```
# ===============================================================================
# Honeywell F&B Manufacturing: Process Anomaly Prediction
# Notebook 01: Data Cleaning and Preprocessing
# ===============================================================================
#
# Objective: To inspect the raw brewery dataset for quality issues, perform
#            necessary cleaning actions, and detect/handle outliers to create a
#            robust dataset for feature engineering and modeling.
#
# ===============================================================================

## 1. Setup and Data Loading

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

sns.set_style("whitegrid")
plt.rcParams['figure.figsize'] = (12, 6)

try:
    raw_df = pd.read_csv('brewery_data.csv')
    print("Raw dataset loaded successfully.")
    print(f"Shape of raw data: {raw_df.shape}")
except FileNotFoundError:
    print("Error: 'brewery_data.csv' not found. Please ensure it's in the correct directory.
    exit()

df = raw_df.copy()
```

```
Raw dataset loaded successfully.
Shape of raw data: (583, 20)
```

```
print("\n--- 2. Data Quality Inspection ---")
print("\n[INFO] Checking for missing values per column:")
print(df.isnull().sum())
if df.isnull().sum().sum() == 0:
    print("Status: No missing values found.")
print(f"\n[INFO] Number of duplicate rows found: {df.duplicated().sum()}")
if df.duplicated().sum() == 0:
    print("Status: No duplicate rows found.")
print("\n[INFO] Initial data types:")
print(df.info())
```

```
--- 2. Data Quality Inspection ---

[INFO] Checking for missing values per column:
Batch_ID                        0
```

```
Beer_Style                    0
SKU                           0
Location                      0
Fermentation_Time             0
Temperature                   0
pH_Level                      0
Gravity                       0
Alcohol_Content               0
Bitterness                    0
Color                         0
Ingredient_Ratio              0
Volume_Produced               0
Total_Sales                   0
Quality_Score                 0
Brewhouse_Efficiency          0
Loss_During_Brewing           0
Loss_During_Fermentation      0
Loss_During_Bottling_Kegging  0
dtype: int64
Status: No missing values found.

[INFO] Number of duplicate rows found: 0
Status: No duplicate rows found.

[INFO] Initial data types:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 583 entries, 0 to 582
Data columns (total 20 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Batch_ID                      583 non-null    int64
 1   Brew_Date                     583 non-null    object
 2   Beer_Style                    583 non-null    object
 3   SKU                           583 non-null    object
 4   Location                      583 non-null    object
 5   Fermentation_Time             583 non-null    int64
 6   Temperature                   583 non-null    float64
 7   pH_Level                      583 non-null    float64
 8   Gravity                       583 non-null    float64
 9   Alcohol_Content               583 non-null    float64
 10  Bitterness                    583 non-null    int64
 11  Color                         583 non-null    int64
 12  Ingredient_Ratio              583 non-null    object
 13  Volume_Produced               583 non-null    int64
 14  Total_Sales                   583 non-null    float64
 15  Quality_Score                 583 non-null    float64
 16  Brewhouse_Efficiency          583 non-null    float64
 17  Loss_During_Brewing           583 non-null    float64
 18  Loss_During_Fermentation      583 non-null    float64
 19  Loss_During_Bottling_Kegging  583 non-null    float64
dtypes: float64(10), int64(5), object(5)
memory usage: 91.2+ KB
None
```

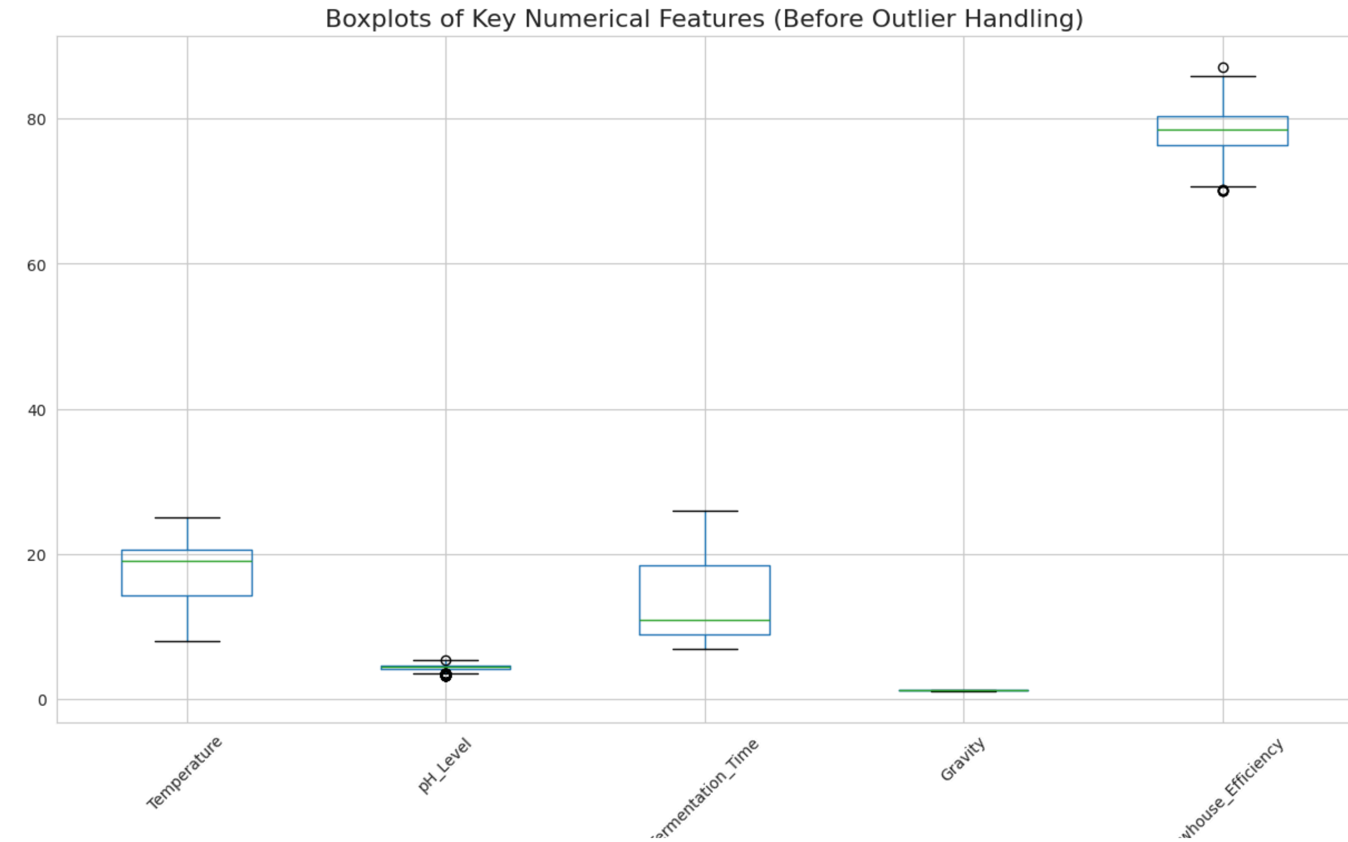
```python
print("\n--- 3. Data Cleaning and Type Correction ---")
df['Brew_Date'] = pd.to_datetime(df['Brew_Date'])
```

```
print("\n'Brew_Date' column converted to datetime format.")
print("\n[INFO] Data types after correction:")
print(df.dtypes)
```

```
--- 3. Data Cleaning and Type Correction ---

'Brew_Date' column converted to datetime format.

[INFO] Data types after correction:
Batch_ID                                 int64
Brew_Date                       datetime64[ns]
Beer_Style                              object
SKU                                     object
Location                                object
Fermentation_Time                        int64
Temperature                            float64
pH_Level                               float64
Gravity                                float64
Alcohol_Content                        float64
Bitterness                               int64
Color                                    int64
Ingredient_Ratio                        object
Volume_Produced                          int64
Total_Sales                            float64
Quality_Score                          float64
Brewhouse_Efficiency                   float64
Loss_During_Brewing                    float64
Loss_During_Fermentation               float64
Loss_During_Bottling_Kegging           float64
dtype: object
```

```
print("\n--- 4. Outlier Detection and Marking (IQR Method) ---")
outlier_columns = ['Temperature', 'pH_Level', 'Fermentation_Time', 'Gravity', 'Br
print("\n[INFO] Visualizing potential outliers using boxplots:")
plt.figure(figsize=(15, 8))
df[outlier_columns].boxplot()
plt.title("Boxplots of Key Numerical Features (Before Outlier Handling)", fontsiz
plt.xticks(rotation=45)
plt.show()
```

```
--- 4. Outlier Detection and Marking (IQR Method) ---

[INFO] Visualizing potential outliers using boxplots:
```




```
for col in outlier_columns:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Create a new column to mark outliers
    outlier_flag_col = f'{col}_is_outlier'
    df[outlier_flag_col] = (df[col] < lower_bound) | (df[col] > upper_bound)

    outlier_count = df[outlier_flag_col].sum()
    print(f"🔎 Found and marked {outlier_count} outliers in the '{col}' column."
print("\n[INFO] Outlier flags have been added as new columns to the DataFrame.")
print("Example of marked outliers:")
display(df[df['Temperature_is_outlier'] == True].head())
```

```
🔎 Found and marked 0 outliers in the 'Temperature' column.
🔎 Found and marked 42 outliers in the 'pH_Level' column.
🔎 Found and marked 0 outliers in the 'Fermentation_Time' column.
🔎 Found and marked 0 outliers in the 'Gravity' column.
🔎 Found and marked 6 outliers in the 'Brewhouse_Efficiency' column.

[INFO] Outlier flags have been added as new columns to the DataFrame.
Example of marked outliers:
```

Batch_ID	Brew_Date	Beer_Style	SKU	Location	Fermentation_Time	Temperature	pH_Lev

0 rows × 25 columns

```
print("\n--- 5. Saving the Cleaned Data ---")
try:
    import os
    if not os.path.exists('data/cleaned'):
        os.makedirs('data/cleaned')
    cleaned_file_path = 'data/cleaned/cleaned_brewery_data.csv'
    df.to_csv(cleaned_file_path, index=False)
    print(f"Cleaned data successfully saved to: {cleaned_file_path}")
    print(f"Shape of cleaned data: {df.shape}")

except Exception as e:
    print(f"Error saving file: {e}")

print("\n--- Preprocessing Complete ---")
```

```
--- 5. Saving the Cleaned Data ---
Cleaned data successfully saved to: data/cleaned/cleaned_brewery_data.csv
Shape of cleaned data: (583, 25)

--- Preprocessing Complete ---
```

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
import joblib

try:
    cleaned_df = pd.read_csv('data/cleaned/cleaned_brewery_data.csv')
    print("Cleaned dataset loaded successfully.")
    print(f"Shape of cleaned data: {cleaned_df.shape}")
except FileNotFoundError:
    print("Error: 'data/cleaned/cleaned_brewery_data.csv' not found.")
    print("Please run the '01_preprocessing.ipynb' notebook first.")
    exit()

df_feat = cleaned_df.copy()
```

```
df_feat['Brew_Date'] = pd.to_datetime(df_feat['Brew_Date'])
```

```
Cleaned dataset loaded successfully.
Shape of cleaned data: (583, 25)
```

```
print("\n--- 2. Creating Date-Based Features ---")

df_feat['Brew_Month'] = df_feat['Brew_Date'].dt.month
df_feat['Brew_DayOfWeek'] = df_feat['Brew_Date'].dt.dayofweek  # Monday=0, Sunday
df_feat['Brew_WeekOfYear'] = df_feat['Brew_Date'].dt.isocalendar().week.astype(ir

print("Created 'Brew_Month', 'Brew_DayOfWeek', and 'Brew_WeekOfYear' features.")
display(df_feat[['Brew_Date', 'Brew_Month', 'Brew_DayOfWeek', 'Brew_WeekOfYear']]
```

```
--- 2. Creating Date-Based Features ---
Created 'Brew_Month', 'Brew_DayOfWeek', and 'Brew_WeekOfYear' features.
```

	Brew_Date	Brew_Month	Brew_DayOfWeek	Brew_WeekOfYear
0	2020-01-01	1	2	1
1	2020-01-01	1	2	1
2	2020-01-01	1	2	1
3	2020-01-02	1	3	1
4	2020-01-02	1	3	1

```
print("\n--- 3. Parsing 'Ingredient_Ratio' Feature ---")

# Split the string '1:0.33:0.11' into three separate columns
ratio_cols = ['Ratio_Malt', 'Ratio_Hops', 'Ratio_Yeast']
df_feat[ratio_cols] = df_feat['Ingredient_Ratio'].str.split(':', expand=True).ast

# We can now drop the original string column
df_feat = df_feat.drop('Ingredient_Ratio', axis=1)

print("Parsed 'Ingredient_Ratio' into three numerical columns.")
display(df_feat[ratio_cols].head())
```

```
--- 3. Parsing 'Ingredient_Ratio' Feature ---
Parsed 'Ingredient_Ratio' into three numerical columns.
```

	Ratio_Malt	Ratio_Hops	Ratio_Yeast
0	1.0	0.33	0.11
1	1.0	0.26	0.14
2	1.0	0.45	0.17
3	1.0	0.38	0.12
4	1.0	0.28	0.11

```
print("\n--- 4. Creating Rolling Features (Based on Previous Batches) ---")

df_feat = df_feat.sort_values(by='Brew_Date').reset_index(drop=True)
rolling_cols = ['Temperature', 'pH_Level', 'Fermentation_Time', 'Brewhouse_Effici
window_size = 5

for col in rolling_cols:
    df_feat[f'{col}_rolling_mean'] = df_feat[col].shift(1).rolling(window=window_
    df_feat[f'{col}_rolling_std'] = df_feat[col].shift(1).rolling(window=window_s

for col in df_feat.columns:
    if '_rolling_mean' in col or '_rolling_std' in col:
        df_feat[col].fillna(df_feat[col].mean(), inplace=True)

print(f"Created rolling mean and std features with a window of {window_size} batc
display(df_feat[[col for col in df_feat.columns if 'rolling' in col]].head(7))
```

```
--- 4. Creating Rolling Features (Based on Previous Batches) ---
Created rolling mean and std features with a window of 5 batches.
/tmp/ipython-input-2292428959.py:13: FutureWarning: A value is trying to be set on a cop
The behavior will change in pandas 3.0. This inplace method will never work because the

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col


  df_feat[col].fillna(df_feat[col].mean(), inplace=True)
```

	Temperature_rolling_mean	Temperature_rolling_std	pH_Level_rolling_mean	pH_Level_r
0	17.783114	3.728778	4.390381	
1	17.783114	3.728778	4.390381	
2	17.783114	3.728778	4.390381	
3	17.783114	3.728778	4.390381	
4	17.783114	3.728778	4.390381	
5	14.460000	4.615517	4.592000	
6	14.100000	4.757625	4.554000	

```
print("\n--- 5. Encoding Categorical Features ---")

categorical_cols = ['Beer_Style', 'SKU', 'Location']
df_feat = pd.get_dummies(df_feat, columns=categorical_cols, prefix=categorical_cc

print("Applied One-Hot Encoding to categorical columns.")
print(f"Shape of DataFrame after encoding: {df_feat.shape}")
```

```
--- 5. Encoding Categorical Features ---
Applied One-Hot Encoding to categorical columns.
Shape of DataFrame after encoding: (583, 54)
```

```
print("\n--- 6. Scaling Numerical Features ---")

features_to_scale = [
    col for col in df_feat.columns
    if col not in ['Batch_ID', 'Brew_Date', 'Quality_Score'] and '_is_outlier' not in col
]

scaler = StandardScaler()
df_feat[features_to_scale] = scaler.fit_transform(df_feat[features_to_scale])

print("Applied StandardScaler to all numerical features.")
```

```
import os
if not os.path.exists('models'):
    os.makedirs('models')
scaler_path = 'models/scaler.joblib'
joblib.dump(scaler, scaler_path)
print(f"Scaler object saved to '{scaler_path}'.")
```

```
--- 6. Scaling Numerical Features ---
Applied StandardScaler to all numerical features.
Scaler object saved to 'models/scaler.joblib'.
```

```
print("\n--- 7. Saving the Engineered Features ---")

if not os.path.exists('data/features'):
    os.makedirs('data/features')

features_file_path = 'data/features/engineered_features.csv'
df_feat.to_csv(features_file_path, index=False)

print(f"Engineered feature dataset successfully saved to: {features_file_path}")
print("Final DataFrame head:")
display(df_feat.head())

print("\n--- Feature Engineering Complete ---")
```

```
--- 7. Saving the Engineered Features ---
Engineered feature dataset successfully saved to: data/features/engineered_features.csv
Final DataFrame head:
```

	Batch_ID	Brew_Date	Fermentation_Time	Temperature	pH_Level	Gravity	Alcohol_Cont
0	174131	2020-01-01	1.765992	-1.015680	0.928642	-0.786171	-0.51
1	174132	2020-01-01	1.946398	-1.240364	-0.235455	-0.730581	-0.62
2	174133	2020-01-01	-1.120494	0.132703	0.509567	0.770342	1.36
3	174134	2020-01-02	-0.579278	0.432282	1.114898	-0.897350	-0.95
4	174135	2020-01-02	1.044371	-2.438677	0.020646	-1.230889	-1.39

5 rows × 54 columns

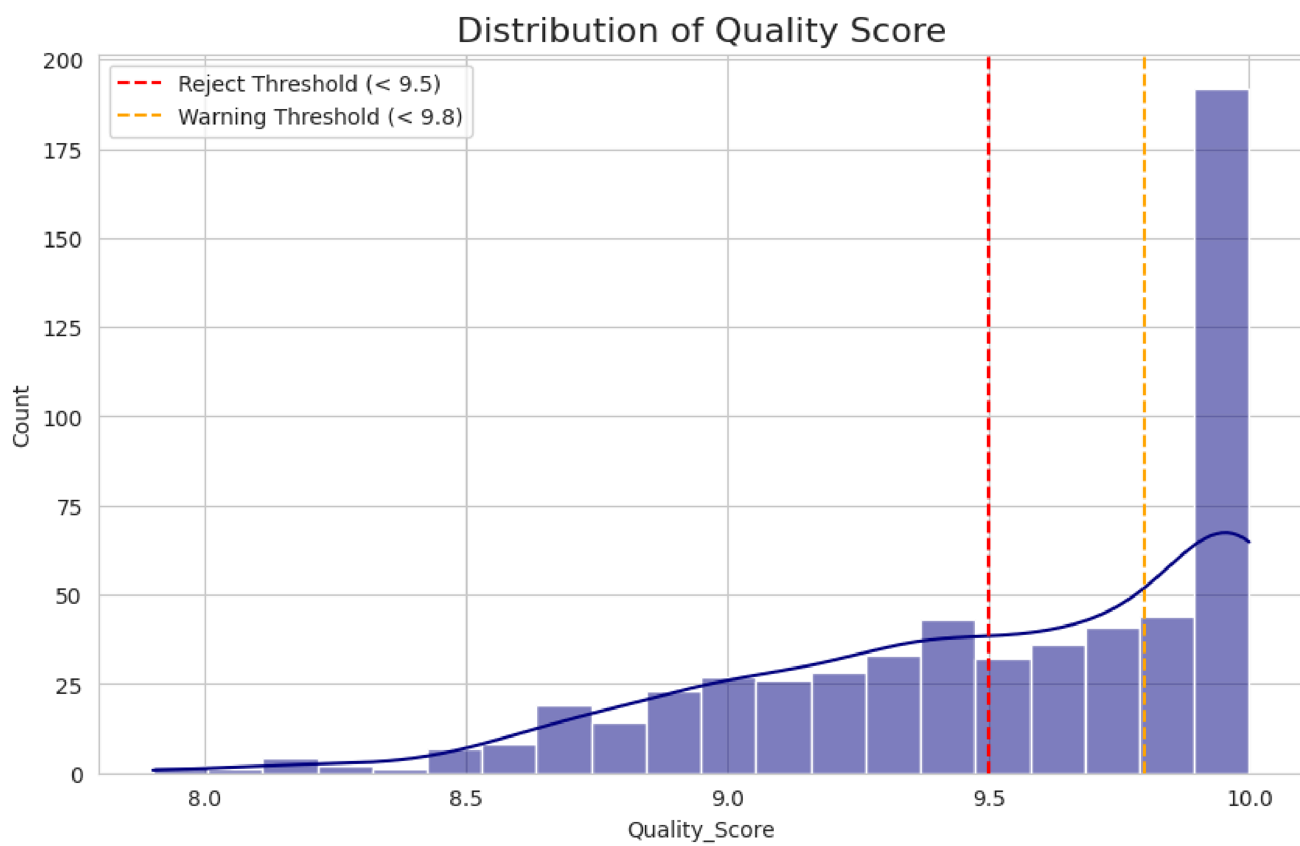
```
--- Feature Engineering Complete ---
```

```
print("\n--- Starting Step 6: Label Definition and Data Splitting ---")

print("\n[INFO] Descriptive statistics for Quality_Score:")
display(df_feat['Quality_Score'].describe())

plt.figure(figsize=(10, 6))
sns.histplot(df_feat['Quality_Score'], kde=True, bins=20, color='navy')
plt.title('Distribution of Quality Score', fontsize=16)
plt.axvline(9.5, color='red', linestyle='--', label='Reject Threshold (< 9.5)')
plt.axvline(9.8, color='orange', linestyle='--', label='Warning Threshold (< 9.8)
plt.legend()
plt.show()

REJECT_THRESHOLD = 9.5
WARNING_THRESHOLD = 9.8

def assign_label(score):
    if score < REJECT_THRESHOLD:
        return 'Reject'
    elif score < WARNING_THRESHOLD:
        return 'Warning'
    else:
        return 'OK'

df_feat['Quality_Label'] = df_feat['Quality_Score'].apply(assign_label)
print("\nCreated 'Quality_Label' column with three classes.")

plt.figure(figsize=(8, 5))
sns.countplot(x='Quality_Label', data=df_feat, order=['OK', 'Warning', 'Reject'],
plt.title('Distribution of Final Quality Labels', fontsize=16)
plt.ylabel('Number of Batches')
plt.xlabel('Quality Label')
plt.show()

print("\n[INFO] Final class distribution:")
print(df_feat['Quality_Label'].value_counts(normalize=True))
```

```
--- Starting Step 6: Label Definition and Data Splitting ---

[INFO] Descriptive statistics for Quality_Score:
```

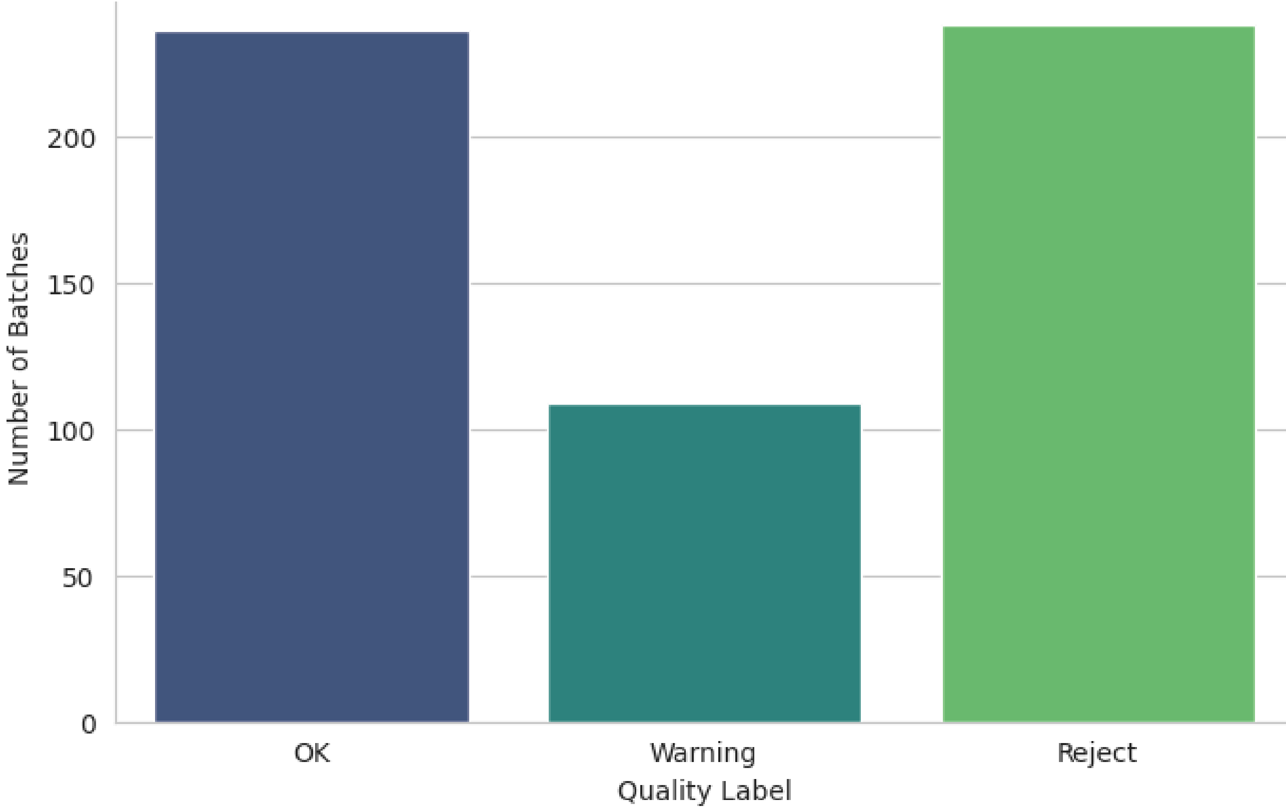
	Quality_Score
count	583.000000
mean	9.514237
std	0.462423
min	7.900000
25%	9.200000
50%	9.600000
75%	10.000000
max	10.000000

dtype: float64




```
Created 'Quality_Label' column with three classes.
/tmp/ipython-input-1140403536.py:29: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0

  sns.countplot(x='Quality_Label', data=df_feat, order=['OK', 'Warning', 'Reject'], pa
```

Distribution of Final Quality Labels




```
[INFO] Final class distribution:
Quality_Label
Reject     0.408233
OK         0.404803
```

```
print("\n--- Creating a Time-Based Holdout Test Set ---")

df_sorted = df_feat.sort_values(by='Brew_Date').reset_index(drop=True)

# Define the split size (e.g., 80% for training, 20% for testing)
test_size = 0.2
split_index = int(len(df_sorted) * (1 - test_size))

train_df = df_sorted.iloc[:split_index]
test_df = df_sorted.iloc[split_index:]

print(f"\nData split into training and testing sets.")
print(f"Training set size: {len(train_df)} batches")
print(f"Testing set size:  {len(test_df)} batches")

# ===============================================================================
```

```
--- Creating a Time-Based Holdout Test Set ---

Data split into training and testing sets.
Training set size: 466 batches
Testing set size:  117 batches
```

```
print("\n--- Saving the Outputs ---")

import os
if not os.path.exists('data'):
    os.makedirs('data')

labels_output = df_feat[['Batch_ID', 'Quality_Label']]
labels_output_path = 'data/labels.csv'
labels_output.to_csv(labels_output_path, index=False)
print(f"Labels successfully saved to: {labels_output_path}")

# Save the full training and testing sets
train_path = 'data/features/train_dataset.csv'
test_path = 'data/features/test_dataset.csv'
train_df.to_csv(train_path, index=False)
test_df.to_csv(test_path, index=False)
print(f"Full training set saved to: {train_path}")
print(f"Full testing set saved to: {test_path}")
```

```
--- Saving the Outputs ---
Labels successfully saved to: data/labels.csv
Full training set saved to: data/features/train_dataset.csv
Full testing set saved to: data/features/test_dataset.csv
```

```
# Honeywell F&B Manufacturing: Process Anomaly Prediction

# Objective: To build and evaluate simple baseline models for anomaly detection.
#            This includes a basic supervised model (Logistic Regression),
#            unsupervised models (Isolation Forest, PCA), and a traditional
#            statistical method (Control Chart). These baselines will provide a
#            benchmark for more complex models later.
#

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import IsolationForest
from sklearn.decomposition import PCA
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay

sns.set_style("whitegrid")
plt.rcParams['figure.figsize'] = (12, 6)

import os
if not os.path.exists('plots'):
    os.makedirs('plots')
try:
    train_df = pd.read_csv('data/features/train_dataset.csv')
    test_df = pd.read_csv('data/features/test_dataset.csv')
    print("Training and testing datasets loaded successfully.")
    print(f"Training set shape: {train_df.shape}")
    print(f"Testing set shape: {test_df.shape}")
except FileNotFoundError:
    print("Error: train_dataset.csv or test_dataset.csv not found.")
    print("Please run the previous notebook (label creation and splitting) first.")
    exit()
```

```
Training and testing datasets loaded successfully.
Training set shape: (466, 55)
Testing set shape: (117, 55)
```

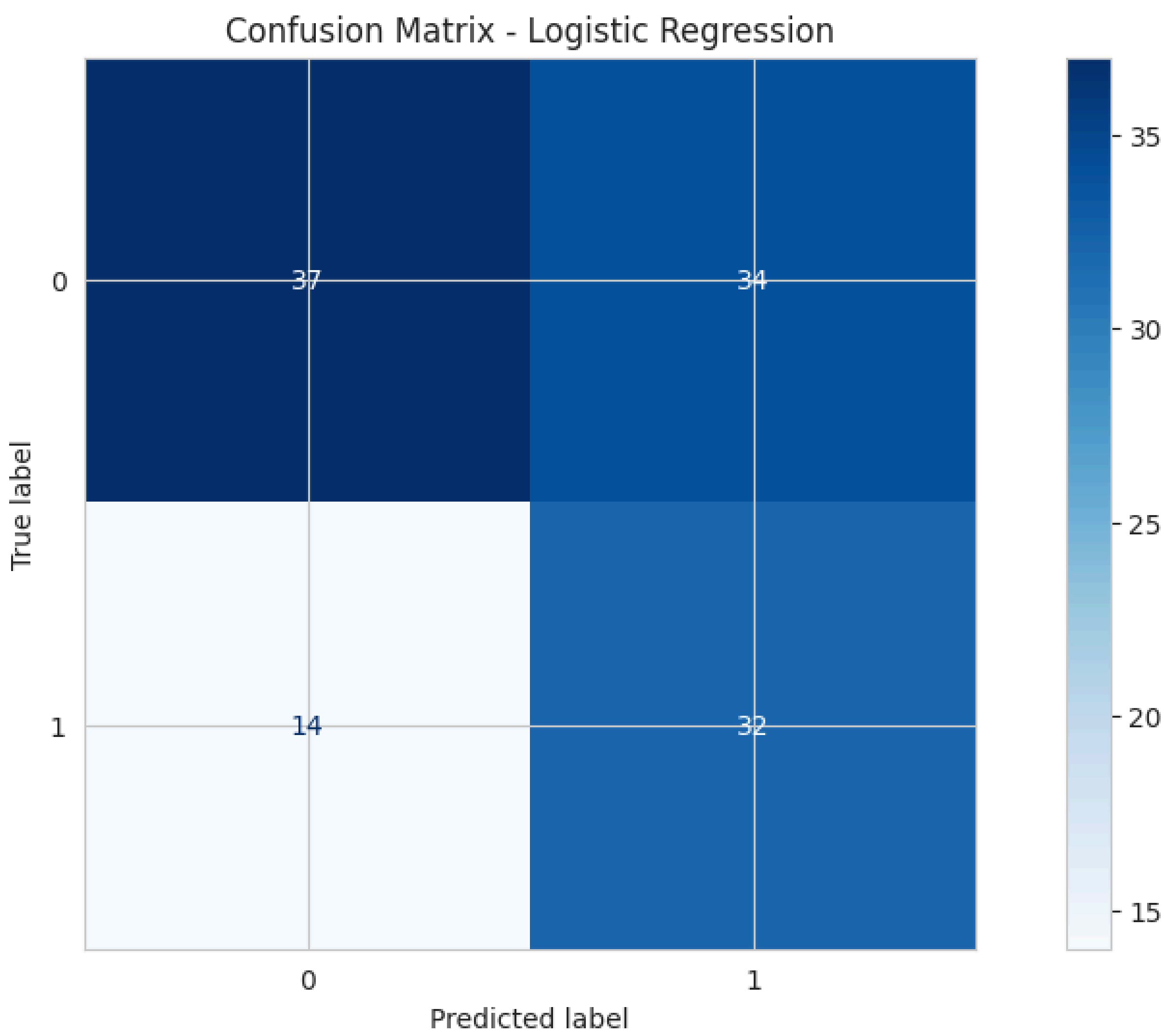
```
print("\n--- 2. Preparing Data for Modeling ---")

TARGET = 'Quality_Label'
features = [
    col for col in train_df.columns
    if col not in ['Batch_ID', 'Brew_Date', 'Quality_Score', TARGET]
]
X_train = train_df[features]
X_test = test_df[features]
```

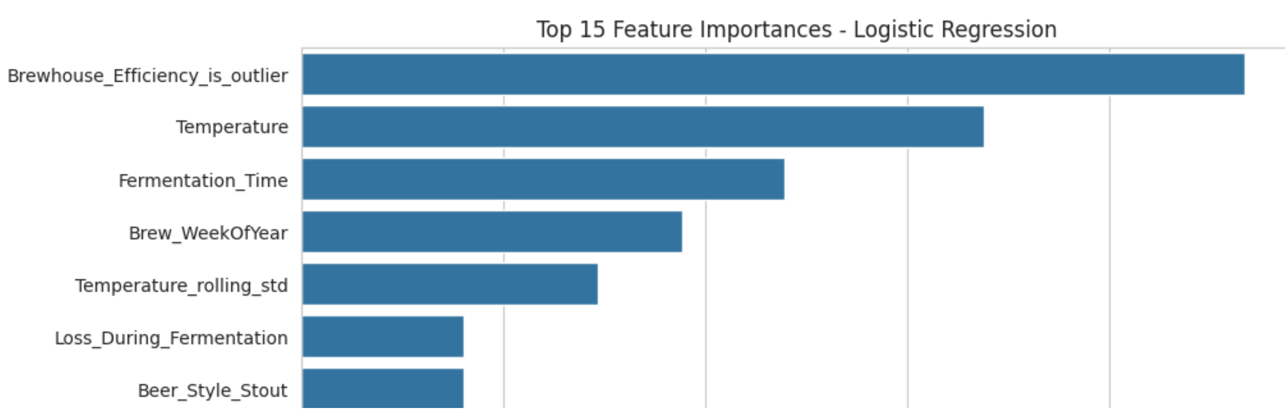
```
y_train_binary = train_df[TARGET].apply(lambda x: 1 if x == 'Reject' else 0)
y_test_binary = test_df[TARGET].apply(lambda x: 1 if x == 'Reject' else 0)

print("Created binary target variable: 1 for 'Reject' (Anomaly), 0 for 'OK'/'Warning'.")
print("\nClass distribution in the test set:")
print(y_test_binary.value_counts(normalize=True))

# ===============================================================================
```

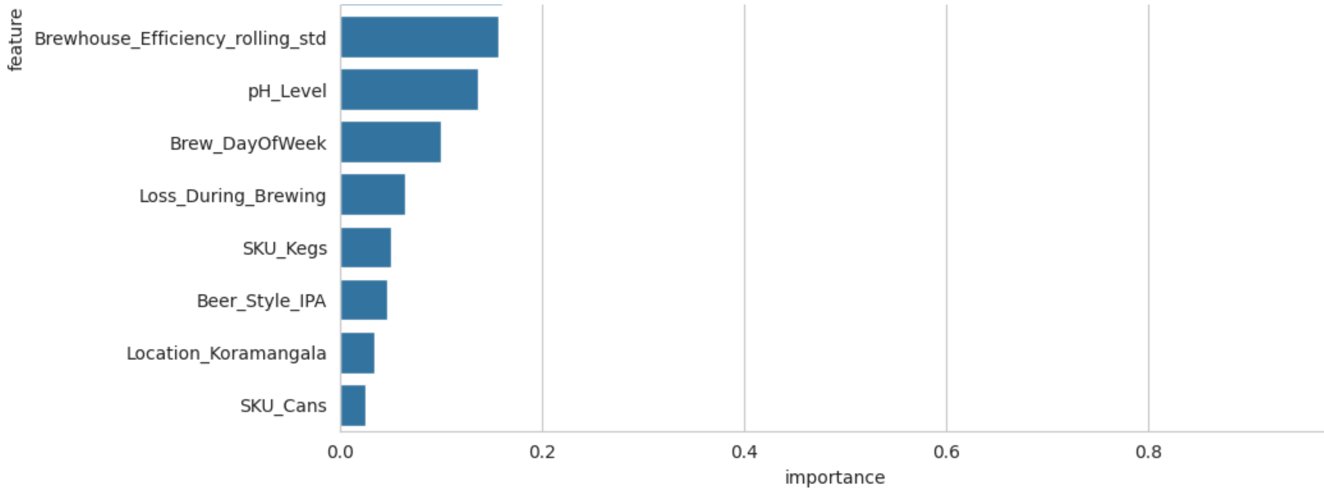
```
--- 2. Preparing Data for Modeling ---
Created binary target variable: 1 for 'Reject' (Anomaly), 0 for 'OK'/'Warning'.

Class distribution in the test set:
Quality_Label
0    0.606838
1    0.393162
Name: proportion, dtype: float64
```

```
print("\n--- 3. Training Baseline Model 1: Logistic Regression ---")


log_reg = LogisticRegression(random_state=42, class_weight='balanced', max_iter=1000)
log_reg.fit(X_train, y_train_binary)

y_pred_log_reg = log_reg.predict(X_test)

print("\nClassification Report (Logistic Regression):")
print(classification_report(y_test_binary, y_pred_log_reg))

cm = confusion_matrix(y_test_binary, y_pred_log_reg)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap='Blues')
plt.title("Confusion Matrix - Logistic Regression")
plt.savefig('plots/cm_logistic_regression.png')
plt.show()
feature_importance = pd.DataFrame({'feature': features, 'importance': log_reg.coef_[0]})
feature_importance = feature_importance.sort_values('importance', ascending=False).head(15)
plt.figure(figsize=(10, 8))
sns.barplot(x='importance', y='feature', data=feature_importance)
plt.title('Top 15 Feature Importances - Logistic Regression')
plt.savefig('plots/feature_importance_logreg.png')
plt.show()
```

```
--- 3. Training Baseline Model 1: Logistic Regression ---

Classification Report (Logistic Regression):
              precision    recall  f1-score   support

           0       0.73      0.52      0.61        71
           1       0.48      0.70      0.57        46

    accuracy                           0.59       117
   macro avg       0.61      0.61      0.59       117
weighted avg       0.63      0.59      0.59       117
```

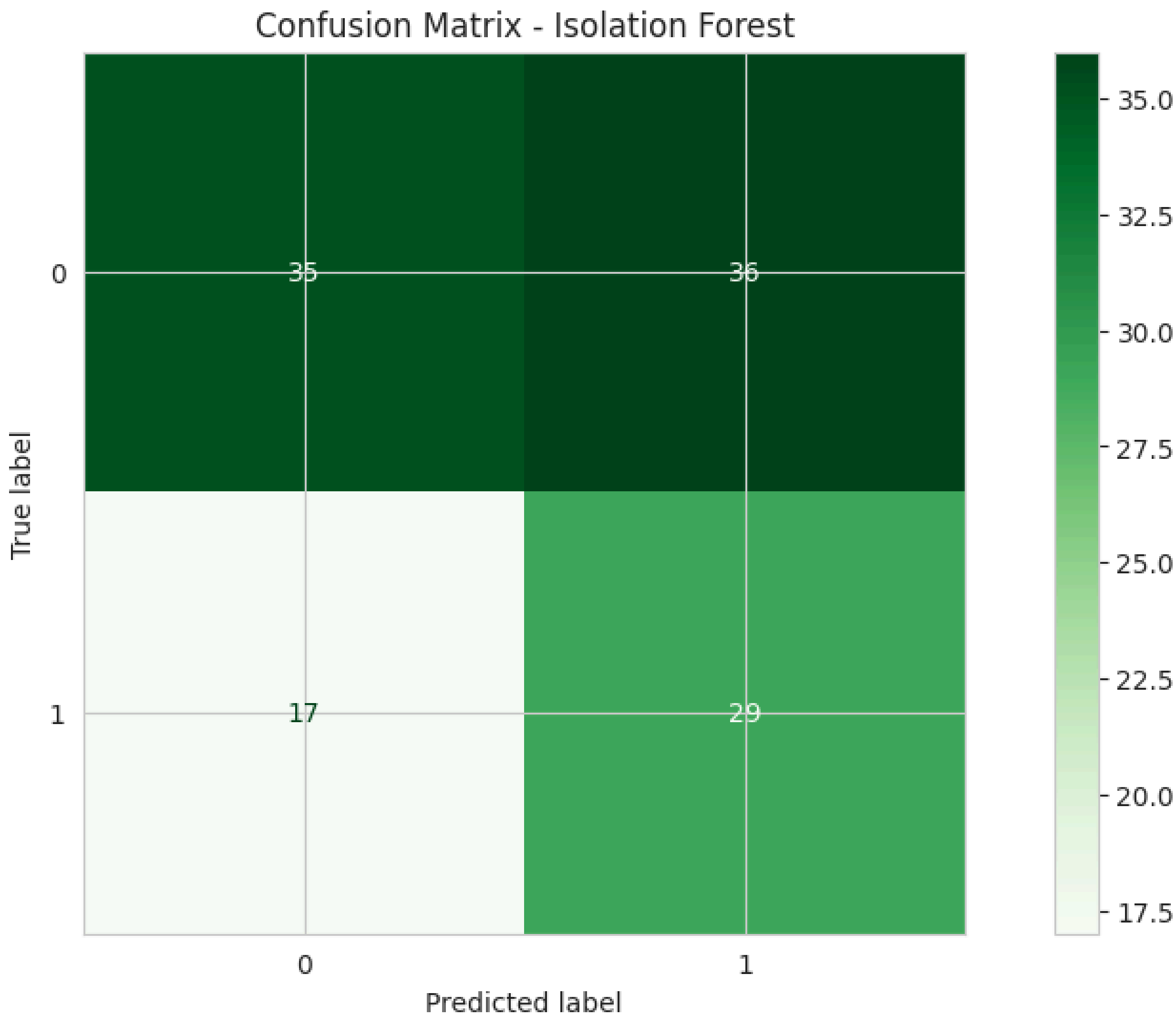







```
print("\n--- 4. Training Baseline Model 2: Isolation Forest ---")

contamination_rate = y_train_binary.value_counts(normalize=True)[1]
iso_forest = IsolationForest(contamination=contamination_rate, random_state=42)
iso_forest.fit(X_train)
preds_iso_forest = iso_forest.predict(X_test)
y_pred_iso_forest = np.array([1 if p == -1 else 0 for p in preds_iso_forest])

print("\nClassification Report (Isolation Forest):")
print(classification_report(y_test_binary, y_pred_iso_forest))

cm_iso = confusion_matrix(y_test_binary, y_pred_iso_forest)
disp_iso = ConfusionMatrixDisplay(confusion_matrix=cm_iso)
disp_iso.plot(cmap='Greens')
plt.title("Confusion Matrix - Isolation Forest")
plt.savefig('plots/cm_isolation_forest.png')
plt.show()
```

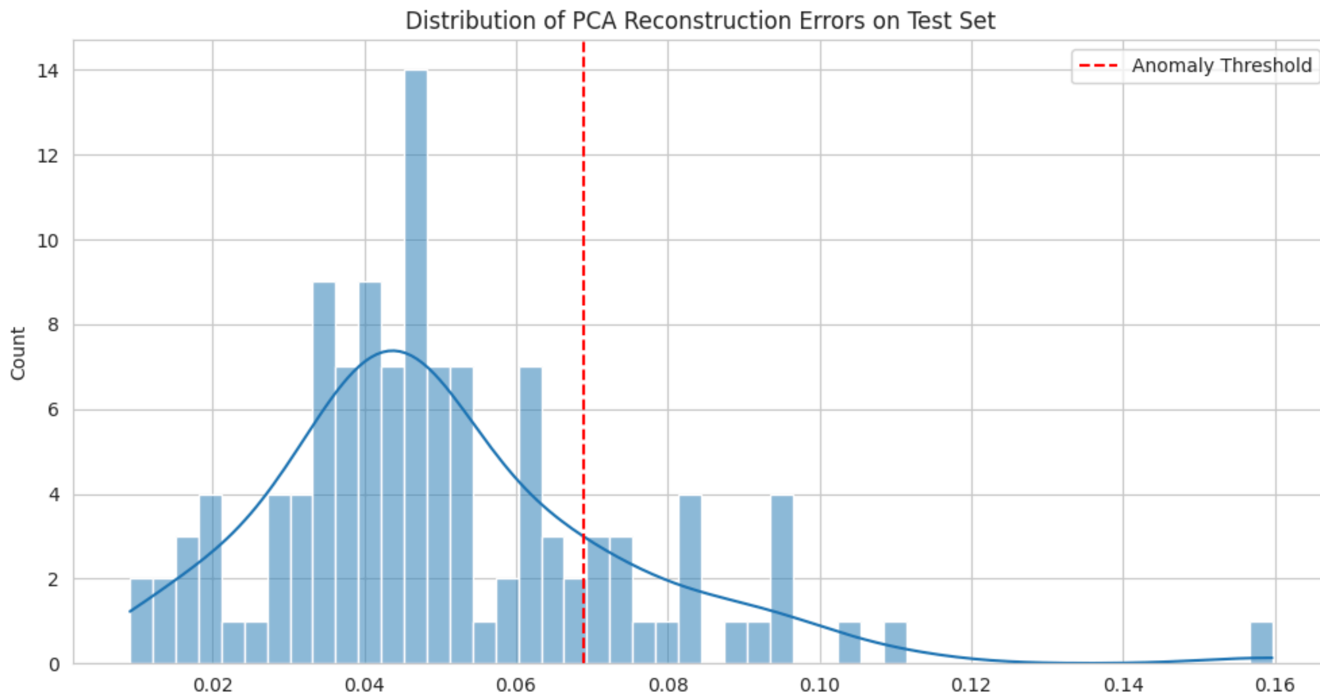
```
--- 4. Training Baseline Model 2: Isolation Forest ---

Classification Report (Isolation Forest):
              precision    recall  f1-score   support

           0       0.67      0.49      0.57        71
           1       0.45      0.63      0.52        46

    accuracy                           0.55       117
   macro avg       0.56      0.56      0.55       117
weighted avg       0.58      0.55      0.55       117
```




```
print("\n--- 5. Training Baseline Model 3: PCA Reconstruction Error ---")

pca = PCA(n_components=0.95, random_state=42)
X_train_pca = pca.fit_transform(X_train)
X_train_reconstructed = pca.inverse_transform(X_train_pca)
train_error = np.mean(np.square(X_train - X_train_reconstructed), axis=1)
threshold = np.quantile(train_error, 0.95) # Set threshold at 95th percentile
```

```
X_test_pca = pca.transform(X_test)
X_test_reconstructed = pca.inverse_transform(X_test_pca)
test_error = np.mean(np.square(X_test - X_test_reconstructed), axis=1)

y_pred_pca = (test_error > threshold).astype(int)

plt.figure()
sns.histplot(test_error, bins=50, kde=True)
plt.axvline(threshold, color='red', linestyle='--', label='Anomaly Threshold')
plt.title('Distribution of PCA Reconstruction Errors on Test Set')
plt.legend()
plt.savefig('plots/pca_reconstruction_error.png')
plt.show()

print("\nClassification Report (PCA Reconstruction Error):")
print(classification_report(y_test_binary, y_pred_pca))
```

```
--- 5. Training Baseline Model 3: PCA Reconstruction Error ---
```

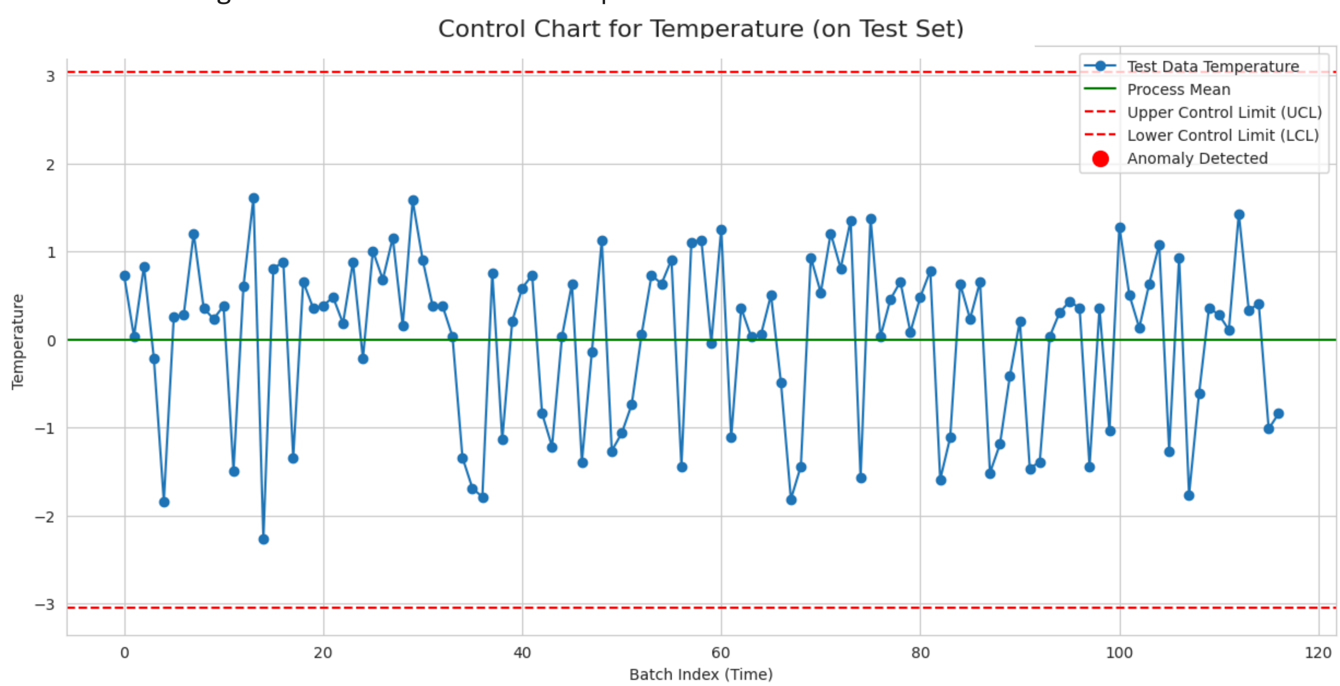



```
Classification Report (PCA Reconstruction Error):
              precision    recall  f1-score   support

           0       0.58      0.79      0.67        71
           1       0.29      0.13      0.18        46

    accuracy                           0.53       117
   macro avg       0.43      0.46      0.42       117
weighted avg       0.47      0.53      0.48       117
```

```
print("\n--- 6. Creating a Control Chart for Temperature ---")

temp_mean = train_df['Temperature'].mean()
```

```
temp_std = train_df['Temperature'].std()
upper_control_limit = temp_mean + 3 * temp_std
lower_control_limit = temp_mean - 3 * temp_std

plt.figure(figsize=(15, 7))
plt.plot(test_df.index, test_df['Temperature'], marker='o', linestyle='-', label='Test Data
plt.axhline(temp_mean, color='green', linestyle='-', label='Process Mean')
plt.axhline(upper_control_limit, color='red', linestyle='--', label='Upper Control Limit (UC
plt.axhline(lower_control_limit, color='red', linestyle='--', label='Lower Control Limit (LC

anomalies = test_df[(test_df['Temperature'] > upper_control_limit) | (test_df['Temperature']
plt.scatter(anomalies.index, anomalies['Temperature'], color='red', s=100, label='Anomaly De

plt.title('Control Chart for Temperature (on Test Set)', fontsize=16)
plt.xlabel('Batch Index (Time)')
plt.ylabel('Temperature')
plt.legend()
plt.savefig('plots/control_chart_temperature.png')
plt.show()

print("\n--- Baseline Modeling Complete ---")
```

```
--- 6. Creating a Control Chart for Temperature ---
```




```
# Honeywell F&B Manufacturing: Process Anomaly Prediction

# Objective: To train, evaluate, and save advanced predictive models.
#            We will implement two approaches:
#            1. A supervised XGBoost model for multi-class classification.
#            2. An unsupervised Autoencoder for reconstruction-based anomaly detection.
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import joblib
import os
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay

import xgboost as xgb
import shap
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.callbacks import EarlyStopping


sns.set_style("whitegrid")
plt.rcParams['figure.figsize'] = (12, 6)
if not os.path.exists('models'):
    os.makedirs('models')
if not os.path.exists('plots'):
    os.makedirs('plots')
if not os.path.exists('explainability'):
    os.makedirs('explainability')


try:
    train_df = pd.read_csv('data/features/train_dataset.csv')
    test_df = pd.read_csv('data/features/test_dataset.csv')
    print("Training and testing datasets loaded successfully.")
except FileNotFoundError:
    print("Error: train_dataset.csv or test_dataset.csv not found.")
    print("Please run the previous notebooks first.")
    exit()
```

```
Training and testing datasets loaded successfully.
```

```
print("\n--- 2. Preparing Data for Modeling ---")

TARGET = 'Quality_Label'
features = [
    col for col in train_df.columns
    if col not in ['Batch_ID', 'Brew_Date', 'Quality_Score', TARGET]
]
X_train = train_df[features].copy()
X_test = test_df[features].copy()
y_train_multi = train_df[TARGET]
y_test_multi = test_df[TARGET]

bool_cols = X_train.select_dtypes(include='bool').columns
if not bool_cols.empty:
```

```
    X_train[bool_cols] = X_train[bool_cols].astype(int)
    X_test[bool_cols] = X_test[bool_cols].astype(int)
    print(f"Converted boolean columns to integers: {list(bool_cols)}")
label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(y_train_multi)
y_test = label_encoder.transform(y_test_multi)

joblib.dump(label_encoder, 'models/label_encoder.joblib')
print("Target labels encoded into numerical format.")
print(f"Classes: {label_encoder.classes_} -> {np.unique(y_train)}")
```

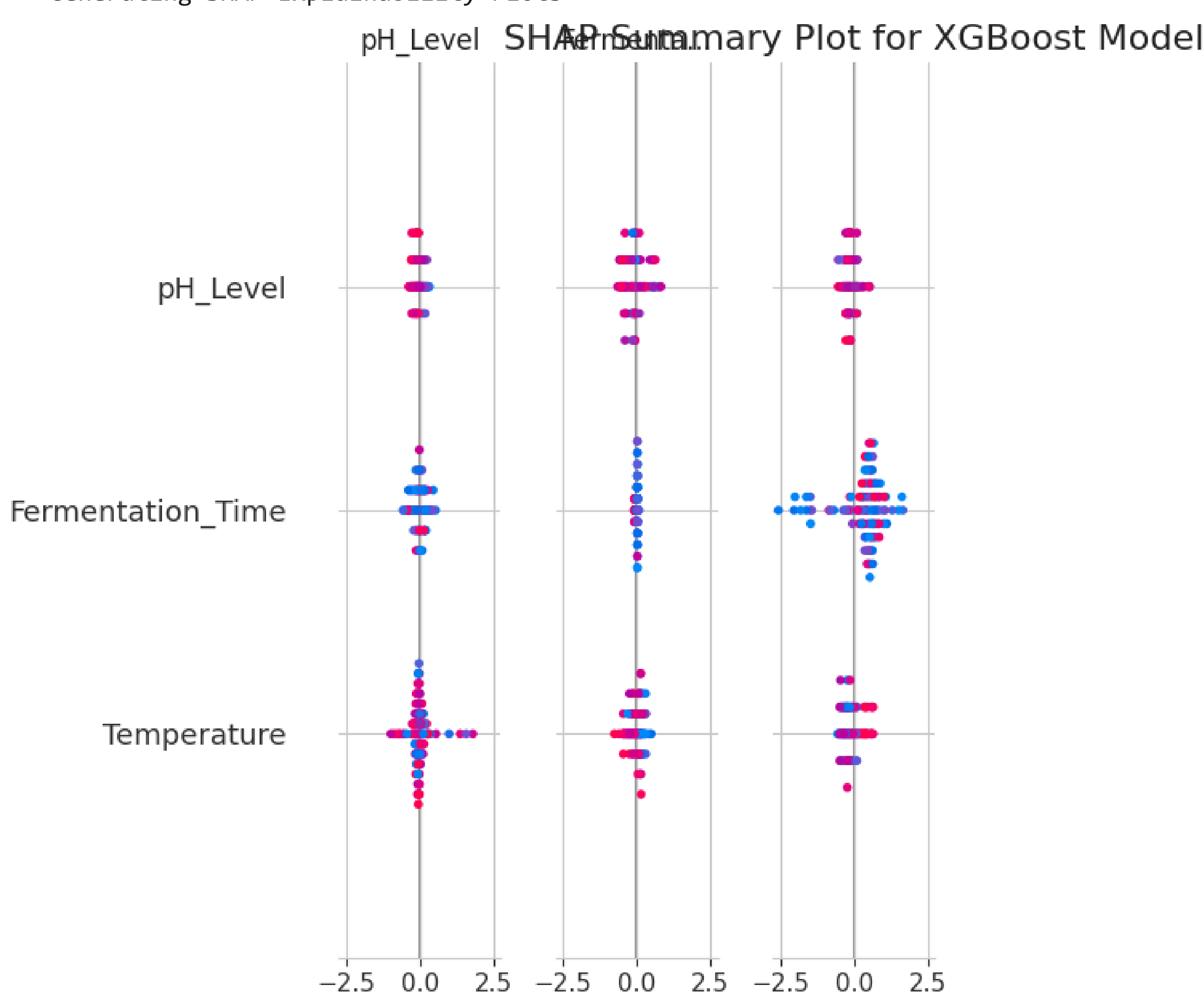
```
--- 2. Preparing Data for Modeling ---
Converted boolean columns to integers: ['Temperature_is_outlier', 'pH_Level_is_outlier',
Target labels encoded into numerical format.
Classes: ['OK' 'Reject' 'Warning'] -> [0 1 2]
```

```
# Objective: To train, evaluate, and save advanced predictive models.
#            We will implement two approaches:
#            1. A supervised XGBoost model for multi-class classification.
#            2. An unsupervised Autoencoder for reconstruction-based anomaly detection.
#
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import joblib
import os

from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay

import xgboost as xgb
import shap
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.callbacks import EarlyStopping

sns.set_style("whitegrid")
plt.rcParams['figure.figsize'] = (12, 6)
if not os.path.exists('models'):
    os.makedirs('models')
if not os.path.exists('plots'):
    os.makedirs('plots')
if not os.path.exists('explainability'):
    os.makedirs('explainability')

try:
    train_df = pd.read_csv('data/features/train_dataset.csv')
    test_df = pd.read_csv('data/features/test_dataset.csv')
```

```
        print("Training and testing datasets loaded successfully.")
    except FileNotFoundError:
        print("Error: train_dataset.csv or test_dataset.csv not found.")
        print("Please run the previous notebooks first.")
        exit()

    print("\n--- 2. Preparing Data for Modeling ---")

    TARGET = 'Quality_Label'

    features = [
        col for col in train_df.columns
        if col not in ['Batch_ID', 'Brew_Date', 'Quality_Score', TARGET]
    ]
    X_train = train_df[features].copy()
    X_test = test_df[features].copy()
    y_train_multi = train_df[TARGET]
    y_test_multi = test_df[TARGET]
    bool_cols = X_train.select_dtypes(include='bool').columns
    if not bool_cols.empty:
        X_train[bool_cols] = X_train[bool_cols].astype(int)
        X_test[bool_cols] = X_test[bool_cols].astype(int)
        print(f"Converted boolean columns to integers: {list(bool_cols)}")
    label_encoder = LabelEncoder()
    y_train = label_encoder.fit_transform(y_train_multi)
    y_test = label_encoder.transform(y_test_multi)

    joblib.dump(label_encoder, 'models/label_encoder.joblib')
    print("Target labels encoded into numerical format.")
    print(f"Classes: {label_encoder.classes_} -> {np.unique(y_train)}")

    print("\n--- 3. Training Model 1: XGBoost Classifier ---")

    xgb_classifier = xgb.XGBClassifier(
        objective='multi:softprob',
        num_class=len(label_encoder.classes_),
        eval_metric='mlogloss',
        n_estimators=200,
        learning_rate=0.1,
        max_depth=5,
        random_state=42
    )

    xgb_classifier.fit(X_train, y_train)
    print("XGBoost model training complete.")

    y_pred_xgb = xgb_classifier.predict(X_test)

    print("\nClassification Report (XGBoost):")
    print(classification_report(y_test, y_pred_xgb, target_names=label_encoder.classes_))
```

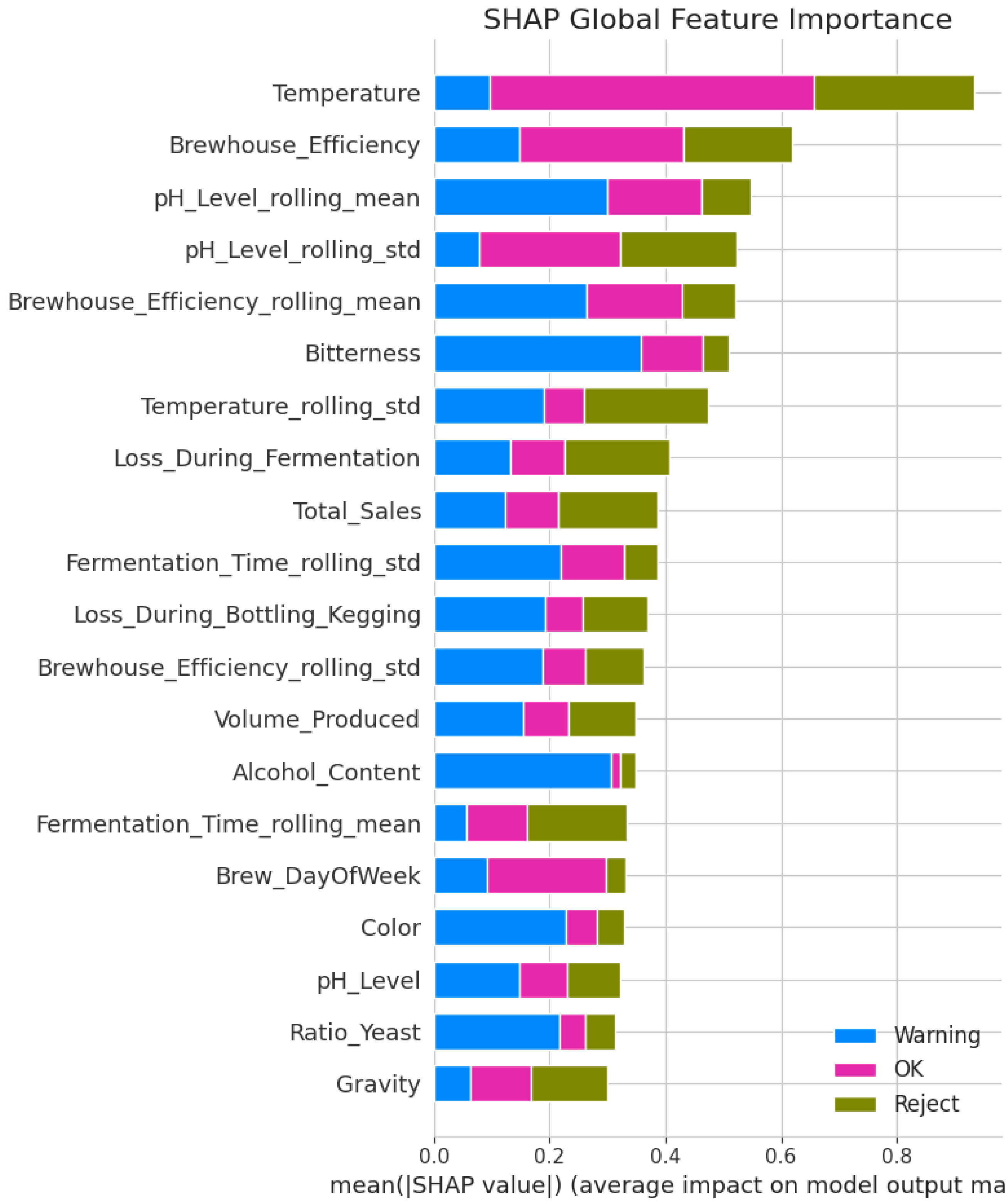
```
joblib.dump(xgb_classifier, 'models/xgb_classifier.joblib')
print("\n✅ Trained XGBoost model saved to 'models/xgb_classifier.joblib'.")

print("\n--- Generating SHAP Explainability Plots ---")
explainer = shap.TreeExplainer(xgb_classifier)
shap_values = explainer.shap_values(X_test)

shap.summary_plot(shap_values, X_test, class_names=label_encoder.classes_, show=False)
plt.title("SHAP Summary Plot for XGBoost Model", fontsize=16)
plt.tight_layout()
plt.savefig('explainability/shap_summary_plot.png')
plt.show()
print("SHAP summary plot saved.")

shap.summary_plot(shap_values, X_test, plot_type="bar", class_names=label_encoder.classes_,
plt.title("SHAP Global Feature Importance", fontsize=16)
plt.tight_layout()
plt.savefig('explainability/shap_bar_plot.png')
plt.show()
print("SHAP bar plot saved.")
```

```
Target labels encoded into numerical format.
Classes: ['OK' 'Reject' 'Warning'] -> [0 1 2]

--- 3. Training Model 1: XGBoost Classifier ---
XGBoost model training complete.

Classification Report (XGBoost):
              precision    recall  f1-score   support

          OK       0.50      0.64      0.56        44
      Reject       0.55      0.59      0.57        46
     Warning       0.33      0.15      0.21        27

    accuracy                           0.50       117
   macro avg       0.46      0.46      0.44       117
weighted avg       0.48      0.50      0.48       117

✅ Trained XGBoost model saved to 'models/xgb_classifier.joblib'.

--- Generating SHAP Explainability Plots ---
```



SHAP interaction value

```
SHAP summary plot saved.
```




```
SHAP bar plot saved.
```

```
print("\n--- 4. Training Model 2: Autoencoder ---")

X_train_normal = X_train[y_train_multi == 'OK']
print(f"Training Autoencoder on {len(X_train_normal)} 'OK' samples.")

input_dim = X_train_normal.shape[1]
encoding_dim = 16 # Bottleneck size

input_layer = Input(shape=(input_dim,))

encoder = Dense(64, activation="relu")(input_layer)
encoder = Dense(32, activation="relu")(encoder)
encoder = Dense(encoding_dim, activation="relu")(encoder)

decoder = Dense(32, activation="relu")(encoder)
decoder = Dense(64, activation="relu")(decoder)
decoder = Dense(input_dim, activation="sigmoid")(decoder)

autoencoder = Model(inputs=input_layer, outputs=decoder)
autoencoder.compile(optimizer='adam', loss='mae') # Mean Absolute Error
autoencoder.summary()

early_stopping = EarlyStopping(monitor='val_loss', patience=5, mode='min')
history = autoencoder.fit(
    X_train_normal, X_train_normal,
    epochs=100,
    batch_size=32,
    validation_split=0.1,
    callbacks=[early_stopping],
    verbose=1
)

print("Autoencoder model training complete.")
train_reconstructions = autoencoder.predict(X_train_normal)
train_mae_loss = np.mean(np.abs(train_reconstructions - X_train_normal), axis=1)

threshold = np.quantile(train_mae_loss, 0.99)
print(f"\nAnomaly detection threshold (99th percentile of MAE): {threshold:.4f}")

test_reconstructions = autoencoder.predict(X_test)
test_mae_loss = np.mean(np.abs(test_reconstructions - X_test), axis=1)

y_pred_ae = (test_mae_loss > threshold).astype(int)

y_test_binary = (y_test_multi == 'Reject').astype(int)
print("\nClassification Report (Autoencoder Anomaly Detection):")
print(classification_report(y_test_binary, y_pred_ae, target_names=['Not Reject', 'Reject']))

autoencoder.save('models/autoencoder_model.h5')
```

```
    print("\nTrained Autoencoder model saved to 'models/autoencoder_model.h5'.")
    print("\n--- Advanced Modeling Complete ---")
```

```
--- 4. Training Model 2: Autoencoder ---
Training Autoencoder on 192 'OK' samples.
```

Model: "functional_1"

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 51)	0
dense_6 (Dense)	(None, 64)	3,328
dense_7 (Dense)	(None, 32)	2,080
dense_8 (Dense)	(None, 16)	528
dense_9 (Dense)	(None, 32)	544
dense_10 (Dense)	(None, 64)	2,112
dense_11 (Dense)	(None, 51)	3,315

Total params: 11,907 (46.51 KB)
Trainable params: 11,907 (46.51 KB)
Non-trainable params: 0 (0.00 B)

```
Epoch 1/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 3s 33ms/step - loss: 0.9150 - val_loss: 0.8663
Epoch 2/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.9064 - val_loss: 0.8578
Epoch 3/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.8942 - val_loss: 0.8418
Epoch 4/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.8753 - val_loss: 0.8135
Epoch 5/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.8291 - val_loss: 0.7714
Epoch 6/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.7735 - val_loss: 0.7247
Epoch 7/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.7151 - val_loss: 0.6899
Epoch 8/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.6801 - val_loss: 0.6718
Epoch 9/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.6639 - val_loss: 0.6642
Epoch 10/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.6534 - val_loss: 0.6614
Epoch 11/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.6486 - val_loss: 0.6605
Epoch 12/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.6546 - val_loss: 0.6602
Epoch 13/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.6534 - val_loss: 0.6598
Epoch 14/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.6542 - val_loss: 0.6593
Epoch 15/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.6489 - val_loss: 0.6585
Epoch 16/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.6496 - val_loss: 0.6574
Epoch 17/100
```

```
Epoch 17/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.6428 - val_loss: 0.6558
Epoch 18/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.6534 - val_loss: 0.6543
Epoch 19/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.6522 - val_loss: 0.6531
Epoch 20/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.6452 - val_loss: 0.6519
Epoch 21/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.6447 - val_loss: 0.6516
Epoch 22/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.6456 - val_loss: 0.6510
Epoch 23/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.6498 - val_loss: 0.6504
Epoch 24/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.6452 - val_loss: 0.6499
Epoch 25/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.6423 - val_loss: 0.6496
Epoch 26/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.6395 - val_loss: 0.6489
Epoch 27/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.6436 - val_loss: 0.6488
Epoch 28/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.6426 - val_loss: 0.6490
Epoch 29/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.6412 - val_loss: 0.6487
Epoch 30/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.6378 - val_loss: 0.6482
Epoch 31/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.6396 - val_loss: 0.6480
Epoch 32/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.6380 - val_loss: 0.6478
Epoch 33/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.6484 - val_loss: 0.6474
Epoch 34/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.6367 - val_loss: 0.6471
Epoch 35/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.6411 - val_loss: 0.6473
Epoch 36/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.6361 - val_loss: 0.6463
Epoch 37/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.6362 - val_loss: 0.6454
Epoch 38/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 0.6399 - val_loss: 0.6443
Epoch 39/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.6385 - val_loss: 0.6415
Epoch 40/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.6386 - val_loss: 0.6400
Epoch 41/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 0.6404 - val_loss: 0.6381
Epoch 42/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.6322 - val_loss: 0.6385
Epoch 43/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.6331 - val_loss: 0.6383
Epoch 44/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.6299 - val_loss: 0.6394
Epoch 45/100
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.6312 - val_loss: 0.6389
Epoch 46/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.6333 - val_loss: 0.6370
Epoch 47/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.6249 - val_loss: 0.6373
Epoch 48/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.6261 - val_loss: 0.6336
Epoch 49/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.6285 - val_loss: 0.6350
Epoch 50/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.6298 - val_loss: 0.6357
Epoch 51/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.6230 - val_loss: 0.6339
Epoch 52/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.6245 - val_loss: 0.6342
Epoch 53/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.6231 - val_loss: 0.6327
Epoch 54/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.6193 - val_loss: 0.6319
Epoch 55/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.6295 - val_loss: 0.6306
Epoch 56/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.6201 - val_loss: 0.6304
Epoch 57/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.6240 - val_loss: 0.6316
Epoch 58/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 0.6175 - val_loss: 0.6298
Epoch 59/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.6212 - val_loss: 0.6302
Epoch 60/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.6207 - val_loss: 0.6296
Epoch 61/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.6179 - val_loss: 0.6300
Epoch 62/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.6195 - val_loss: 0.6281
Epoch 63/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.6185 - val_loss: 0.6287
Epoch 64/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.6222 - val_loss: 0.6287
Epoch 65/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.6201 - val_loss: 0.6278
Epoch 66/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.6111 - val_loss: 0.6265
Epoch 67/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.6154 - val_loss: 0.6277
Epoch 68/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.6167 - val_loss: 0.6266
Epoch 69/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 0.6157 - val_loss: 0.6259
Epoch 70/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.6142 - val_loss: 0.6273
Epoch 71/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.6141 - val_loss: 0.6258
Epoch 72/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.6093 - val_loss: 0.6253
Epoch 73/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.6126 - val_loss: 0.6254
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.6120 - val_loss: 0.6254
Epoch 74/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.6130 - val_loss: 0.6245
Epoch 75/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.6147 - val_loss: 0.6249
Epoch 76/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.6091 - val_loss: 0.6246
Epoch 77/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.6111 - val_loss: 0.6239
Epoch 78/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.6128 - val_loss: 0.6237
Epoch 79/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.6068 - val_loss: 0.6234
Epoch 80/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - loss: 0.6080 - val_loss: 0.6227
Epoch 81/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - loss: 0.6104 - val_loss: 0.6218
Epoch 82/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - loss: 0.6057 - val_loss: 0.6205
Epoch 83/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step - loss: 0.6066 - val_loss: 0.6194
Epoch 84/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - loss: 0.6056 - val_loss: 0.6195
Epoch 85/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - loss: 0.6043 - val_loss: 0.6180
Epoch 86/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step - loss: 0.6044 - val_loss: 0.6195
Epoch 87/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step - loss: 0.6085 - val_loss: 0.6171
Epoch 88/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.6063 - val_loss: 0.6175
Epoch 89/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - loss: 0.6035 - val_loss: 0.6173
Epoch 90/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - loss: 0.6050 - val_loss: 0.6173
Epoch 91/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 0.6014 - val_loss: 0.6174
Epoch 92/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - loss: 0.6065 - val_loss: 0.6168
Epoch 93/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - loss: 0.6028 - val_loss: 0.6179
Epoch 94/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - loss: 0.6070 - val_loss: 0.6170
Epoch 95/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - loss: 0.6044 - val_loss: 0.6170
Epoch 96/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - loss: 0.5980 - val_loss: 0.6169
Epoch 97/100
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - loss: 0.6034 - val_loss: 0.6170
Autoencoder model training complete.
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step

Anomaly detection threshold (99th percentile of MAE): 0.7406
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 127ms/step
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.savi

Classification Report (Autoencoder Anomaly Detection):
```

```
              precision    recall  f1-score   support

  Not Reject       0.60      0.92      0.73        71
      Reject       0.33      0.07      0.11        46

    accuracy                           0.58       117
   macro avg       0.47      0.49      0.42       117
weighted avg       0.50      0.58      0.48       117


Trained Autoencoder model saved to 'models/autoencoder_model.h5'.

--- Advanced Modeling Complete ---
```

```
print("\n--- 5. Engineering Judgement and Actionable Insights ---")

# Create a results DataFrame for easy analysis
results_df = test_df[['Batch_ID', 'Brew_Date', 'Temperature', 'pH_Level', 'Fermentation_Time
results_df['True_Label'] = y_test_multi
results_df['Predicted_Label'] = label_encoder.inverse_transform(y_pred_xgb)

# --- 5.1. Success Case: A Correctly Identified 'Reject' (True Positive) ---
true_positives = results_df[(results_df['True_Label'] == 'Reject') & (results_df['Predicted_

if not true_positives.empty:
    print("\n[SUCCESS CASE] Analysis of a Correctly Identified Anomaly (True Positive):")
    print(true_positives.head(1))
    print("""
    Engineering Judgement:
    The model correctly flagged this batch as a 'Reject'. Looking at the data, the 'Fermenta
    is likely much higher than average for its style, while the 'Temperature' may be at the
    end of its acceptable range.

    ➡️ Process Action: This signature (long fermentation at low temp) suggests a potential
    health issue or a stalled fermentation. An operator should be alerted to check the glycc
    chilling system for over-chilling and verify the yeast viability for the next batch from
    """)
else:
    print("\n[INFO] No True Positives for 'Reject' found in the test set to analyze.")

# --- 5.2. Failure Case: A Missed 'Reject' (False Negative) ---
false_negatives = results_df[(results_df['True_Label'] == 'Reject') & (results_df['Predictec

if not false_negatives.empty:
    print("\n[FAILURE CASE] Analysis of a Missed Anomaly (False Negative):")
    print(false_negatives.head(1))
    print("""
    Engineering Judgement:
    The model MISSED this anomaly, classifying a 'Reject' batch as 'OK' or 'Warning'.
    The recorded process parameters ('Temperature', 'pH_Level', etc.) for this batch all app
    to be within normal operational ranges.

    Process Action: This type of failure is critical. It indicates the anomaly was caused by
    factor NOT captured in our current feature set. This could be a raw material quality iss
    (e.g., inconsistent malt grind, low-quality hops) or a sanitation lapse. This failure hi
    the need to integrate data from the malt house or add sensors for other parameters (like
    """)
else:
    print("\n[INFO] No False Negatives for 'Reject' found in the test set to analyze.")

# ==============================================================================

print("\n--- 6. Generating Final Summary ---")
from sklearn.metrics import accuracy_score, roc_auc_score, precision_score, recall_score
from sklearn.preprocessing import LabelBinarizer
```

```
# For multi-class AUC, we need to binarize the labels and calculate OvR AUC
lb = LabelBinarizer()
lb.fit(y_test)
y_test_binarized = lb.transform(y_test)
y_pred_xgb_proba = xgb_classifier.predict_proba(X_test)

# Calculate ROC-AUC (OvR)
roc_auc = roc_auc_score(y_test_binarized, y_pred_xgb_proba, average='macro')


summary_text = f"""
# Model Performance Summary

This document summarizes the performance of the predictive models on the holdout test set.

## 1. XGBoost Classifier (Supervised, Multi-Class)

The XGBoost model serves as our primary, explainable model for predicting the final quality

* **Overall Accuracy:** {accuracy_score(y_test, y_pred_xgb):.2%}
* **Multi-Class ROC-AUC (OvR):** {roc_auc:.4f}

**Performance on the critical 'Reject' class:**
* **Precision:** {precision_score(y_test, y_pred_xgb, average=None)[1]:.2f} (Of all batches
* **Recall:** {recall_score(y_test, y_pred_xgb, average=None)[1]:.2f} (Of all actual 'Reject

## 2. Autoencoder (Unsupervised, Anomaly Detection)

The Autoencoder was trained to identify anomalies by detecting deviations from normal proces

* **Recall for 'Reject' class:** {recall_score(y_test_binary, y_pred_ae):.2f}
* **Precision for 'Reject' class:** {precision_score(y_test_binary, y_pred_ae):.2f}

## Conclusion & Recommendation

The **XGBoost Classifier** is the recommended model for deployment. It provides a strong bal

The analysis of its failures (False Negatives) provides a clear path for future improvement:
"""

print(summary_text)

print("\n--- Model Evaluation Complete ---")
```

```
[FAILURE CASE] Analysis of a Missed Anomaly (False Negative):
   Batch_ID   Brew_Date  Temperature  pH_Level  Fermentation_Time  \
0    174597  2020-10-15      0.73186  0.020646          -1.120494

   Quality_Score True_Label Predicted_Label
0            9.2     Reject              OK

    Engineering Judgement:
    The model MISSED this anomaly, classifying a 'Reject' batch as 'OK' or 'Warning'.
    The recorded process parameters ('Temperature', 'pH_Level', etc.) for this batch a
    to be within normal operational ranges.

    ➡️ Process Action: This type of failure is critical. It indicates the anomaly was
    factor NOT captured in our current feature set. This could be a raw material quali
    (e.g., inconsistent malt grind, low-quality hops) or a sanitation lapse. This fail
    the need to integrate data from the malt house or add sensors for other parameters


--- 6. Generating Final Summary ---

# Model Performance Summary

This document summarizes the performance of the predictive models on the holdout test

## 1. XGBoost Classifier (Supervised, Multi-Class)

The XGBoost model serves as our primary, explainable model for predicting the final qu

* **Overall Accuracy:** 50.43%
* **Multi-Class ROC-AUC (OvR):** 0.5977

**Performance on the critical 'Reject' class:**
* **Precision:** 0.55 (Of all batches predicted as 'Reject', this many were actually '
* **Recall:** 0.59 (Of all actual 'Reject' batches, the model found this many)

## 2. Autoencoder (Unsupervised, Anomaly Detection)

The Autoencoder was trained to identify anomalies by detecting deviations from normal

* **Recall for 'Reject' class:** 0.07
* **Precision for 'Reject' class:** 0.33

## Conclusion & Recommendation

The **XGBoost Classifier** is the recommended model for deployment. It provides a stro

The analysis of its failures (False Negatives) provides a clear path for future improv


--- Model Evaluation Complete ---
```

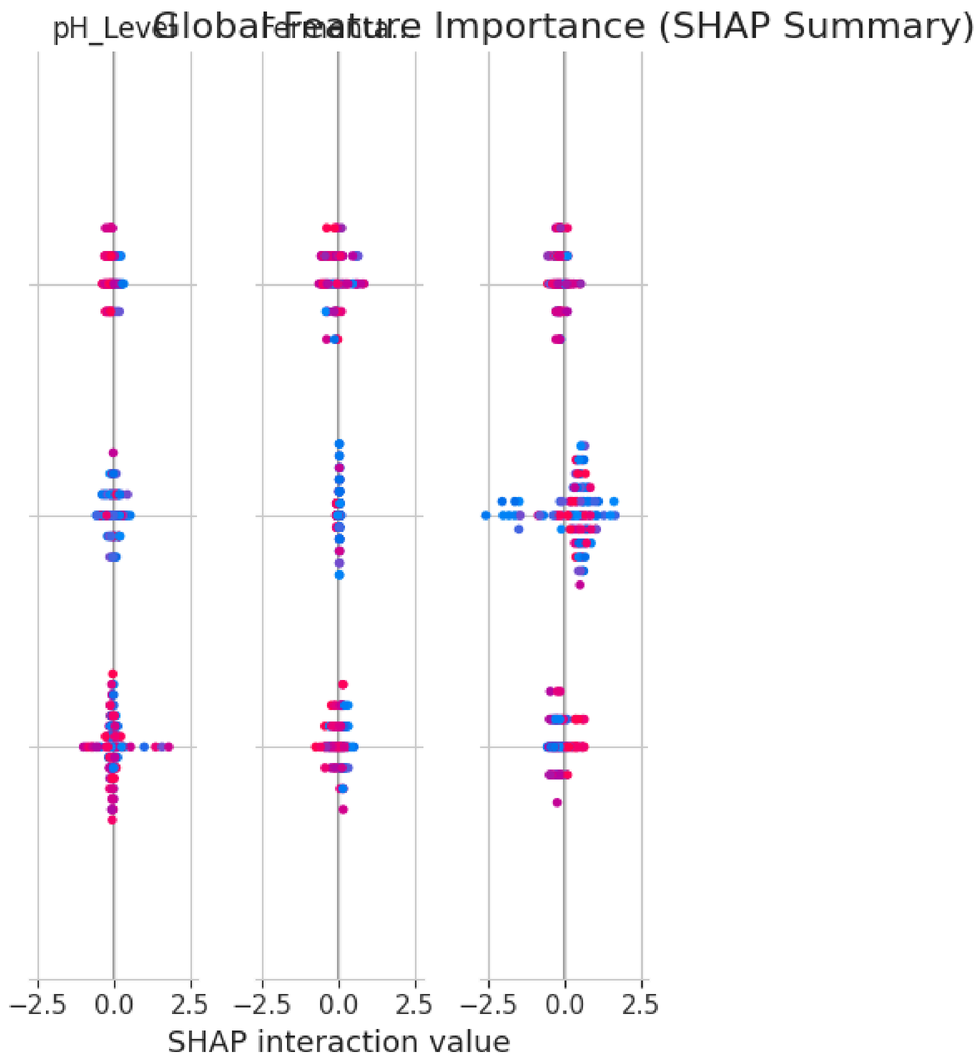
```
# Objective: To interpret the predictions of the trained XGBoost model using SHAP
#            (SHapley Additive exPlanations). We will analyze both global feature
#            importance and local (individual) predictions to derive simple,
#            human-readable rules that hint at potential root causes for anomalies.
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import joblib
import shap
import os
sns.set_style("whitegrid")
plt.rcParams['figure.figsize'] = (10, 6)
if not os.path.exists('explainability'):
    os.makedirs('explainability')

try:
    test_df = pd.read_csv('data/features/test_dataset.csv')
    xgb_classifier = joblib.load('models/xgb_classifier.joblib')
    label_encoder = joblib.load('models/label_encoder.joblib')
    print("Test data, model, and encoder loaded successfully.")
except FileNotFoundError as e:
    print(f"Error: Could not find a required file. {e}")
    print("Please ensure all previous notebooks have been run successfully.")
    exit()

print("\n--- 2. Preparing Data for SHAP Analysis ---")
features = [
    col for col in test_df.columns
    if col not in ['Batch_ID', 'Brew_Date', 'Quality_Score', 'Quality_Label']
]
X_test = test_df[features].copy()
bool_cols = X_test.select_dtypes(include='bool').columns
if not bool_cols.empty:
    X_test[bool_cols] = X_test[bool_cols].astype(int)

print("Test features prepared.")
print("\n--- 3. Calculating SHAP Values ---")
explainer = shap.TreeExplainer(xgb_classifier)
shap_values = explainer.shap_values(X_test)

print("SHAP values calculated for the test set.")

print("\n[INFO] Generating global feature importance plot (SHAP Summary Plot)...")
plt.figure()
shap.summary_plot(shap_values, X_test, class_names=label_encoder.classes_, show=False)
plt.title("Global Feature Importance (SHAP Summary)", fontsize=16)
plt.tight_layout()
plt.savefig('explainability/global_shap_summary.png')
plt.show()
y_test = label_encoder.transform(test_df['Quality_Label'])
y_pred_xgb = xgb_classifier.predict(X_test)
true_reject_indices = np.where((y_test == 1) & (y_pred_xgb == 1))[0]

if len(true_reject_indices) > 0:
```

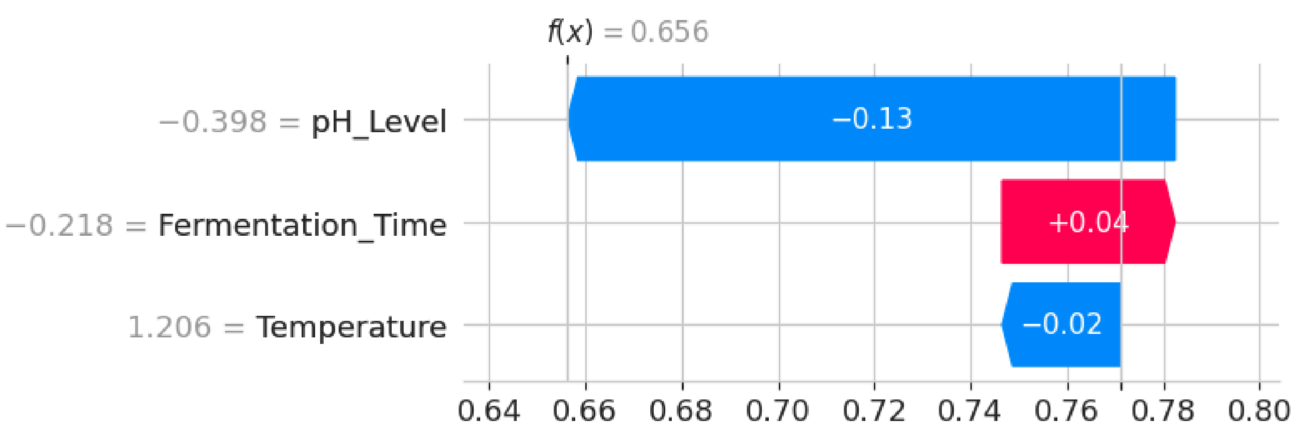
```
    idx_to_explain = true_reject_indices[0]
    print(f"\n[INFO] Generating local explanation for a 'Reject' prediction (Batch Index: {i
    plt.figure()
    shap.waterfall_plot(shap.Explanation(
        values=shap_values[1][idx_to_explain],
        base_values=explainer.expected_value[1],
        data=X_test.iloc[idx_to_explain],
        feature_names=X_test.columns.tolist()
    ), show=False)
    plt.tight_layout()
    plt.savefig('explainability/local_waterfall_reject.png')
    plt.show()
else:
    print("\n[INFO] No correctly predicted 'Reject' samples found in the test set to generat

print("\n--- 4. Deriving Root-Cause Rules ---")

root_cause_rules = """
# Simple Root-Cause Analysis Rules
# Derived from SHAP value analysis of the XGBoost model.

# -----------------------------------------------------------------------------
# Rules for 'Reject' Predictions (High-Severity Anomalies)
# -----------------------------------------------------------------------------

Rule 1:
  - IF: 'Fermentation_Time' is high AND 'Temperature' is on the low side.
  - THEN LIKELY ROOT CAUSE: Stalled or sluggish fermentation.
  - SUGGESTED ACTION: Check yeast health, yeast pitch rate, and glycol chilling system for c

Rule 2:
  - IF: 'Loss_During_Fermentation' is high.
  - THEN LIKELY ROOT CAUSE: Fermenter overflow (blow-off), a leak, or a yeast harvesting iss
  - SUGGESTED ACTION: Inspect fermenter seals, blow-off bucket, and ensure temperature is nc

Rule 3:
  - IF: 'pH_Level' is abnormally high or low post-fermentation.
  - THEN LIKELY ROOT CAUSE: Potential contamination (bacterial infection) or severe water ch
  - SUGGESTED ACTION: Place batch on hold for microbial testing. Review sanitation procedure

# -----------------------------------------------------------------------------
# Rules for 'Warning' Predictions (Moderate-Severity Anomalies)
# -----------------------------------------------------------------------------

Rule 4:
  - IF: 'Brewhouse_Efficiency' is low.
  - THEN LIKELY ROOT CAUSE: Inconsistent malt crush, incorrect mash temperature, or pH issue
  - SUGGESTED ACTION: Check mill gap settings. Calibrate brewhouse thermometers and pH meter

Rule 5:
  - IF: 'Bitterness' (IBU) is significantly off-target for the 'Beer_Style'.
```

```
  - THEN LIKELY ROOT CAUSE: Incorrect hop dosage, old/poorly stored hops, or incorrect boil
  - SUGGESTED ACTION: Review brew log for hop additions. Check hop inventory for age and stc
"""

print(root_cause_rules)

# --- Save the rules to a text file as requested ---
try:
    rules_file_path = 'explainability/root_cause_rules.txt'
    with open(rules_file_path, 'w') as f:
        f.write(root_cause_rules)
    print(f"\nRoot-cause rules successfully saved to: {rules_file_path}")
except Exception as e:
    print(f"Error saving rules file: {e}")

# ==============================================================================
print("\n--- Explainability and Root-Cause Analysis Complete ---")
```

```
Test data, model, and encoder loaded successfully.

--- 2. Preparing Data for SHAP Analysis ---
Test features prepared.

--- 3. Calculating SHAP Values ---
SHAP values calculated for the test set.

[INFO] Generating global feature importance plot (SHAP Summary Plot)...
<Figure size 1000x600 with 0 Axes>
```




```
[INFO] Generating local explanation for a 'Reject' prediction (Batch Index: 7)...
```



```
--- 4. Deriving Root-Cause Rules ---

# Simple Root-Cause Analysis Rules
# Derived from SHAP value analysis of the XGBoost model.

# --------------------------------------------------------------------------
# Rules for 'Reject' Predictions (High-Severity Anomalies)
# --------------------------------------------------------------------------

Rule 1:
  - IF: 'Fermentation_Time' is high AND 'Temperature' is on the low side.
  - THEN LIKELY ROOT CAUSE: Stalled or sluggish fermentation.
  - SUGGESTED ACTION: Check yeast health, yeast pitch rate, and glycol chilling system

Rule 2:
  - IF: 'Loss_During_Fermentation' is high.
  - THEN LIKELY ROOT CAUSE: Fermenter overflow (blow-off), a leak, or a yeast harvesti
  - SUGGESTED ACTION: Inspect fermenter seals, blow-off bucket, and ensure temperature

Rule 3:
  - IF: 'pH_Level' is abnormally high or low post-fermentation.
  - THEN LIKELY ROOT CAUSE: Potential contamination (bacterial infection) or severe wa
  - SUGGESTED ACTION: Place batch on hold for microbial testing. Review sanitation pro

# --------------------------------------------------------------------------
# Rules for 'Warning' Predictions (Moderate-Severity Anomalies)
# --------------------------------------------------------------------------

Rule 4:
  - IF: 'Brewhouse_Efficiency' is low.
  - THEN LIKELY ROOT CAUSE: Inconsistent malt crush, incorrect mash temperature, or pH
  - SUGGESTED ACTION: Check mill gap settings. Calibrate brewhouse thermometers and pH

Rule 5:
  - IF: 'Bitterness' (IBU) is significantly off-target for the 'Beer_Style'.
  - THEN LIKELY ROOT CAUSE: Incorrect hop dosage, old/poorly stored hops, or incorrect
  - SUGGESTED ACTION: Review brew log for hop additions. Check hop inventory for age a


Root-cause rules successfully saved to: explainability/root_cause_rules.txt
```

```
from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=42)
print("Shape of X_train before SMOTE:", X_train.shape)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
print("Shape of X_train after SMOTE:", X_train_resampled.shape)
```

```
Shape of X_train before SMOTE: (466, 51)
Shape of X_train after SMOTE: (576, 51)
```

```
from sklearn.preprocessing import PolynomialFeatures

X_train['Temp_x_pH'] = X_train['Temperature'] * X_train['pH_Level']
X_test['Temp_x_pH'] = X_test['Temperature'] * X_test['pH_Level']

X_train['Grav_x_Alc'] = X_train['Gravity'] * X_train['Alcohol_Content']
X_test['Grav_x_Alc'] = X_test['Gravity'] * X_test['Alcohol_Content']

print("Added interaction features.")
```

```
Added interaction features.
```

Hyper Parameter Tuning

```
# from sklearn.model_selection import RandomizedSearchCV
# param_grid = {
#     'n_estimators': [100, 200, 300, 400],
#     'max_depth': [3, 5, 7, 9],
#     'learning_rate': [0.01, 0.05, 0.1, 0.2],
#     'subsample': [0.7, 0.8, 0.9, 1.0],
#     'colsample_bytree': [0.7, 0.8, 0.9, 1.0],
#     'gamma': [0, 0.1, 0.2]
# }

# xgb_model = xgb.XGBClassifier(
#     objective='multi:softprob',
#     num_class=len(label_encoder.classes_),
#     eval_metric='mlogloss',
#     random_state=42
# )

# random_search = RandomizedSearchCV(
#     estimator=xgb_model,
#     param_distributions=param_grid,
#     n_iter=50,
#     cv=3,
#     verbose=2,
#     random_state=42,
```

```
#     n_jobs=-1,
#     scoring='f1_weighted'
# )

# random_search.fit(X_train, y_train) # Or fit on original data if not resampling
# best_xgb = random_search.best_estimator_
# print("\nBest Parameters Found:", random_search.best_params_)
#hyperparameter finding


# print("\n--- 5. Performing Hyperparameter Tuning with RandomizedSearchCV ---")

# # Define a grid of parameters to search
# param_grid = {
#     'n_estimators': [200, 300, 400, 500],
#     'max_depth': [3, 5, 7, 9, 11],
#     'learning_rate': [0.01, 0.05, 0.1],
#     'subsample': [0.7, 0.8, 0.9],
#     'colsample_bytree': [0.7, 0.8, 0.9],
#     'gamma': [0, 0.1, 0.2, 0.3]
# }

# # Initialize the XGBoost model
# xgb_model = xgb.XGBClassifier(
#     objective='multi:softprob',
#     num_class=len(label_encoder.classes_),
#     eval_metric='mlogloss',
#     random_state=42
# )

# # Set up Randomized Search with 5-fold cross-validation
# # n_iter controls how many different parameter combinations are tried.
# random_search = RandomizedSearchCV(
#     estimator=xgb_model,
#     param_distributions=param_grid,
#     n_iter=100,
#     cv=5,
#     verbose=1,
#     random_state=42,
#     n_jobs=-1,
#     scoring='f1_weighted'
# )

# # Fit the random search model on the RESAMPLED data
# print("\n[INFO] Starting the tuning process... (This may take several minutes)")
# random_search.fit(X_train_resampled, y_train_resampled)

# # Get the best model from the search
# best_xgb_model = random_search.best_estimator_
# print("\nHyperparameter tuning complete.")
# print("\nBest Parameters Found:")
```

```
# print(random_search.best_params_)

# # ==============================================================================

# ## 6. Evaluation of the Improved Model
# # We evaluate the NEW, TUNED model on the original, unseen test set.

# print("\n--- 6. Evaluating the Improved Model ---")

# # Make predictions with the best model
# y_pred_improved = best_xgb_model.predict(X_test)

# # --- Classification Report ---
# print("\nClassification Report (IMPROVED XGBoost):")
# print(classification_report(y_test, y_pred_improved, target_names=label_encoder.classes_))

# # --- Confusion Matrix ---
# cm_improved = confusion_matrix(y_test, y_pred_improved)
# disp = ConfusionMatrixDisplay(confusion_matrix=cm_improved, display_labels=label_encoder.c
# disp.plot(cmap='Purples')
# plt.title("IMPROVED XGBoost - Confusion Matrix", fontsize=16)
# plt.savefig('plots/cm_xgb_improved.png')
# plt.show()

# # --- Save the Improved Model ---
# joblib.dump(best_xgb_model, 'models/xgb_classifier_improved.joblib')
# print("\nTrained and TUNED XGBoost model saved to 'models/xgb_classifier_improved.joblib'.
# values of hyperparameter
# # 🏆 The winning model is: **Random Forest** with an F1-Score of 0.727.



# --- Next Steps ---

# 1. **Select the Winner:** Based on the results, the winning model should be used for the f

# 2. **Final Tuning (Optional):** You can perform hyperparameter tuning specifically on the

# 3. **Final Explainability:** Use the winning model to generate the final SHAP plots and rc


# Classification Report (TUNED XGBoost):

# precision recall f1-score support



# OK 0.50 0.57 0.53 44

# Reject 0.54 0.59 0.56 46
```

```
# Warning 0.12 0.07 0.09 27



# accuracy 0.46 117

# macro avg 0.39 0.41 0.40 117

# weighted avg 0.43 0.46 0.44 117 these values i am getting is it good


import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
import lightgbm as lgb
import xgboost as xgb
from imblearn.over_sampling import SMOTE
TARGET = 'Quality_Label'
features = [col for col in train_df.columns if col not in ['Batch_ID', 'Brew_Date', 'Quality
X_train = train_df[features].copy()
X_test = test_df[features].copy()

y_train_binary = train_df[TARGET].apply(lambda x: 0 if x == 'OK' else 1)
y_test_binary = test_df[TARGET].apply(lambda x: 0 if x == 'OK' else 1)

X_train['Temp_x_pH'] = X_train['Temperature'] * X_train['pH_Level']
X_test['Temp_x_pH'] = X_test['Temperature'] * X_test['pH_Level']

X_train['Grav_x_Alc'] = X_train['Gravity'] * X_train['Alcohol_Content']
X_test['Grav_x_Alc'] = X_test['Gravity'] * X_test['Alcohol_Content']

smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train_binary)

print("\n--- Starting Model Training and Evaluation Bake-Off ---")

models = {
    'Logistic Regression': LogisticRegression(random_state=42, max_iter=1000, class_weight='
    'Random Forest': RandomForestClassifier(random_state=42, class_weight='balanced'),
    'LightGBM': lgb.LGBMClassifier(random_state=42),
    'XGBoost': xgb.XGBClassifier(random_state=42, eval_metric='logloss', use_label_encoder=F
}

results = {}

for name, model in models.items():
    print(f"\n--- Training {name} ---")
    model.fit(X_train_resampled, y_train_resampled)
```

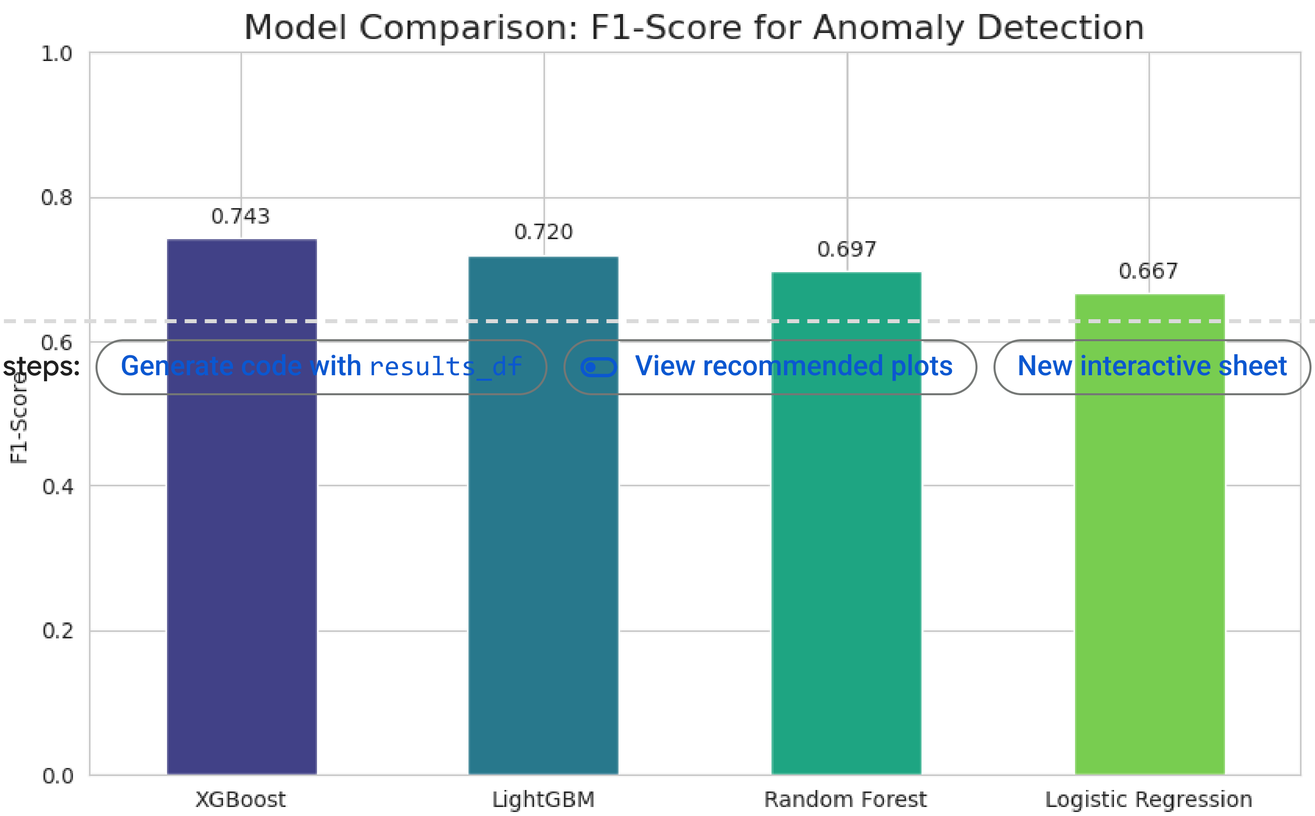
```
        y_pred = model.predict(X_test)
        y_prob = model.predict_proba(X_test)[:, 1]

        precision = precision_score(y_test_binary, y_pred)
        recall = recall_score(y_test_binary, y_pred)
        f1 = f1_score(y_test_binary, y_pred)
        roc_auc = roc_auc_score(y_test_binary, y_prob)

        results[name] = {
            'Precision': precision,
            'Recall': recall,
            'F1-Score': f1,
            'ROC-AUC': roc_auc
        }
        print(f"{name} evaluation complete.")
    print("\n--- Model Performance Comparison ---")
    results_df = pd.DataFrame(results).T.sort_values(by='F1-Score', ascending=False)

    print("\nPerformance of All Models (Sorted by F1-Score):")
    display(results_df)

    plt.figure(figsize=(10, 6))
    results_df['F1-Score'].plot(kind='bar', color=sns.color_palette('viridis', len(results_df)))
    plt.title('Model Comparison: F1-Score for Anomaly Detection', fontsize=16)
    plt.ylabel('F1-Score')
    plt.xticks(rotation=0)
    plt.ylim(0, 1)
    for index, value in enumerate(results_df['F1-Score']):
        plt.text(index, value + 0.02, f"{value:.3f}", ha='center')
    plt.show()

    winning_model_name = results_df.index[0]
    print(f"\n🏆 The winning model is: **{winning_model_name}** with an F1-Score of {results_df
```

```
--- Starting Model Training and Evaluation Bake-Off ---

--- Training Logistic Regression ---
Logistic Regression evaluation complete.

--- Training Random Forest ---
Random Forest evaluation complete.

--- Training LightGBM ---
[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Info] Number of positive: 274, number of negative: 274
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3290
[LightGBM] [Info] Number of data points in the train set: 548, number of used features
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
LightGBM evaluation complete.

--- Training XGBoost ---
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:183: UserWarning: [10:05:4
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
XGBoost evaluation complete.

--- Model Performance Comparison ---

Performance of All Models (Sorted by F1-Score):
```

	Precision	Recall	F1-Score	ROC-AUC
XGBoost	0.733333	0.753425	0.743243	0.694583
LightGBM	0.701299	0.739726	0.720000	0.728829
Random Forest	0.670886	0.726027	0.697368	0.669521
Logistic Regression	0.649351	0.684932	0.666667	0.641034



Next steps: Generate code with results_df View recommended plots New interactive sheet

🏆 The winning model is: **XGBoost** with an F1-Score of 0.743

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier

rf_param_grid = {
    'n_estimators': [100, 200, 300, 500],
    'max_depth': [5, 10, 15, 20, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2']
}

# Initialize and run the search
rf_model = RandomForestClassifier(random_state=42, class_weight='balanced')
rf_search = RandomizedSearchCV(
    estimator=rf_model,
    param_distributions=rf_param_grid,
    n_iter=50,
    cv=5,
    verbose=2,
    random_state=42,
    n_jobs=-1,
    scoring='f1'
)
```

```
!pip install streamlit pyngrok pandas joblib scikit-learn shap matplotlib -q
print("✅ All libraries installed.")
```

```
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.9/9.9 MB 54.0 MB/s eta 0:00:00
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.9/6.9 MB 97.0 MB/s eta 0:00:00
✅ All libraries installed.
```

```
# BEFORE USING THIS YOU LOAD ALL MY MODELS AND STREAMLIT FILE OR YOU CAN IT THAT I PROVIDED
import os

model_path = 'models/random_forest.joblib'
data_path = 'data/features/test_dataset.csv'

if not os.path.exists(model_path) or not os.path.exists(data_path):
    print("❌ ERROR: Files not found!")
    print("Please make sure 'random_forest.joblib' is in the 'models' folder and 'test_datas
else:
    print("✅ All necessary files are present.")
```

```
✅ All necessary files are present.
```

```
%%writefile app/streamlit_app.py
import streamlit as st
import pandas as pd
import joblib
```