load(model_file)
    return scaler, model

# Function to make a prediction based on user input
def predict_spoilage():
    try:
        # Retrieve user input
        temperature = float(entry_temperature.get())
        humidity = float(entry_humidity.get())
        ethylene = float(entry_ethylene.get())

        # Prepare the input data
        input_data = np.array([[temperature, humidity, ethylene]])

        # Scale the input data
        input_scaled = scaler.transform(input_data)

        # Make prediction
        prediction = model.predict(input_scaled)

        # Map prediction to label
        label_mapping = {1: 'Will Spoil', 0: 'Will not Spoil'}
        result = label_mapping[prediction[0]]

        # Display the result
        messagebox.showinfo("Prediction Result", f"The prediction is: {result}")

    except ValueError:
        messagebox.showerror("Input Error", "Please enter valid numerical values.")

# Load the scaler and model
scaler, model = load_model()

# Create the main window
root = tk.Tk()
root.title("Food Spoilage Prediction")

# Create and place the labels and entry widgets
tk.Label(root, text="Temperature (°C):").grid(row=0, column=0, padx=10, pady=5)
entry_temperature = tk.Entry(root)
entry_temperature.grid(row=0, column=1, padx=10, pady=5)

tk.Label(root, text="Humidity (%):").grid(row=1, column=0, padx=10, pady=5)
entry_humidity = tk.Entry(root)
entry_humidity.grid(row=1, column=1, padx=10, pady=5)
```





Completion and Future works

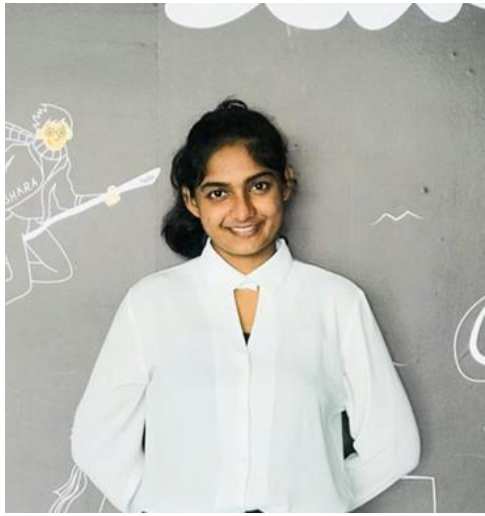
- ✓ Trained the model using synthetic food spoilage data
- ✓ Developed a basic output interface using **Tkinter**
- ✓ Verified model predictions on sample data

Future Works

- Integrate real sensor data into the model
- Build a REST API for model communication
- Design a mobile UI
- Integrate and image upload features

FreshiFy

Suggest a recipe based on the food that are near expiration



IT21829710 | PERERA P.A.E.U.K.

BSc (Hons) Degree in Information Technology specialization in Information Technology

INTRODUCTION

- Leverages artificial intelligence to provide personalized recipe suggestions,
- encouraging users to creatively use ingredients nearing expiration.
- By matching available items with a recipe database, the system empowers users to minimize waste.

Research Question

Problem : Households often struggle to find practical ways to use food nearing expiration, leading to significant food waste.

Existing recipe apps lack real-time integration with household inventories or the ability to prioritize expiring ingredients.

Identifies expiring items and suggests appropriate recipes.

Specific & Sub Objectives

Sub

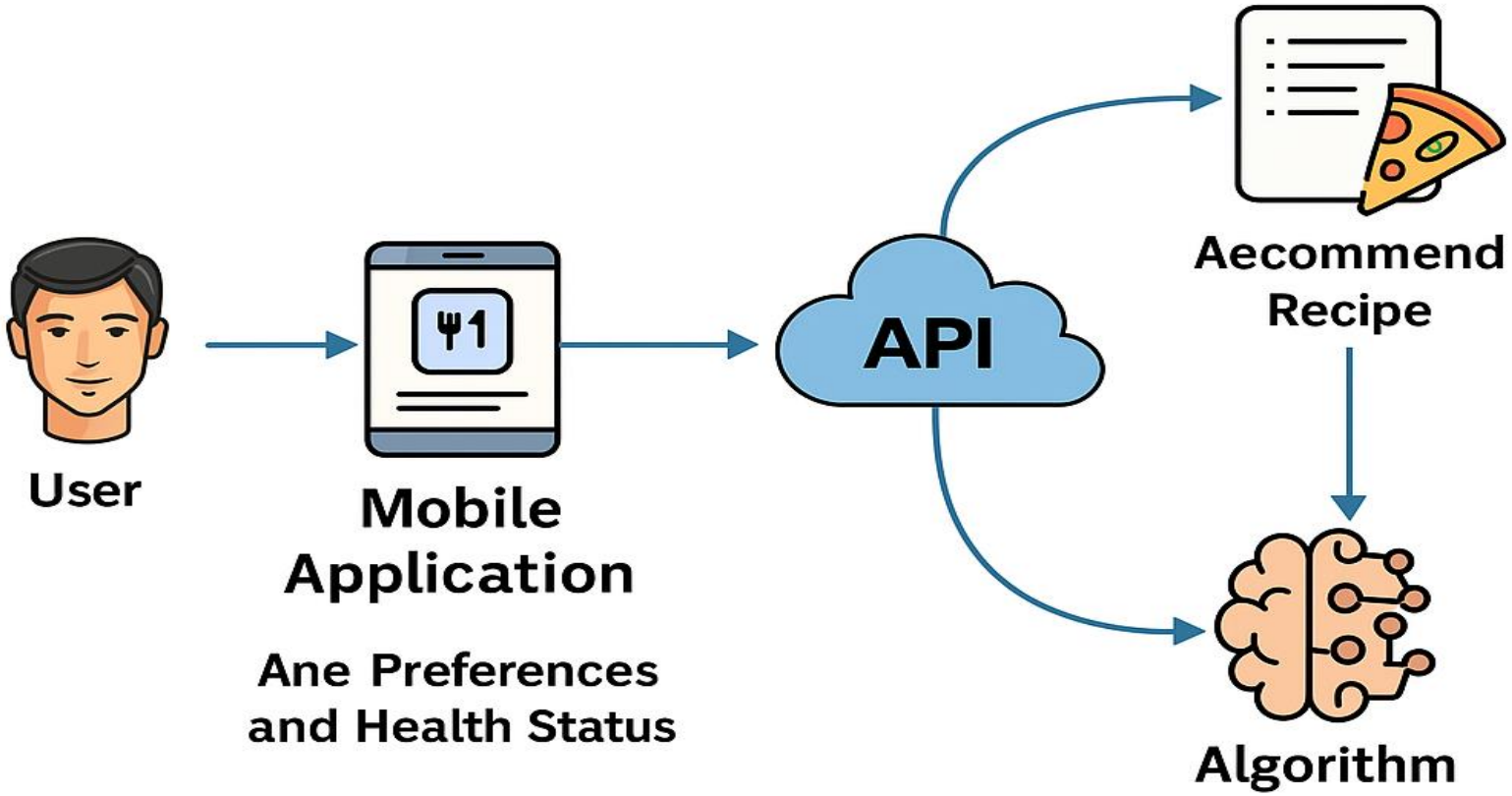
To utilize inventory data of expiring food items to recommend recipes dynamically while integrating user preferences and interactive chatbot assistance.

Specific

Identify food items nearing expiration through inventory tracking.

- Match available ingredients with recipes using a recommendation system (e.g., collaborative or content-based filtering).
- Provide step-by-step cooking guidance via an interactive chatbot.

System Diagram



Tools & Technologies

- Programming Language : Python, Kotlin
- Python dependencies : hugging face dataset, pandas, flask, pickle, nltk,scikit learn, numpy
- Al Algorithm : Cosine Similarity
- API and Backend : Python flask
- Databases : Sqlite
- Frontend : kotlin, jetpack composer, retrofit

Requirements

Functional Requirements

Ability to track and identify food items nearing expiration.

- Matching expiring ingredients with recipes in a database.
- Providing recipe recommendations based on user preferences (e.g., vegetarian, quick meals).
- Displaying recipe details such as name, preparation time, and difficulty.
- Providing step-by-step cooking instructions via an interactive chatbot.

Non- Functional Requirements

User-friendliness.

- Accessibility on mobile devices.
- Usability for both novice and experienced users.
- Reliability in recipe accuracy and inventory tracking.
- Scalable for expanding recipe databases and inventory sizes.

Personal Requirements

- Home cooks interested in reducing food waste.
- Culinary enthusiasts.
- Nutritionists or food experts for recipe validation.
- Software testers for app usability and functionality.

Software Requirements

- A smartphone running on Android testing.
- Python for machine learning models and backend logic.
- Flask for backend development.
- Flutter for mobile app development.
- PyCharm for development.
- Git for version control and collaboration.

Coding Screenshots

16s

```python
from datasets import load_dataset

ds = load_dataset("nguyenkhoa/recipe1M_v2")
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Goog
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
```

```
README.md: 100% 636/636 [00:00<00:00, 25.1kB/s]
train-00000-of-00001.parquet: 100% 89.2M/89.2M [00:00<00:00, 135MB/s]
test-00000-of-00001.parquet: 100% 19.2M/19.2M [00:00<00:00, 121MB/s]
val-00000-of-00001.parquet: 100% 19.0M/19.0M [00:00<00:00, 161MB/s]
Generating train split: 100% 147062/147062 [00:05<00:00, 17129.11 examples/s]
Generating test split: 100% 31660/31660 [00:00<00:00, 125993.16 examples/s]
Generating val split: 100% 31363/31363 [00:00<00:00, 125310.65 examples/s]
```

0s

```python
ds['test'][0]
```

```
{'title': 'Crunchy Onion Potato Bake',
 'ingredients_processed': '2 12 cups milk. 1 12 cups water. 14 cup butter. mashed potatoes, 1 box, homestyle. 1 (8 ounce) can whole kernel corn
(drained). 1 cup cheddar cheese. 1 cup French-fried onions. ',
 'directions': 'Preheat oven to 350 degrees Fahrenheit. Spray pan with non stick cooking spray. Heat milk, water and butter to boiling; stir in
contents of both pouches of potatoes; let stand one minute. Stir in corn. Spoon half the potato mixture in pan. Sprinkle half each of cheese and
onions; top with remaining potatoes. Sprinkle with remaining cheese and onions. Bake 10 to 15 minutes until cheese is melted. Enjoy ! ',
 'NER': 'milk. water. butter. mashed potatoes. whole kernel corn. cheddar cheese. French - fried onions. ',
 'link': 'http://www.food.com/recipe/crunchy-onion-potato-bake-479149'}
```

FreshiFy

```
[6] import pandas as pd
```

```
recipes = []
for item in ds['train']:
    recipe = {
        "title": item["title"],
        "ingredients": item["ingredients_processed"],
        "directions": item["directions"],
        "NER": item["NER"],
        "link": item["link"]
    }
    recipes.append(recipe)
```

```
[8] recipes_test = []
    for item in ds['test']:
        recipe = {
            "title": item["title"],
            "ingredients": item["ingredients_processed"],
            "directions": item["directions"],
            "NER": item["NER"],
            "link": item["link"]
        }
        recipes_test.append(recipe)
```

```
[9] recipes_val= []
    for item in ds['val']:
        recipe = {
            "title": item["title"],
            "ingredients": item["ingredients_processed"],
            "directions": item["directions"],
            "NER": item["NER"],
            "link": item["link"]
        }
        recipes_val.append(recipe)
```

```
[10] df = pd.DataFrame(recipes+recipes_test+recipes_val)
```

```
[11] df.to_csv("recipes.csv", index=False)
```

```python
[12] import re
     import nltk
     from nltk.corpus import stopwords
     from nltk.tokenize import word_tokenize
     from nltk.stem import WordNetLemmatizer

     nltk.download("punkt")
     nltk.download("stopwords")
     nltk.download("wordnet")
     nltk.download('punkt_tab')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
True
```

```python
[13] def preprocess_ingredients(ingredients):
         ingredients = ingredients.lower()
         ingredients = re.sub(r'[^a-zA-Z0-9\s]', '', ingredients)

         # tokenization
         tokens = word_tokenize(ingredients)

         # remove stop words
         stop_words = set(stopwords.words("english"))

         tokens = [word for word in tokens if word not in stop_words]

         preprocessed_text = " ".join(tokens)

         return preprocessed_text
```

```python
[14] from sklearn.feature_extraction.text import TfidfVectorizer
```

```python
[15] # convert ingredient lists to a string format
     # formating the ingredient column to a structured format like tokenizing, lemmatizing
     df["ingredients_str"] = df["ingredients"].apply(lambda x: preprocess_ingredients(x))

     # create a TF-IDF vectorizer
     vectorizer = TfidfVectorizer()
     ingredient_vectors = vectorizer.fit_transform(df["ingredients_str"])

     # model serialization
     # save the vectorizer
     import pickle
     with open("ingredient_vectorizer.pkl", "wb") as f:
         pickle.dump(vectorizer, f)
```

recipes_prediction_backend

Project

- recipes_prediction_backend C:\RP\recipes_prediction\re
  - venv library root
  - .gitignore
  - food_suggestion.ipynb
  - ingredient_vectorizer.pkl
  - main.py
  - recipes_processed.csv
  - vectorizer.pkl
- External Libraries
- Scratches and Consoles

main.py

```python
@app.route('/chat', methods=['POST'])
def chat():
    user_input = request.json.get("message")

    refined = f"act as food instructer or health instruct based on the given message {use

    # Make request to local Ollama server
    response = requests.post("http://localhost:11434/api/generate", json={
        "model": "gemma:2b",
        "prompt": user_input,
        "stream" : False
    })

    full_reply = ""
```

Run main

```
* Debugger PIN: 724-088-846
127.0.0.1 - - [06/Apr/2025 22:45:33] "POST /predict-recipies HTTP/1.1" 200 -
127.0.0.1 - - [06/Apr/2025 22:45:43] "POST /predict-recipies HTTP/1.1" 200 -
127.0.0.1 - - [06/Apr/2025 22:47:00] "POST /chat HTTP/1.1" 200 -
127.0.0.1 - - [06/Apr/2025 22:51:37] "POST /chat HTTP/1.1" 200 -
```

recipes_prediction_backend > main.py 37:7 LF UTF-8 4 spaces Python 3.10 (recipes_prediction_backend)

food-prediction / **predict**

Save

POST http://127.0.0.1:5000/predict-recipies Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
{
  "records_count":100, //maxcount 1000
  "food_items" : ["chicken","carrot"],
  "user_exclud" : ["apple"]
}
```

Body Cookies Headers (5) Test Results 200 OK 235 ms 90.78 KB Save as example

Pretty Raw Preview Visualize JSON

```
{
    "recipies": [
        {
            "directions": "Coat chick in flour. Deep Fry in vegetable oil. drain. ",
            "ingredients": "1 whole chicken. 2 cups flour. ",
            "title": "Fried Chicken"
        },
        {
```

Completion and Future works

Completion of the components

- Train the model for Recipe Suggestion
- Identify food allergies by using the user inputs
- Suggest recommendation for food that are near expiration
- Check output using postman

Future Implementations

- Create user interfaces
- Create a database based on the user registration and maintain a database regarding the user preferences and and allergies

COMMERCIALIZATION

Target Audience

- Family members
- Housewives

THANK YOU!