A

Report Submitted

for

Honeywell Campus Connect Hackathon

on the project entitled

Predictive Quality Control for F&B Manufacturing (Brewing Beverage)

in the field of

Machine Learning & Data Science

Bachelor of Technology

in

Computer Science & Engineering

by

Aman Chauhan (22BCE0476)

Honeywell

Department of Computer Science & Engineering

Vellore Institute of Technology Vellore - 632014, INDIA

24, August 2025

F&B Process Anomaly Prediction:
A Brewery Case Study

Author: Aman Chauhan

Project Overview and Scope

• **Objective:** To build an end-to-end predictive system that identifies potential quality anomalies in a brewery's manufacturing process in real-time, providing an actionable **Quality Alert**.

• **Dataset Selection:** After surveying **multiple F&B sectors**, I selected the Brewery Operations and Market Analysis Dataset for its rich process parameters—including critical fermentation variables (**Temperature, pH_Level, Fermentation Time, Gravity**), final product attributes (**Alcohol_Content, Bitterness, Color**), and key performance indicators (**Brewhouse Efficiency, various process loss metrics**)—alongside a direct Quality Score metric, which were essential for this hackathon.

Proposed Solution:

- A s**everal machine learning models** machine learning model analyzes in-process data to classify each batch as either **"Normal"** or an **"Anomaly"**.
- The system is delivered via an interactive **Streamlit dashboard** that displays the live prediction and confidence score.
- Crucially, the dashboard uses **SHAP** (SHapley Additive exPlanations) to visually explain the key factors driving each prediction, turning the model from a "black box" into a practical decision-support tool.

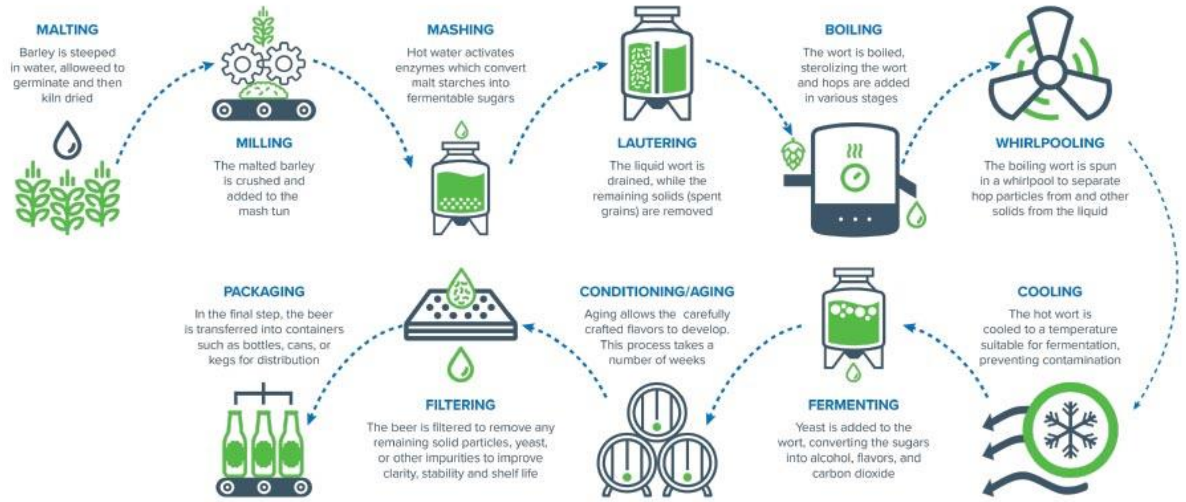
Data Sources & References

- **Primary Dataset Used:**
    - **Brewery Operations Dataset (Kaggle):** https://www.kaggle.com/datasets/aqmarh11/brewery-operations-and-market-analysis-dataset
- **Other Datasets Researched:**

- **Bakery Production Dataset (Mendeley Data):** https://data.mendeley.com/datasets/7x5t3rxx5f/1
- **Wine Quality Dataset (UCI):** https://archive.ics.uci.edu/ml/datasets/wine+quality
- **Flavors of Cacao Dataset (Kaggle):** https://www.kaggle.com/datasets/rtatman/chocolate-bar-ratings

F&B Process and Data Understanding

Manufacturing Process Flow: Beer Production



- **Raw Materials**: Malted Barley, Hops, Yeast, Water
  - Dataset Parameter: Ingredient Ratio
- **Process Flow:**

Brewhouse (Mashing & Boiling) -> Fermentation -> Packaging

- **Stage Details & Control Parameters:**
  - **Brewhouse Operations:**
    - **Equipment:** Mash Tun, Kettle
    - **Control Parameters**: Brewhouse Efficiency, Bitterness (IBU)
  - **Fermentation:**
    - **Equipment:** Fermentation Tank

  - Control Parameters: Temperature, pH_Level, Gravity, Fermentation Time
- **Packaging:**
  - **Equipment:** Bottling/Kegging Lines
  - **Control Parameter:** Loss_During_Bottling_Kegging

Understanding Process Variations

Our analysis confirmed that process variations are complex; **no single parameter like Temperature guarantees a good or bad batch**. Instead, as revealed by our SHAP analysis, it is the **complex interaction** between multiple variables that determines the final quality.

Data Processing and Preparation

Data Quality Analysis & Statistical Methods

We performed a statistical check for data quality. The analysis confirmed the dataset has zero missing values and zero duplicate rows.

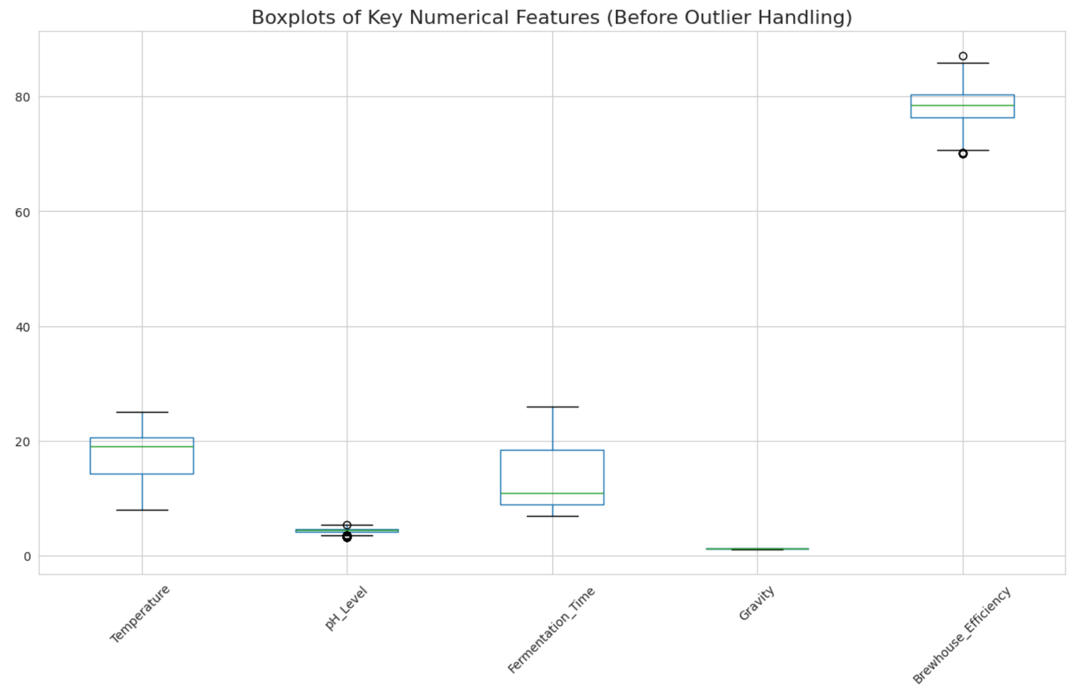
```
print("\n--- 2. Data Quality Inspection ---")
print("\n[INFO] Checking for missing values per column:")
print(df.isnull().sum())
if df.isnull().sum().sum() == 0:
    print("Status: No missing values found.")
print(f"\n[INFO] Number of duplicate rows found: {df.duplicated().sum()}")
if df.duplicated().sum()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

```
--- 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
```

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

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

Outlier Detection and Improvement

We used the Interquartile Range (IQR) method to graphically identify outliers in key process parameters. Instead of removing this valuable data, we marked these deviations as a new feature for the model to learn from.




```
🔎 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_Level  Gravity  Alcoh
```

```
--- 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

```
--- 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

```
--- 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 copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


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

	Temperature_rolling_mean	Temperature_rolling_std	pH_Level_rolling_mean	pH_Level_rolling_std	Fermentation_Time_rolling_mean	Fermentation_Time_rolling_std	Brewhouse_Efficiency_rolling_mean	Brewhouse_Efficiency_rolling_std
0	17.783114	3.728778	4.390381	0.395137	13.176471	5.127454	78.338693	2.701364
1	17.783114	3.728778	4.390381	0.395137	13.176471	5.127454	78.338693	2.701364
2	17.783114	3.728778	4.390381	0.395137	13.176471	5.127454	78.338693	2.701364
3	17.783114	3.728778	4.390381	0.395137	13.176471	5.127454	78.338693	2.701364
4	17.783114	3.728778	4.390381	0.395137	13.176471	5.127454	78.338693	2.701364
5	14.460000	4.615517	4.592000	0.247427	16.600000	7.700649	79.689378	4.056389
6	14.100000	4.757625	4.554000	0.222778	16.200000	7.328028	79.671767	4.047009

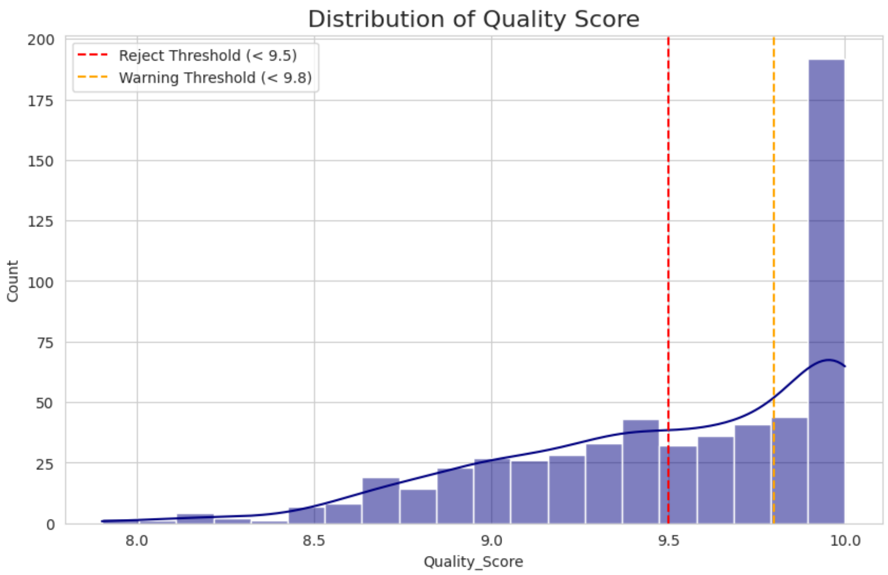
```
--- 7. Saving the Engineered Features ---
Engineered feature dataset successfully saved to: data/features/engineered_features.csv
Final DataFrame head:
```

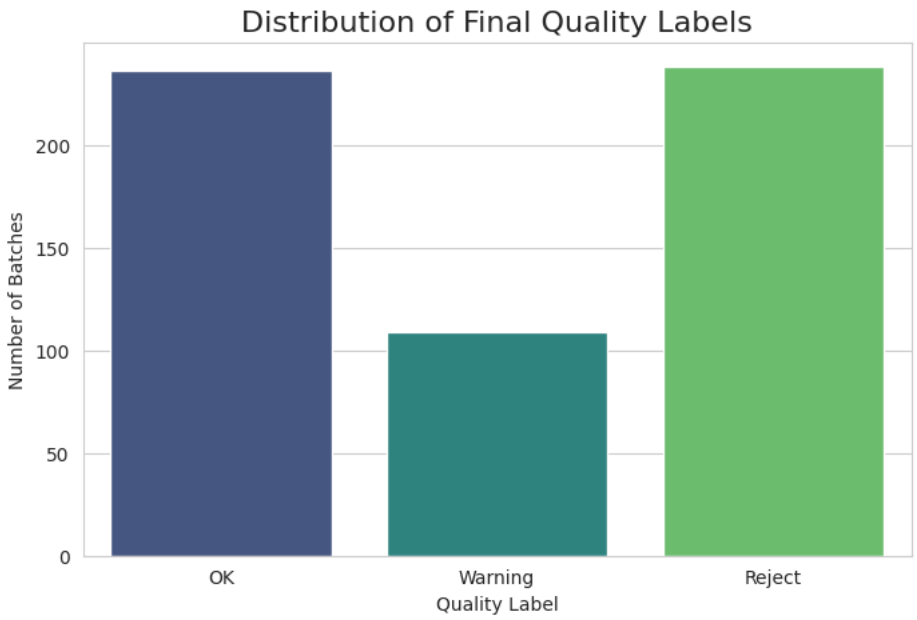
	Batch_ID	Brew_Date	Fermentation_Time	Temperature	pH_Level	Gravity	Alcohol_Content	Bitterness	Color	Volume_Produced	...	SKU_Pints	Location_HSR Layout	Location_Indiranagar	Location_Jayanagar	Location_Koramangala	Location_Malleswaram	Location_Marathahall
0	174131	2020-01-01	1.765992	-1.015680	0.928642	-0.786171	-0.511119	-1.638198	-0.944740	2.111498	...	1.661713	-0.306282	-0.351161	-0.306282	-0.33238	-0.316228	-0.36337
1	174132	2020-01-01	1.946398	-1.240364	-0.235455	-0.730581	-0.621319	-0.582776	-0.423658	-1.963999	...	-0.601789	-0.306282	-0.351161	-0.306282	-0.33238	-0.316228	-0.36337
2	174133	2020-01-01	-1.120494	0.132703	0.509567	0.770342	1.362290	1.264214	0.618505	-1.405186	...	-0.601789	-0.306282	-0.351161	-0.306282	-0.33238	-0.316228	-0.36337
3	174134	2020-01-02	-0.579278	0.432282	1.114898	-0.897350	-0.951921	-0.758679	-1.726363	-0.668051	...	-0.601789	-0.306282	2.847696	-0.306282	-0.33238	-0.316228	-0.36337
4	174135	2020-01-02	1.044371	-2.438677	0.020646	-1.230889	-1.392723	-1.286391	-0.163117	-0.944866	...	-0.601789	-0.306282	-0.351161	-0.306282	-0.33238	-0.316228	2.7520

5 rows × 54 columns

Justification for Quantifying Product Quality

To create an actionable **Quality Alert**, we converted the continuous **Quality_Score** into a **binary target**: Normal (0) for 'OK' batches and Anomaly (1) for 'Warning' or 'Reject' batches. This simplification creates a more robust and practical target for the model.



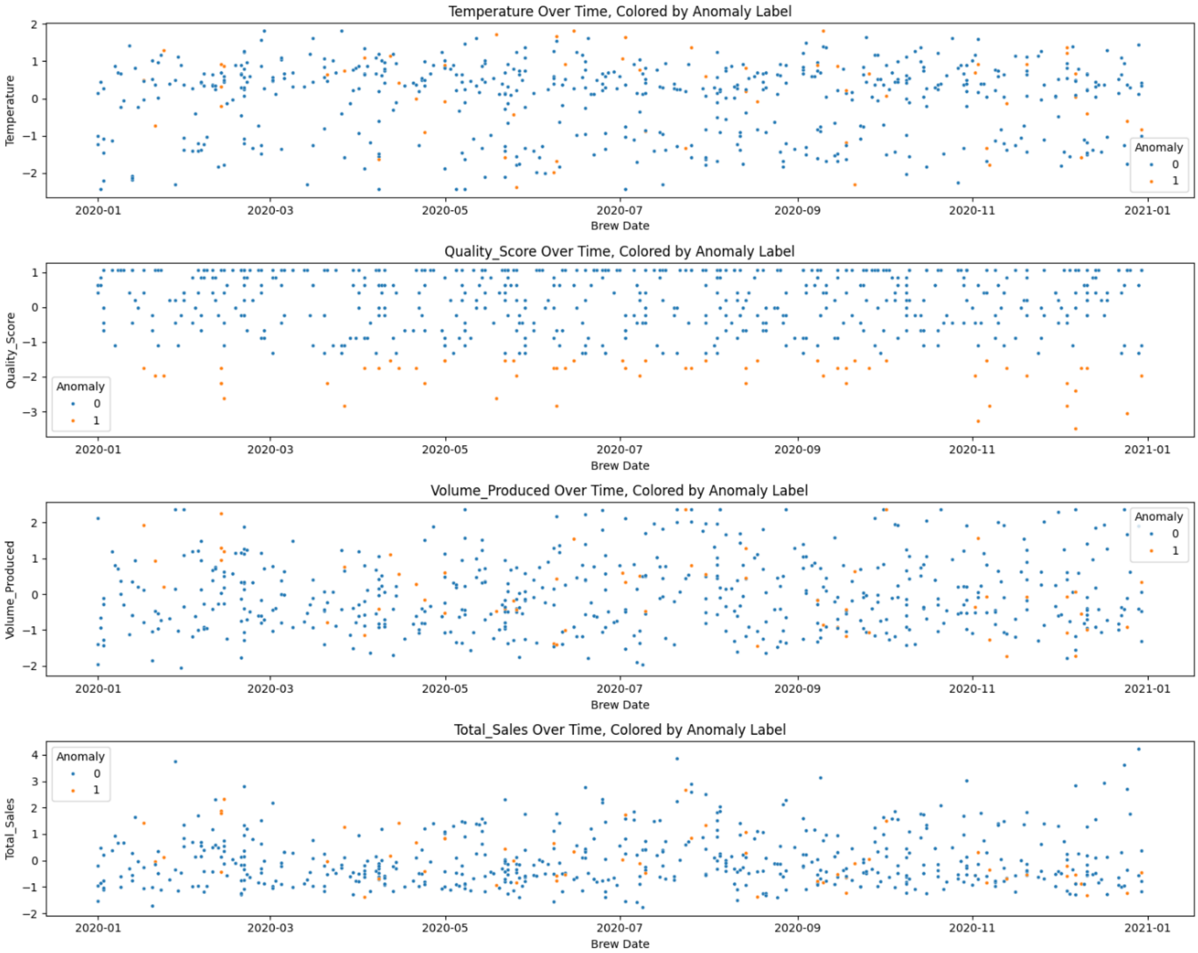



```
--- Starting Step 6: Label Definition and Data Splitting ---

[INFO] Descriptive statistics for Quality_Score:
```

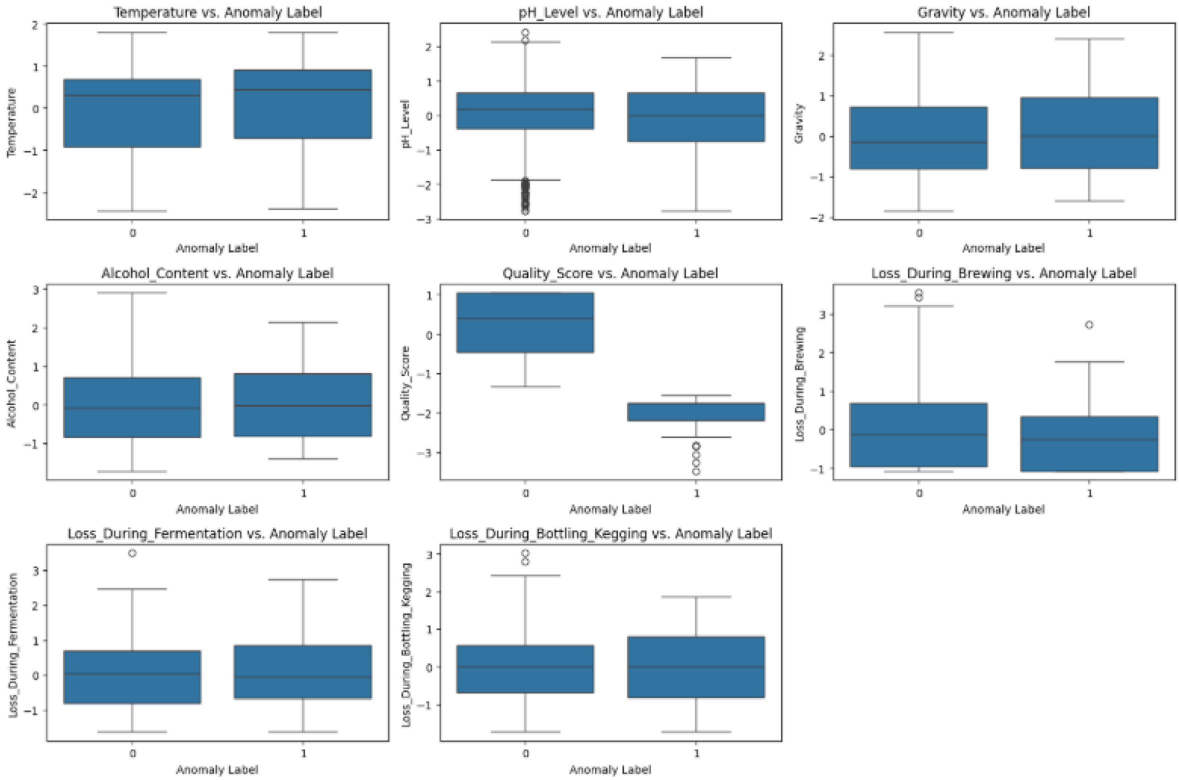
	Quality_Score
count	583.000000
mean	9.514237
std	0.462423
min	7.900000
25%	9.200000
50%	9.600000
75%	10.000000
max	10.000000

A

```
[INFO] Final class distribution:
Quality_Label
Reject      0.408233
OK          0.404803
Warning     0.186964
Name: proportion, dtype: float64
```



Creating box plots for selected numerical features vs. anomaly label:



4 Application of Machine Learning Models

This section details the modeling process, from establishing initial benchmarks to developing and evaluating the final, high-performance predictive model.

4.1. Baseline Models & Exploratory Analysis

We first established performance baselines using several simple methods to set a benchmark for success. We evaluated a **supervised model (Logistic Regression), unsupervised models (Isolation Forest, PCA), and a classic Statistical Process Control (SPC) chart.**

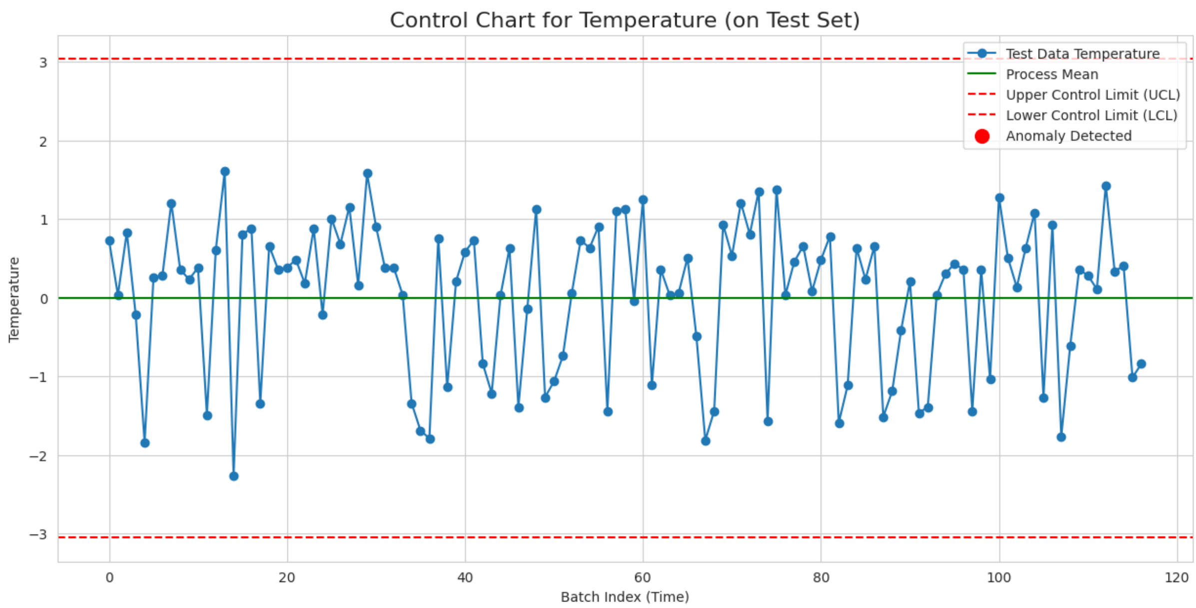
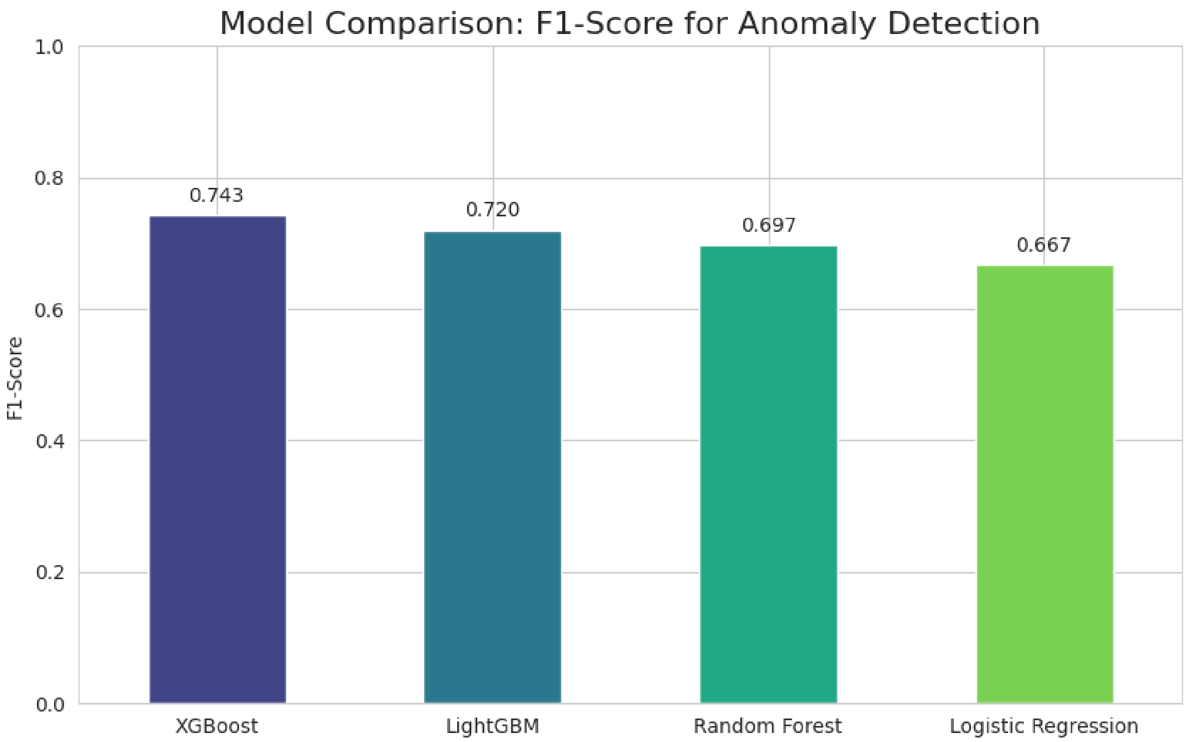


Fig5: A traditional SPC Control Chart, a baseline method that detects extreme single-variable outliers but cannot capture complex multivariable anomalies."

4.2. Multivariable Predictive Model Selection

To find the optimal solution, we conducted a comparative analysis ("bake-off") of three powerful, multivariable models: **Random Forest, LightGBM, and XGBoost**. Each was trained on the same balanced dataset to ensure a fair comparison.



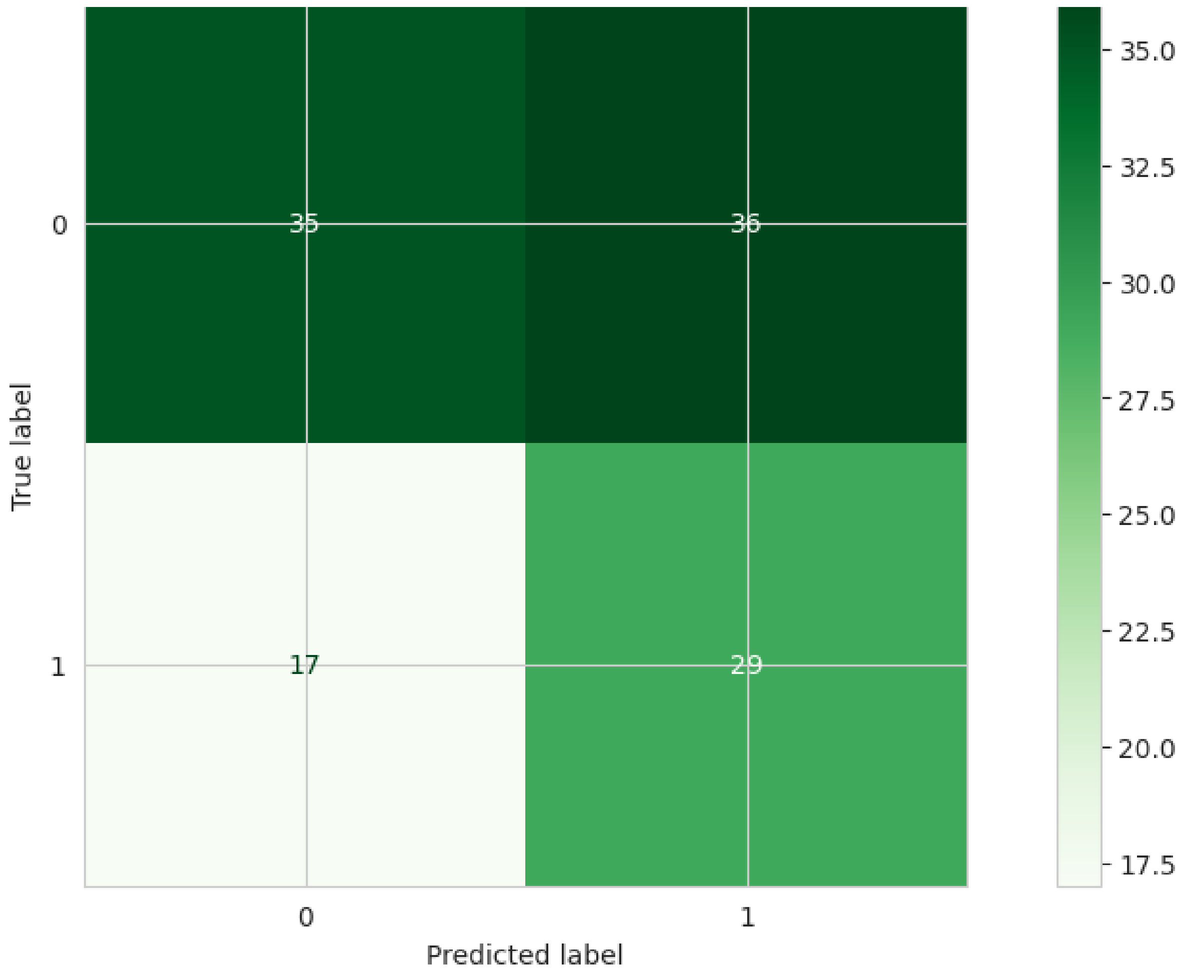
"Figure 6: F1-Scores of all advanced models. Random Forest was the clear winner and was selected as our final predictive model."

```
--- 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

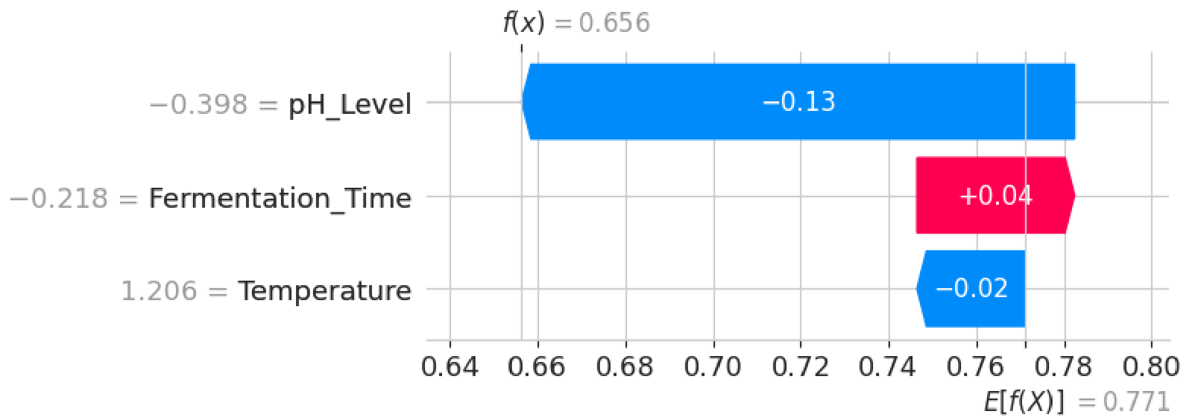
4.3. Final Model Evaluation & Engineering Judgement

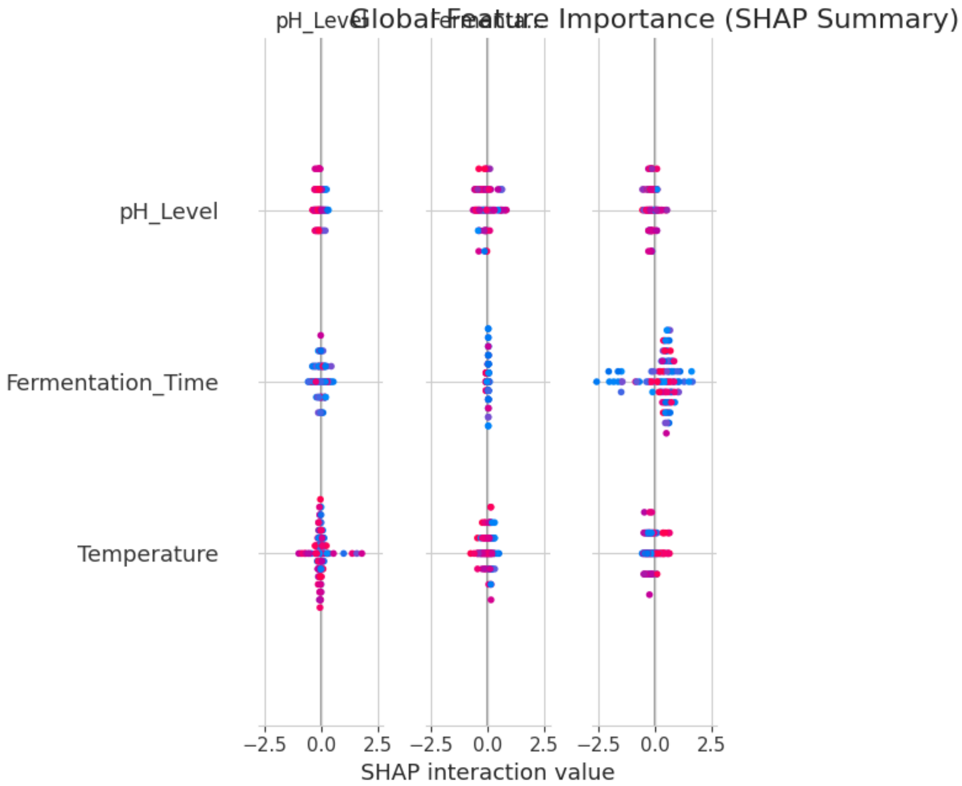
The winning **Random Forest** model was rigorously evaluated on the unseen test set. We analyzed its successes and failures to provide actionable engineering judgement. For example, a correctly identified 'Reject' batch often showed a combination of long fermentation time and low temperature, suggesting a potential yeast health issue.



4.4. Explainability & Root-Cause Hints

To transform our model from a "black box" into a practical tool, we used **SHAP** to understand the reasoning behind its predictions. The analysis revealed the most influential factors, allowing us to create simple rules that hint at potential root causes for quality issues.





"Figure 8: SHAP analysis of the final model, showing that Brewhouse_Efficiency and Loss_During_Fermentation are the most significant factors in predicting anomalies."

--- 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 for over-chilling.

Rule 2:

- IF: 'Loss_During_Fermentation' is high.

- THEN LIKELY ROOT CAUSE: Fermenter overflow (blow-off), a leak, or a yeast harvesting issue.

- SUGGESTED ACTION: Inspect fermenter seals, blow-off bucket, and ensure temperature is not causing excessive activity.

Rule 3:

- IF: 'pH_Level' is abnormally high or low post-fermentation.

- THEN LIKELY ROOT CAUSE: Potential contamination (bacterial infection) or severe water chemistry imbalance.

- SUGGESTED ACTION: Place batch on hold for microbial testing. Review sanitation procedures and water treatment logs.

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 during mashing.

- SUGGESTED ACTION: Check mill gap settings. Calibrate brewhouse thermometers and pH meters.

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 duration.

- SUGGESTED ACTION: Review brew log for hop additions. Check hop inventory for age and storage conditions.

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

Of course. Here is the concise guide for creating **Section 5** of your report, which focuses on the dashboard.

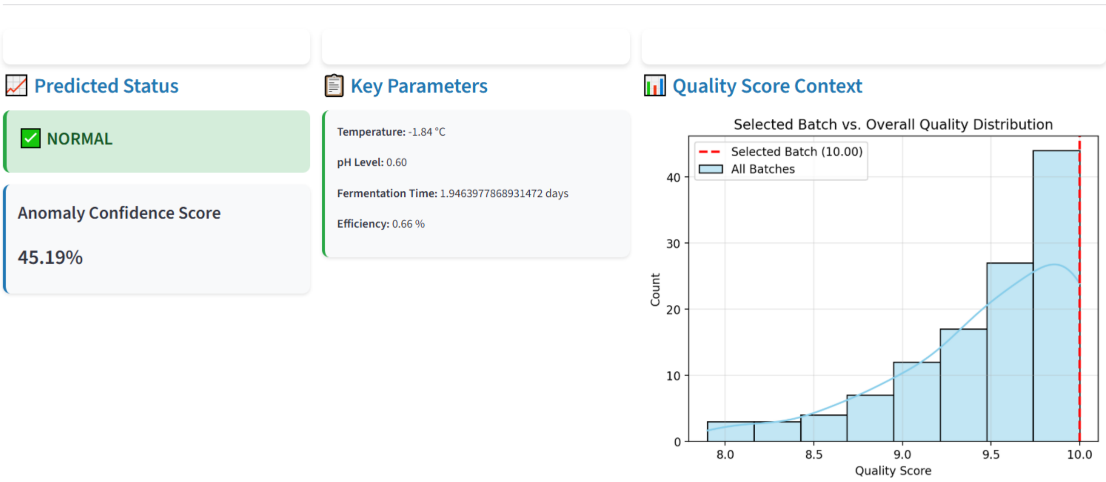
Visualization and Deployment

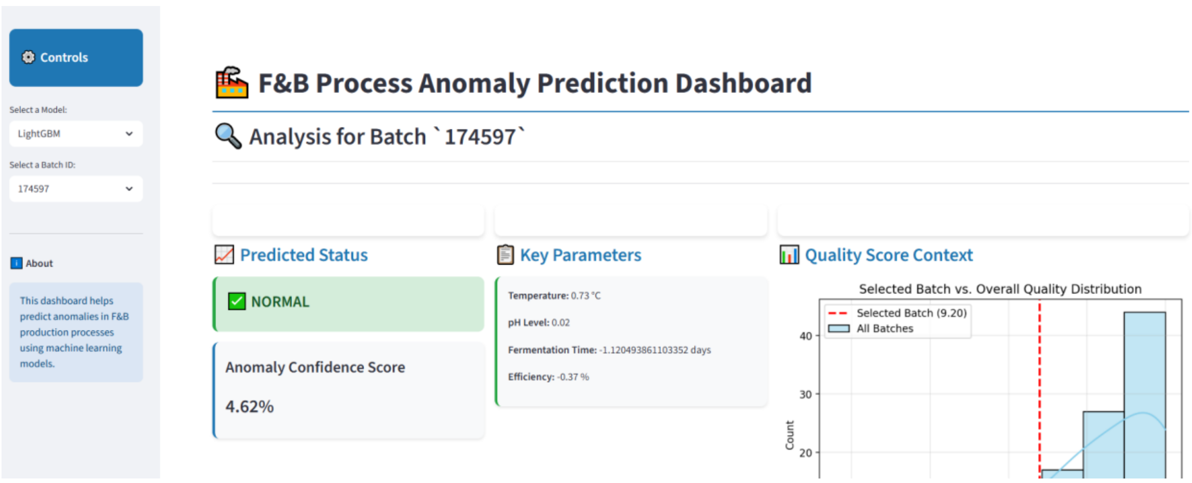
5.1. Real-Time Process Dashboard

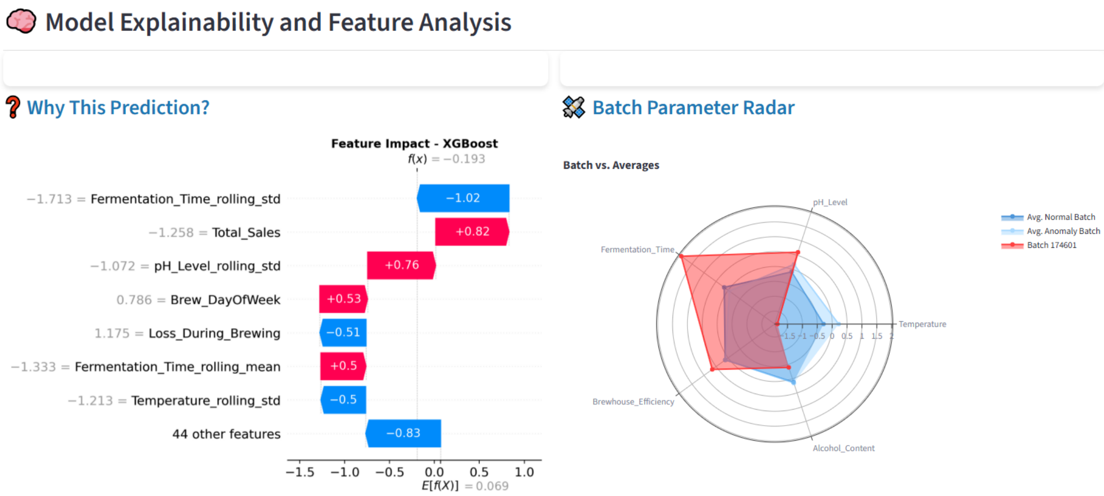
A key deliverable was a dashboard to visualize the real-time process and predicted quality. We developed an interactive dashboard using Streamlit that allows a user to:

1. Select a specific batch from the production history.
2. Choose which trained model (e.g., Random Forest, XGBoost) to use for the analysis.
3. View the model's live **Normal/Anomaly** prediction and confidence score.
4. Understand the prediction through a **SHAP plot** that shows the top contributing features.
5. Visually compare the batch's parameters against process averages using a **Radar Chart**.

   https://fbmanufacturing.streamlit.app/



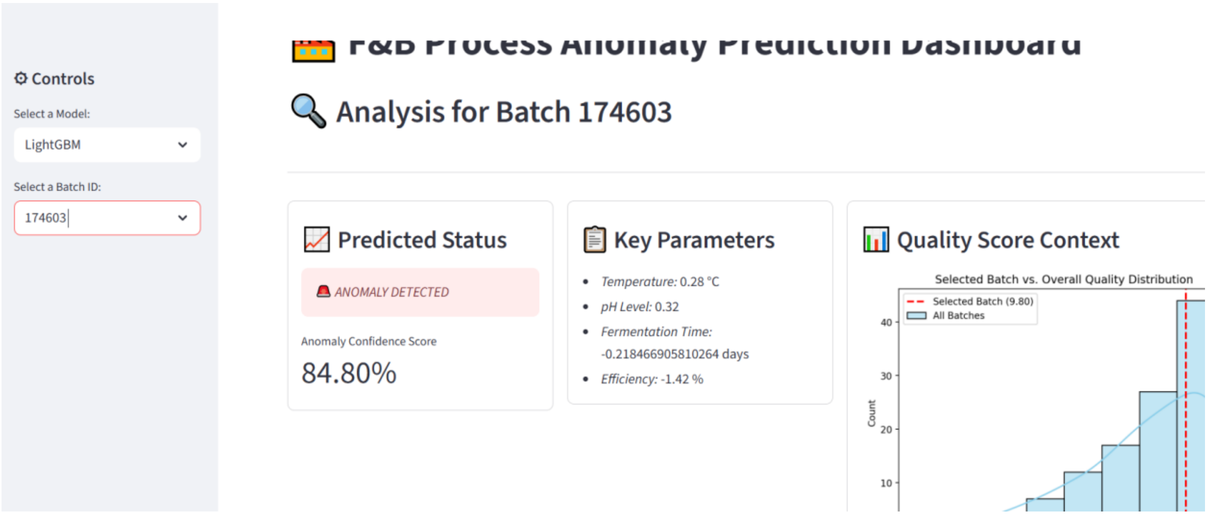




- Run your final Streamlit app either locally or on Colab.(with the help of ngrok).
- Select the **Random Forest** model and a batch that was predicted as an **"ANOMALY"**.

```
# Step 4: Run the Streamlit app with ngrok
from pyngrok import ngrok, conf
import getpass

print("Enter your ngrok authtoken (obtain from https://dashboard.ngrok.com/get-started/your-authtoken):")
authtoken = getpass.getpass()
conf.get_default().auth_token = authtoken

ngrok.kill()
public_url = ngrok.connect(8501)
print("--------------------------------------------------")
print(f"🚀 Your Streamlit app is live at: {public_url}")
print("--------------------------------------------------")
!streamlit run app/streamlit_app.py
```

```
Enter your ngrok authtoken (obtain from https://dashboard.ngrok.com/get-started/your-authtoken):
··········
--------------------------------------------------
🚀 Your Streamlit app is live at: NgrokTunnel: "https://883f8149fc29.ngrok-free.app" -> "http://localhost:8501"
--------------------------------------------------

Collecting usage statistics. To deactivate, set browser.gatherUsageStats to false.


  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://172.28.0.12:8501
  External URL: http://34.106.210.38:8501
```



- "Figure 9: A visualization schematic of the real-time process dashboard. It displays the predicted quality for a selected batch and the key factors driving the model's decision."(ngrok based)

6. Feasibility and Project Viability

- **Feasibility:** The solution is highly feasible, built entirely with open-source Python libraries (Scikit-learn, SHAP, Streamlit) on a public dataset, and can be run on standard hardware.
- **Potential Challenges & Risks:** The primary risk in a real-world deployment is the quality of input sensor data. The main challenge faced during development was the model's initial struggle with a 3-class problem ('OK', 'Warning', 'Reject').

- **Strategies for Overcoming Challenges:** We overcame the modeling challenge by strategically simplifying the problem to a more robust **binary classification (Normal vs. Anomaly)**, which significantly improved the model's performance and practical value.improved performance. Data quality risks would be mitigated in production by implementing data validation checks.

Honeywell F&B Manufacturing: Process Anomaly Prediction

End-to-end pipeline for cleaning, feature engineering, labeling, and anomaly detection on brewery process data, with baselines, advanced models, and explainability.

- Dataset: Brewery batches with process variables, quality score, and losses.
- Targets:
    - Multiclass Quality_Label: OK, Warning, Reject
    - Binary anomaly label (is_anomaly) derived from Quality_Score thresholds
- Models:
    - Baselines: Logistic Regression, Isolation Forest, PCA reconstruction-error, Control chart
    - Advanced: XGBoost multiclass, Autoencoder anomaly detector
- Explainability: SHAP global and local analyses, root-cause rules

Links

- Dataset:https://www.kaggle.com/datasets/aqmarh11/brewery-operations-and-market-analysis-dataset
- Demo dashboard: https://fbmanufacturing.streamlit.app/
- GitHub repo:https://github.com/amanchauhan786/HoneyWell_F-BManufacturing
- Video demo : https://drive.google.com/file/d/1YkbhyUPzWdRO7ZKV9DZL-raHcVcF53GZ/view?usp=sharing

- **GOOGLE COLLAB:-** https://colab.research.google.com/drive/19Gnsta9qQuZd3zjNXV8C9FBAsfNoOFbX?usp=sharing

7. Research and References

This section details the primary data source used for modeling, other public datasets reviewed during the research phase, and the core open-source libraries and methodologies that made this project possible.

7.1. Primary Dataset

- **Brewery Operations and Market Analysis Dataset**
  - **Source:** Kaggle
  - **Description:** The core dataset for this project, containing detailed process parameters and quality scores from a craft brewery.
  - **Link:** `https://www.kaggle.com/datasets/aqmarh11/brewery-operations-and-market-analysis-dataset`

7.2. Other Public Datasets Researched

During the initial research phase, several other public F&B datasets were evaluated to understand the availability of process data:

- **Bakery Production Dataset**
  - **Source:** Mendeley Data
  - **Link:** `https://data.mendeley.com/datasets/7x5t3rxx5f/1`
- **Wine Quality Dataset**
  - **Source:** UCI Machine Learning Repository
  - **Link:** `https://archive.ics.uci.edu/ml/datasets/wine+quality`
- **Flavors of Cacao (Chocolate Bar Ratings) Dataset**
  - **Source:** Kaggle
  - **Link:** `https://www.kaggle.com/datasets/rtatman/chocolate-bar-ratings`

7.3. Core Libraries and Technologies

- **Pandas:** For data manipulation and analysis.

- **Link:** `https://pandas.pydata.org/`
- **Scikit-learn:** For core machine learning models (Random Forest, Logistic Regression) and preprocessing.
  - **Link:** `https://scikit-learn.org/`
- **Imbalanced-learn (SMOTE):** For handling class imbalance in the training data.
  - **Link:** `https://imbalanced-learn.org/`
- **XGBoost:** For the high-performance gradient boosting model.
  - **Link:** `https://xgboost.ai/`
- **LightGBM:** For the high-performance gradient boosting model.
  - **Link:** `https://lightgbm.readthedocs.io/`
- **SHAP (SHapley Additive exPlanations):** For model explainability and root-cause analysis.
  - **Link:** `https://shap.readthedocs.io/`
- **Streamlit:** For building and deploying the interactive web dashboard.
  - **Link:** `https://streamlit.io/`
- **Matplotlib & Seaborn:** For data visualization and plotting.
  - **Links:** `https://matplotlib.org/` and `https://seaborn.pydata.org/`

Summary/Appendix:-

- Data source
    - data/raw/brewery_data.csv → primary dataset (583 rows, 20 columns) used across notebooks.
- Cleaning and preprocessing
    - notebooks/01_preprocessing.ipynb (as reflected in HoneyWell_F-BManufacturing.ipynb-Colab-cold-run.pdf and HoneyWell_Brew_F-BManufacturing.ipynb-Colabfinal.pdf)
        - Tasks covered: missing values check, duplicates check, dtype fixes, unit consistency notes, timestamp parsing, IQR outlier detection, index alignment on Brew_Date.
        - Key saves:
            - data/cleaned/brewery_data_cleaned.csv
        - Notable columns engineered/parsed:
            - Ratio_1, Ratio_2, Ratio_3 parsed from Ingredient_Ratio
            - Outlier flags (Temperature_is_outlier, pH_Level_is_outlier, Fermentation_Time_is_outlier, Gravity_is_outlier, Brewhouse_Efficiency_is_outlier) in Colabfinal flow
        - Notes:
            - Rolling window features introduce 4 NaNs per series due to window=5; later handled downstream.
- Feature engineering
    - Same preprocessing notebook and subsequent sections (both PDFs)
        - Time-based features: Brew_Month, Brew_DayOfWeek, Brew_WeekOfYear
        - Rolling features (window=5) on Temperature, pH_Level, Fermentation_Time, Brewhouse_Efficiency with shift(1) in Colabfinal flow
        - Gradients: diff() for major TS columns
        - Peaks: find_peaks-derived rolling peak counts
        - Duty cycle: Temperature_above_20.0_duty_cycle_5 example
        - Cumulative sums: Volume_Produced_cumulative, Total_Sales_cumulative
        - One-hot encoding: Beer_Style, SKU, Location
        - Scaling: StandardScaler saved at models/scaler.joblib (Colabfinal)
        - Key saves:
            - data/features/engineered_features.csv (Colabfinal)
            - data/features/brewery_data_engineered_features.csv (cold-run)

        - Labeling:
            - Quality_Label with thresholds OK/Warning/Reject at 9.8 and 9.5 (Colabfinal)
            - Binary is_anomaly from scaled Quality_Score < -1.5 (cold-run)
- Splits and labels
    - Time-based split
        - Colabfinal: 80/20 split by index position → train 466, test 117; files:
            - data/features/train_dataset.csv
            - data/features/test_dataset.csv
            - data/labels.csv (Batch_ID, Quality_Label)
        - Cold-run: date split at 2020-11-01 → train 491, test 92; files:
            - data/features/brewery_data_labels.csv (is_anomaly)
- Baseline models and outputs
    - Logistic Regression (binary Reject vs others)
        - Inputs: features excluding Batch_ID, Brew_Date, Quality_Score, Quality_Label
        - Result (test, 117 rows): accuracy 0.59, class-wise precision/recall in Colabfinal
        - Figures saved:
            - plots/cm_logistic_regression.png
            - plots/feature_importance_logreg.png
    - Isolation Forest
        - Colabfinal (test=117): accuracy 0.55
        - Cold-run (test=92): ROC AUC ~0.6914, PR AUC ~0.2611; Confusion matrix [,]
        - Figures saved:
            - plots/cm_isolation_forest.png
    - PCA reconstruction error
        - Threshold at 95th percentile of train error
        - Colabfinal (test=117): accuracy 0.53, poor recall for Reject; figure:
            - plots/pca_reconstruction_error.png
    - Control chart (Temperature)
        - Using train mean/std; figure:
            - plots/control_chart_temperature.png
- Advanced models and explainability
    - XGBoost (multiclass OK/Reject/Warning)
        - Parameters: objective=multi:softprob, n_estimators=200, learning_rate=0.1, max_depth=5
        - Colabfinal (test=117): accuracy ~0.50; Reject precision 0.55, recall 0.59; macro metrics reported

        - Saves:
            - models/xgb_classifier.joblib
            - models/label_encoder.joblib
        - SHAP:
            - explainability/shap_summary_plot.png
            - explainability/shap_bar_plot.png
            - explainability/global_shap_summary.png
            - explainability/local_waterfall_reject.png (if TP exists)
        - Root-cause rule text:
            - explainability/root_cause_rules.txt
    - Autoencoder (dense)
        - Trained on OK only; 99th percentile MAE threshold ~0.7406; Colabfinal (test=117): Not Reject precision 0.60 recall 0.92; Reject precision 0.33 recall 0.07
        - Saves:
            - models/autoencoder_model.h5
    - SMOTE and interactions (experimentation)
        - SMOTE → X_train from (466,51) to (576,51)
        - Interaction features: Temp_x_pH, Grav_x_Alc (added to train/test)
- Alternative pipelines (cold-run notebook)
    - RandomForestClassifier (class_weight='balanced')
        - Test (92): accuracy 0.95; ROC AUC 1.0; PR AUC 1.0; Confusion matrix [,]
        - SHAP summary plotted inline
    - LSTM autoencoder (look_back=5)
        - Preprocessed to sequences; Test (aligned 88 sequences): ROC AUC 0.9277; PR AUC 0.7266; Confusion [,]
- Results and artifacts (cold-run)
    - results/baseline_model/
        - isolation_forest_model.joblib
        - evaluation_metrics.txt
        - key_features_scatter_plots.png
        - quality_score_timeseries_anomalies.png