Golden Harvesting: A Predictive Model for Apple Quality Assurance

Team Members:
Aryaman Abbi
Akkiraju Pruthvi
Arunima
Nivid Saxena

Team ID: SWTID1749727628

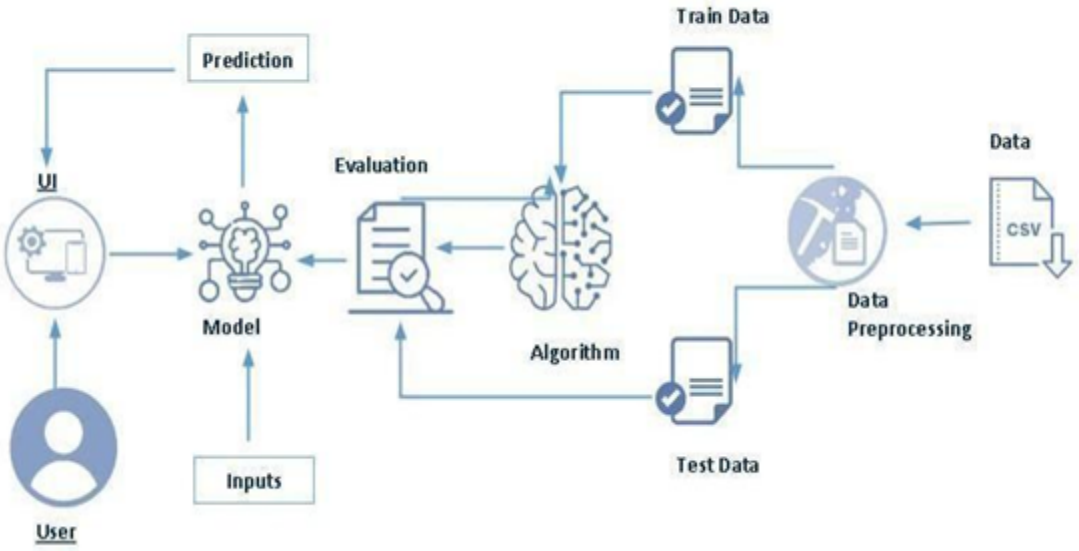
SmartInternz

Introduction

In the agricultural industry, ensuring the quality of produce is paramount for market competitiveness and consumer satisfaction. Apples, being one of the most widely consumed fruits globally, require rigorous quality assessment. Traditional methods of quality inspection can be time-consuming, subjective, and labor-intensive. This project aims to develop a predictive model for apple quality assurance, leveraging machine learning techniques to automate and enhance the quality classification process. This Golden Harvesting project will analyze various characteristics of apples to classify them as 'good' or 'bad' quality, thereby facilitating efficient sorting and quality control.

Technical Architecture:

The technical architecture for the Golden Harvesting project is designed to process apple quality data efficiently, train a predictive model, and provide real-time quality predictions.



- **UI (User Interface):** A web-based interface (Flask) allows users to input apple characteristics.
- **User:** Interacts with the UI to provide input parameters.
- **Prediction:** The trained Machine Learning model processes the input and generates a quality prediction (Good/Bad).
- **Evaluation:** The model's performance is evaluated using metrics like accuracy, precision, recall, and F1-score.
- **Model:** The core Machine Learning model (e.g., Random Forest, XGBoost, Logistic Regression) trained on historical apple quality data.
- **Algorithm:** Various classification algorithms are explored and tuned to find the best performing model.
- **Inputs:** Features such as size, weight, redness, ripeness, etc., are provided by the user or extracted from sensors.
- **Train Data:** A subset of the collected data used to train the machine learning model.
- **Test Data:** A separate subset of the collected data used to evaluate the trained model's performance on unseen data.
- **Data Preprocessing:** Cleaning, transforming, and preparing raw data for model training (e.g., handling missing values, scaling features).
- **Data:** Raw dataset containing apple characteristics and their corresponding quality labels.
- **CSV:** The primary format for the raw dataset.

Project Flow:

The project follows a standard machine learning pipeline, adapted for the apple quality prediction task:

1. **User Interaction:** The user interacts with the web-based UI to input the relevant features of an apple (e.g., size, weight, color, etc.).
2. **Input Analysis by Model:** The entered input is sent to the deployed machine learning model (e.g., best.pkl loaded by app1.py).
3. **Prediction Generation:** The model analyzes the input features and generates a prediction indicating whether the apple is of 'Good' or 'Bad' quality (0 for bad, 1 for good).
4. **Result Display:** The prediction result is then showcased on the web page to the user.

Prior Knowledge:

To understand effectively and contribute to this project, a foundational understanding of the following concepts is beneficial:

- **Python Programming:** Proficiency in Python, including libraries for data manipulation (Pandas, NumPy) and machine learning (Scikit-learn, XGBoost).
- **Machine Learning Fundamentals:**
- **Supervised Learning:** Understanding of classification problems.
- **Data Preprocessing:** Concepts like handling missing values, encoding categorical features, and feature scaling.
- **Model Training & Evaluation:** Knowledge of splitting data into training and testing sets, and evaluating model performance using metrics.
- **Classification Algorithms:** Familiarity with algorithms such as Logistic Regression, Random Forest, and XGBoost.
- **Flask:** Basic understanding of Flask framework for building web applications.
- **HTML/CSS:** Fundamental knowledge for creating web pages and forms.
- **Jupyter Notebook/Colab:** Experience with interactive development environments for data science.

Project Structure:

The project is organized into a structured directory to manage code, data, and models effectively.
Golden_Harvesting_A_Predictive_Model_for_Apple_Quality_Assurance.ipynb and app1.py,

```
Golden-Harvesting-A-Predictive-Model-for-Apple-Quality-Assurance/
├── static/
├── templates/                # HTML templates for the web app
│   ├── index.html            # Main landing page
│   ├── inner-page.html       # Input form page
│   └── output.html           # Prediction result page
├── app1.py                   # Flask application for deployment
└── Golden_Harvesting_A_Predictive_Model_for_Apple_Quality_Assurance.ipynb
└── apple_quality.csv
└── best.pkl
└── README.md                 # Project description
└── requirements.txt          # Python dependencies
```

Milestone 1: Define Problem / Problem Understanding

This milestone focuses on thoroughly understanding the problem that Golden Harvesting aims to solve, its business implications, and the technical requirements.

Business Problem:

The primary business problem is the inefficient and subjective process of apple quality assessment in the agricultural supply chain. This leads to:

- **Increased Labor Costs:** Manual inspection requires significant human effort.
- **Inconsistency in Quality:** Subjective human judgment can lead to variations in classification.
- **Time Consumption:** Slow inspection processes can delay market entry.
- **Waste Reduction:** Accurate early detection of bad quality apples reduces post-harvest losses.
- **Customer Satisfaction:** Delivering consistent high-quality apples enhances consumer trust and brand reputation.

Key Requirements:

- Develop a predictive model capable of classifying apple quality (Good/Bad).
- The model should achieve high accuracy and reliability.
- Create a user-friendly web interface for inputting apple characteristics and displaying predictions.
- The system should be scalable and easily deployable.

Literature Survey:

Research on existing solutions for fruit quality prediction using machine learning, including:

- Computer vision techniques for defect detection.
- Sensor-based data collection for internal and external quality attributes.
- Application of various classification algorithms (SVM, Neural Networks, Ensemble Methods) in agricultural contexts.

Social & Business Impact:

- **Social Impact:**
    - Reduced food waste by accurately identifying and diverting lower quality apples for other uses.
    - Improved food safety and quality for consumers.
    - Potential for fair pricing for farmers based on consistent quality grading.
- **Business Impact:**
    - Streamlined quality control processes, leading to cost savings.
    - Increased efficiency in sorting and packaging.
    - Enhanced reputation and market advantage for producers delivering consistent quality.
    - Data-driven decision-making for cultivation and harvesting practices.

Milestone 2: Data Collection & Preparation

This milestone covers the critical steps of acquiring, cleaning, and preparing the apple quality dataset for machine learning.

Dataset Collection:

The project relies on a dataset containing various attributes of apples along with their quality labels.

Import and Read Dataset from Git

```
import os
if not os.path.exists("Golden-Harvesting-A-Predictive-Model-for-Apple-Quality-Assurance"):
    !git clone https://github.com/ary4m4n03/Golden-Harvesting-A-Predictive-Model-for-Apple-Quality-Assurance.git
%cd Golden-Harvesting-A-Predictive-Model-for-Apple-Quality-Assurance
```

Library Imports:

The following Python libraries are essential for data manipulation, visualization, and machine learning:

Import Libraries

```
[ ] import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns
    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import StandardScaler
    from sklearn.linear_model import LogisticRegression
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, ConfusionMatrixDisplay
    import xgboost as xgb
    import pickle
    from sklearn.model_selection import GridSearchCV
```

Data Loading and Initial Exploration:

- Loading the Dataset:
  The dataset is loaded into a Pandas DataFrame.

Read Dataset

```
[ ]  data = pd.read_csv("apple_quality.csv")
     print("Dataset loaded successfully.")
     print("Shape of the dataset:", data.shape)
```

- Displaying First 5 Rows:

Data Preprocessing

```
[ ]  # Initial Exploration
     print("\nDataset Info:")
     print(data.info())
     print("\nFirst few rows:")
     print(data.head())
```

- Descriptive Statistics:
  data.describe() generates descriptive statistics, providing insights into the central tendency, dispersion, and shape of the dataset's distribution.
  data.describe(include='all') is used to include non-numeric columns as well.

Exploratory Data Analysis

```
data.describe()
```

	A_id	Size	Weight	Sweetness	Crunchiness	Juiciness	Ripeness	Acidity	Quality
count	4000.000000	4000.000000	4000.000000	4000.000000	4000.000000	4000.000000	4000.000000	4000.000000	4000.000000
mean	1999.500000	-0.502695	-0.991229	-0.472248	0.984194	0.513127	0.498102	0.076639	0.501000
std	1154.844867	1.917446	1.574517	1.931684	1.369437	1.917024	1.866614	2.101441	0.500062
min	0.000000	-5.750201	-5.075890	-5.548946	-2.684440	-4.757179	-4.578510	-5.709299	0.000000
25%	999.750000	-1.816765	-2.011770	-1.738425	0.062764	-0.801286	-0.771677	-1.377424	0.000000
50%	1999.500000	-0.513703	-0.984736	-0.504758	0.998249	0.534219	0.503445	0.022609	1.000000
75%	2999.250000	0.805526	0.030976	0.801922	1.894234	1.835976	1.766212	1.510493	1.000000
max	3999.000000	4.738963	3.095097	4.612442	4.641439	5.791870	5.573044	5.842368	1.000000

Handling Missing Values:

The notebook identifies missing values using data.isnull().sum(). It shows a single missing value in the 'A_id' column, which is then dropped.

```
# Handling missing values
print("\nMissing Values(per column):")
print(data.isnull().sum())
data = data.dropna()
print("\nAfter dropping missing values, shape:",data.shape)
```

Handling Non-Numeric Values and Data Cleaning:

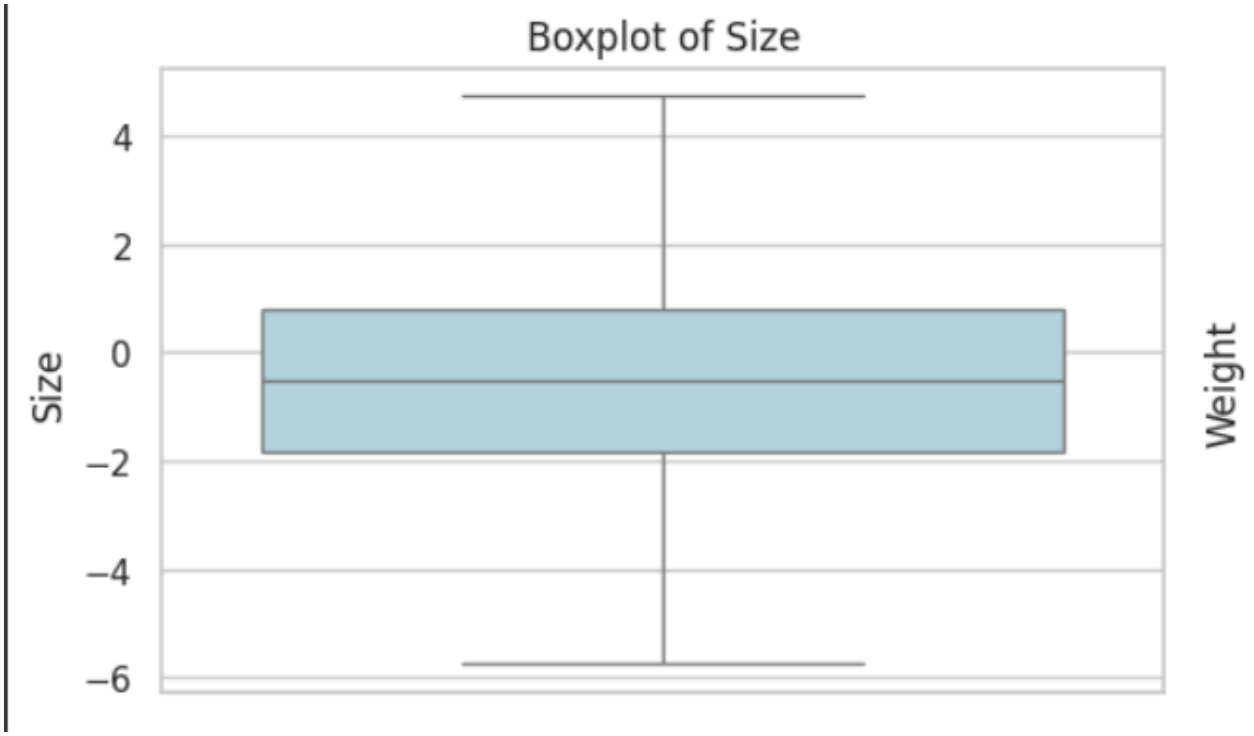
The 'A_id' column is identified as non-numeric and subsequently dropped. The 'Quality' column, which is the target variable, contains 'good' and 'bad' as strings. It's converted to numerical representation (0 for 'bad', 1 for 'good') and also contains an 'object' type value 'goodA'. This entry is replaced with 'good' and then converted to numeric.

```
# Handling categorical data
data['Quality']=data['Quality'].astype('category').cat.codes
print("\nEncoded 'Quality' values:\n", data['Quality'].value_counts())
data['Acidity']=pd.to_numeric(data['Acidity'],errors='coerce')
data = data.dropna()
print("\nAfter converting 'Acidity', new shape:",data.shape)
```

Additionally, other columns (Size, Weight, Sweetness, Crunchiness, Juiciness, Ripeness, Acidity) are identified as having non-numeric characters (e.g., 'object' type with spaces in values). These are converted to numeric using pd.to_numeric with errors='coerce' and then resulting NaN values are imputed with the mean of their respective columns.

Outlier Detection:

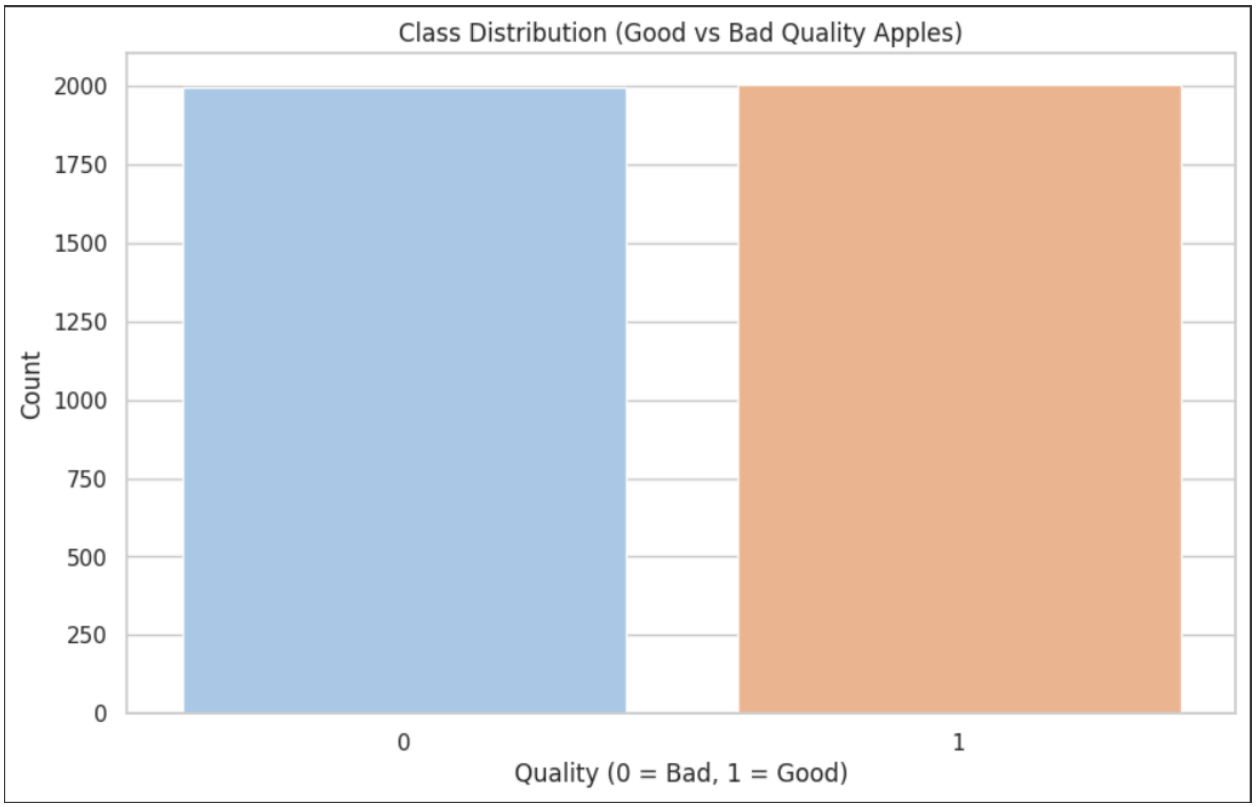
```python
# Outlier handling via IQR
def cap_outliers(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR
    df[column] = np.where(df[column] > upper, upper,
                 np.where(df[column] < lower, lower, df[column]))
    return df

numeric_cols = ['Size', 'Weight', 'Sweetness', 'Crunchiness', 'Juiciness', 'Ripeness', 'Acidity']
for col in numeric_cols:
    data = cap_outliers(data, col)

plt.figure(figsize=(16, 10))
for i, col in enumerate(numeric_cols, 1):
    plt.subplot(3, 3, i)
    sns.boxplot(y=data[col], color="lightblue")
    plt.title(f"Boxplot of {col}")
plt.tight_layout()
plt.show()
```



Class Balance Check:

It's important to check the distribution of the target variable ('Quality') to see if there's a class imbalance, which can affect model performance.

```
# Class balance check
sns.countplot(x='Quality', data=data, palette='pastel')
plt.title("Class Distribution (Good vs Bad Quality Apples)")
plt.xlabel("Quality (0 = Bad, 1 = Good)")
plt.ylabel("Count")
plt.show()
print("\nClass Distribution:\n", data['Quality'].value_counts())
```



The output from the notebook indicates a relatively balanced dataset

Milestone 3: Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is a crucial step to understand the underlying patterns, relationships, and anomalies within the apple quality dataset.

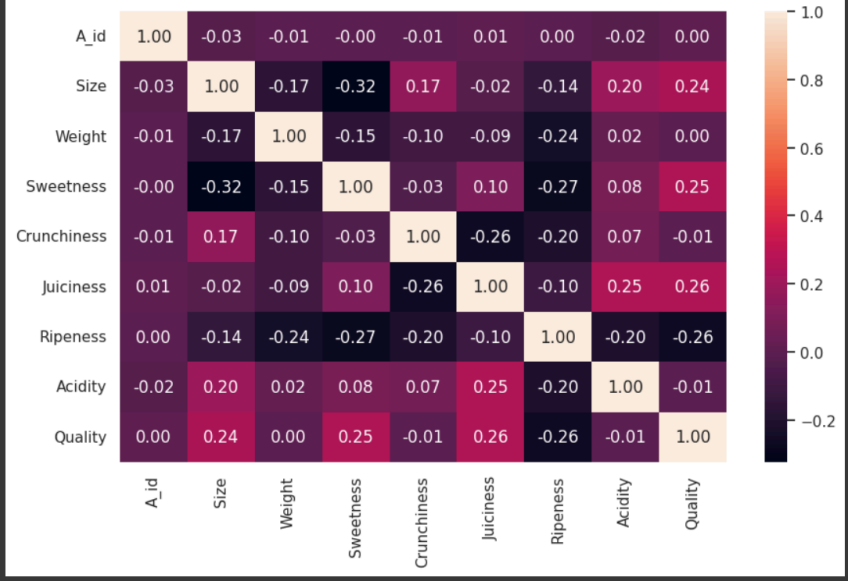
Correlation Matrix/Heatmap:

A correlation matrix helps visualize the linear relationships between numerical features. A heatmap makes it easy to spot highly correlated features, which can be useful for feature selection or understanding multicollinearity.

```
data.corr()
```

	A_id	Size	Weight	Sweetness	Crunchiness	Juiciness	Ripeness	Acidity	Quality
A_id	1.000000	-0.028722	-0.005954	-0.001819	-0.013516	0.005579	0.000252	-0.022019	0.004875
Size	-0.028722	1.000000	-0.166974	-0.324412	0.169820	-0.019437	-0.135910	0.196334	0.244522
Weight	-0.005954	-0.166974	1.000000	-0.152070	-0.095068	-0.092135	-0.243678	0.019696	0.000167
Sweetness	-0.001819	-0.324412	-0.152070	1.000000	-0.033874	0.095436	-0.273578	0.084710	0.250890
Crunchiness	-0.013516	0.169820	-0.095068	-0.033874	1.000000	-0.257884	-0.200391	0.071793	-0.012204
Juiciness	0.005579	-0.019437	-0.092135	0.095436	-0.257884	1.000000	-0.098975	0.248688	0.260135
Ripeness	0.000252	-0.135910	-0.243678	-0.273578	-0.200391	-0.098975	1.000000	-0.201941	-0.264687
Acidity	-0.022019	0.196334	0.019696	0.084710	0.071793	0.248688	-0.201941	1.000000	-0.007634
Quality	0.004875	0.244522	0.000167	0.250890	-0.012204	0.260135	-0.264687	-0.007634	1.000000

```
sns.heatmap(data.corr(), annot=True, fmt=".2f")
```



Milestone 4: Feature Engineering and Selection

Based on the EDA, this milestone involves creating new features or selecting the most relevant existing ones to improve model performance. For this project, given the dataset structure and the algorithms used in the notebook, explicit complex feature engineering might not be highlighted, but feature scaling is a form of transformation.

Feature Scaling:

It's crucial to scale numerical features, especially for algorithms sensitive to feature magnitudes (like Logistic Regression and some distance-based algorithms). The notebook uses StandardScaler.

Splitting Data and Feature Scaling

```
X=data.drop(['A_id', 'Quality'], axis=1)
y=data['Quality']

scaler=StandardScaler()
X=scaler.fit_transform(X)

X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.2, random_state=42)
```

Milestone 5: Model Building and Training

This milestone involves splitting the data, selecting appropriate machine learning models, and training them on the prepared dataset.

Model Selection and Training:

The project explores four common classification algorithms: Logistic Regression, Random Forest, and XGBoost, Decision Tree. Each model is initialized and trained on the X_train and y_train data.

Model Building

```
#Decision Tree
DT=DecisionTreeClassifier()
DT.fit(X_train, y_train)
DT_pred=DT.predict(X_test)
acc=accuracy_score(y_test,DT_pred)
print(acc)

#Random Forest
RF=RandomForestClassifier()
RF.fit(X_train, y_train)
RF_pred=RF.predict(X_test)
acc=accuracy_score(y_test,RF_pred)
print(acc)

#XGB
XGB=xgb.XGBClassifier()
XGB.fit(X_train, y_train)
XGB_pred=XGB.predict(X_test)
acc=accuracy_score(y_test,XGB_pred)
print(acc)

#Logistic Regression
LR=LogisticRegression()
LR.fit(X_train, y_train)
LR_pred=LR.predict(X_test)
acc=accuracy_score(y_test,LR_pred)
print(acc)
```

```
0.805
0.90625
0.915
0.75375
```

Milestone 6: Model Evaluation and Selection

After training, the models are evaluated using various metrics to assess their performance on unseen data.

Performance Testing and Hyperparameter Tuning:

Performance Testing and Hyperparameter Tuning

```
# 1. Performance Evaluation of All Models
print("----- Decision Tree Performance -----")
print(classification_report(y_test, DT_pred))
ConfusionMatrixDisplay.from_predictions(y_test, DT_pred).plot()

print("----- Random Forest Performance -----")
print(classification_report(y_test, RF_pred))
ConfusionMatrixDisplay.from_predictions(y_test, RF_pred).plot()

print("----- XGBoost Performance -----")
print(classification_report(y_test, XGB_pred))
ConfusionMatrixDisplay.from_predictions(y_test, XGB_pred).plot()

print("----- Logistic Regression Performance -----")
print(classification_report(y_test, LR_pred))
ConfusionMatrixDisplay.from_predictions(y_test, LR_pred).plot()

plt.show()
```

```
# 2. Hyperparameter Tuning using GridSearchCV

# Decision Tree Tuning
print("\n🔧 Grid Search for Decision Tree...")
param_grid_dt = {
    'max_depth': [3, 5, 10, None],
    'criterion': ['gini', 'entropy'],
    'min_samples_split': [2, 5, 10]
}

grid_dt = GridSearchCV(DecisionTreeClassifier(random_state=42), param_grid_dt, cv=3, n_jobs=-1)
grid_dt.fit(X_train, y_train)

print("Best Parameters for Decision Tree:", grid_dt.best_params_)
print("Best CV Score:", grid_dt.best_score_)

# Random Forest Tuning
print("\n🔧 Grid Search for Random Forest...")
param_grid_rf = {
    'n_estimators': [50, 100],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
}

grid_rf = GridSearchCV(RandomForestClassifier(random_state=42), param_grid_rf, cv=3, n_jobs=-1)
grid_rf.fit(X_train, y_train)

print("Best Parameters for Random Forest:", grid_rf.best_params_)
print("Best CV Score:", grid_rf.best_score_)
```

Making Predictions:

Predictions are made on the X_test dataset for each trained model.

Accuracy Score:

Accuracy measures the proportion of correctly classified instances.

Classification Report:

Provides precision, recall, f1-score, and support for each class.

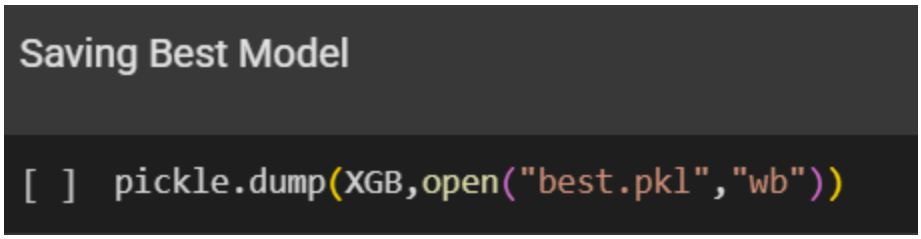
```
----- Decision Tree Performance -----
              precision    recall  f1-score   support

           0       0.81      0.81      0.81       401
           1       0.80      0.80      0.80       399

    accuracy                           0.81       800
   macro avg       0.80      0.80      0.80       800
weighted avg       0.81      0.81      0.81       800

----- Random Forest Performance -----
              precision    recall  f1-score   support

           0       0.90      0.91      0.91       401
           1       0.91      0.90      0.91       399

    accuracy                           0.91       800
   macro avg       0.91      0.91      0.91       800
weighted avg       0.91      0.91      0.91       800

----- XGBoost Performance -----
              precision    recall  f1-score   support

           0       0.92      0.91      0.91       401
           1       0.91      0.92      0.92       399

    accuracy                           0.92       800
   macro avg       0.92      0.92      0.91       800
weighted avg       0.92      0.92      0.91       800

----- Logistic Regression Performance -----
              precision    recall  f1-score   support

           0       0.75      0.76      0.75       401
           1       0.75      0.75      0.75       399

    accuracy                           0.75       800
   macro avg       0.75      0.75      0.75       800
weighted avg       0.75      0.75      0.75       800
```

Model Comparison and Selection:

Based on the accuracy scores and classification reports, the XGBoost appears to perform best. Therefore, it is selected as the best model.

Saving the Best Model:

The best performing model is saved using pickle for later deployment in the web application.



Milestone 7: Project Deployment (Web Application)

This milestone describes the process of deploying the trained machine learning model as a web application using Flask, as detailed in app1.py.

Overview of app1.py:

The app1.py file serves as the backend for the web application, handling requests, loading the trained model, making predictions, and rendering HTML templates.

- **Import Libraries:** Imports necessary libraries like numpy, pickle, Flask, request, and render_template.
- **Flask App Initialization:** Initializes the Flask application.
- **Model Loading:** Loads the best.pkl model using pickle. It includes error handling for potential ValueError during deserialization.
- **Routes:** Defines different URL endpoints for the web application.

Flask Routes:

1. **/ (Home Page):**
- **Function:** home()
- **Method:** GET
- **Purpose:** Renders the main index.html page, which typically serves as the landing page of the application.
2. **/predict (Prediction Input Page):**
- **Function:** predict()
- **Method:** POST, GET
- **Purpose:** Renders the inner-page.html template. This page likely contains a form where users can input apple characteristics.
3. **/submit (Prediction Result Page):**
- **Function:** submit()
- **Method:** POST, GET
- **Purpose:** This is the core logic for prediction.
- **Reads Inputs:** Retrieves input features from the HTML form using request.form.values(). These are converted to integers and then to a NumPy array.
- **Makes Predictions:** Calls model.predict() with the processed input features.
- **Determines Result:** Converts the numerical prediction (0 or 1) into a

human-readable string ("Bad" or "Good").

- **Renders Output:** Renders the output.html template, passing the prediction result to be displayed to the user.

Loading the Model:

The app1.py script opens best.pkl in binary read mode ('rb') and uses pickle.load() to load the pre-trained model into memory.

```
pickle_file_path = "best.pkl"
with open(pickle_file_path, 'rb') as file:
    try:
        model = pickle.load(file)
    except ValueError as e:
        # Error handling for incompatible dtype
        msg = "Incompatible dtype issue in the node array."
        raise ValueError(msg)
```

Input Handling and Prediction Logic:

When the /submit route is triggered, the submit() function executes:

```
@app.route('/submit', methods=["POST", "GET"])
def submit():
    # Read inputs from the form
    input_feature = [int(float(x)) for x in request.form.values()]
    input_feature = [np.array(input_feature)] # Convert to numpy array for model input

    # Make Predictions
    prediction = model.predict(input_feature)
    prediction = int(prediction) # Convert prediction to integer

    if prediction == 0:
        return render_template("output.html", result="Bad")
    else:
```

```
    return render_template("output.html", result="Good")
```

Rendering Templates:

Flask's render_template() function is used to display the appropriate HTML pages to the user. The output.html template receives the result variable to display the apple quality dynamically.

Instructions to Run the Web Application:

To run the Flask web application locally:

1. **Navigate to the project folder** in your terminal or Anaconda Prompt where app1.py and the templates/ folder are located.
2. **Ensure all dependencies are installed** (Flask, scikit-learn, numpy, pandas, xgboost, etc. - usually listed in a requirements.txt file).
   pip install -r requirements.txt
3. **Run the Flask application** using the command:
   python app1.py
4. **Open a web browser** and navigate to the local host URL provided in the console output (typically http://127.0.0.1:5000/).
5. **Interact with the application:**

- Click on the "Predict" button (or similar navigation) to go to the input form.
- Enter the apple characteristics (e.g., Size, Weight, Sweetness, etc.).
- Click "Submit" to get the quality prediction.

Conclusion

The Golden Harvesting: A Predictive Model for Apple Quality Assurance project successfully leverages machine learning to automate and improve the apple quality assessment process. By following a structured approach from problem definition and data preparation to model building, evaluation, and deployment, the project delivers a practical solution for agricultural stakeholders. The deployed web application provides an intuitive interface for real-time quality predictions, contributing to increased efficiency, reduced waste, and enhanced product quality in the apple supply chain. This model serves as a valuable tool for ensuring consistent high-quality produce reaches the market, benefiting both producers and consumers.