ደብረ ብርሃን ዩኒቨርሲቲ

DEBRE BERHAN UNIVERSITY

DEPARTMENT OF SOFTWARE ENGINEERING

Quality Control in Manufacturing using Machine Learning DOCUMENTATION

SUBMIT TO : Derbew Felasman(MSc)

NAME :SARON ZELEKE

ID:1402302

1. Problem Definition

Quality Control in Manufacturing:

Quality control is vital in ensuring that products meet specific standards and customer expectations. Defects in products can lead to increased costs, customer dissatisfaction, and potential safety hazards. Traditional manual inspection methods are time-consuming and prone to human error. This project aims to develop a machine learning model to automate quality control by predicting whether a manufactured item is defective based on various features collected during the production process.

Challenges:

Difficulty in acquiring real-world datasets.

Variability in defect definitions across industries.

Imbalance in defective vs. non-defective samples.

To address the data availability challenge, synthetic data was generated to train the machine learning model.

2. Data Generation and Description

Synthetic Data Generation:

Due to the unavailability of real-world quality control datasets, synthetic data was generated with 500 samples and 30 numerical features. These features represent measurable product characteristics (e.g., weight, thickness, color consistency). The dataset also includes a binary target variable, defective, where 90% of the samples are non-defective, and 10% are defective.

Example Data Generation:

```
import pandas as pd

import numpy as np



n_samples = 500

data = {

    'feature1': np.random.normal(loc=50, scale=5, size=n_samples),

    'feature2': np.random.normal(loc=20, scale=3, size=n_samples),

    'feature3': np.random.normal(loc=30, scale=4, size=n_samples),

    'feature4': np.random.normal(loc=25, scale=2, size=n_samples),

    'feature5': np.random.normal(loc=60, scale=6, size=n_samples),

    'feature6': np.random.normal(loc=15, scale=1.5, size=n_samples),

    'feature7': np.random.normal(loc=70, scale=7, size=n_samples),

    'feature8': np.random.normal(loc=35, scale=5, size=n_samples),

    'feature9': np.random.normal(loc=45, scale=4, size=n_samples),

    'feature10': np.random.normal(loc=55, scale=5, size=n_samples),

    'feature11': np.random.normal(loc=65, scale=6, size=n_samples),

    'feature12': np.random.normal(loc=75, scale=7, size=n_samples),

    'feature13': np.random.normal(loc=80, scale=8, size=n_samples),

    'feature14': np.random.normal(loc=85, scale=9, size=n_samples),

    'feature15': np.random.normal(loc=90, scale=10, size=n_samples),

    'feature16': np.random.normal(loc=95, scale=11, size=n_samples),

    'feature17': np.random.normal(loc=100, scale=12, size=n_samples),

    'feature18': np.random.normal(loc=105, scale=13, size=n_samples),
    'feature19': np.random.normal(loc=110, scale=14, size=n_samples),
    'feature20': np.random.normal(loc=115, scale=15, size=n_samples),
    'feature21': np.random.normal(loc=120, scale=16, size=n_samples),
    'feature22': np.random.normal(loc=125, scale=17, size=n_samples),
    'feature23': np.random.normal(loc=130, scale=18, size=n_samples),
    'feature24': np.random.normal(loc=135, scale=19, size=n_samples),
    'feature25': np.random.normal(loc=140, scale=20, size=n_samples),
    'feature26': np.random.normal(loc=145, scale=21, size=n_samples),
    'feature27': np.random.normal(loc=150, scale=22, size=n_samples),
    'feature28': np.random.normal(loc=155, scale=23, size=n_samples),
    'feature29': np.random.normal(loc=160, scale=24, size=n_samples),
    'feature30': np.random.normal(loc=165, scale=25, size=n_samples),
    'defective': np.random.choice([0, 1], size=n_samples, p=[0.9, 0.1])
}
df = pd.DataFrame(data)
df.to_csv('quality_control_data.csv', index=False)
```

3. Exploratory Data Analysis (EDA) Findings and Visualizations

Findings:

The dataset had no missing values.

Outliers were detected using boxplots.

Data was visualized using histograms and a correlation heatmap.

Features showed varying levels of correlation, requiring feature scaling.

Summary Statistics:

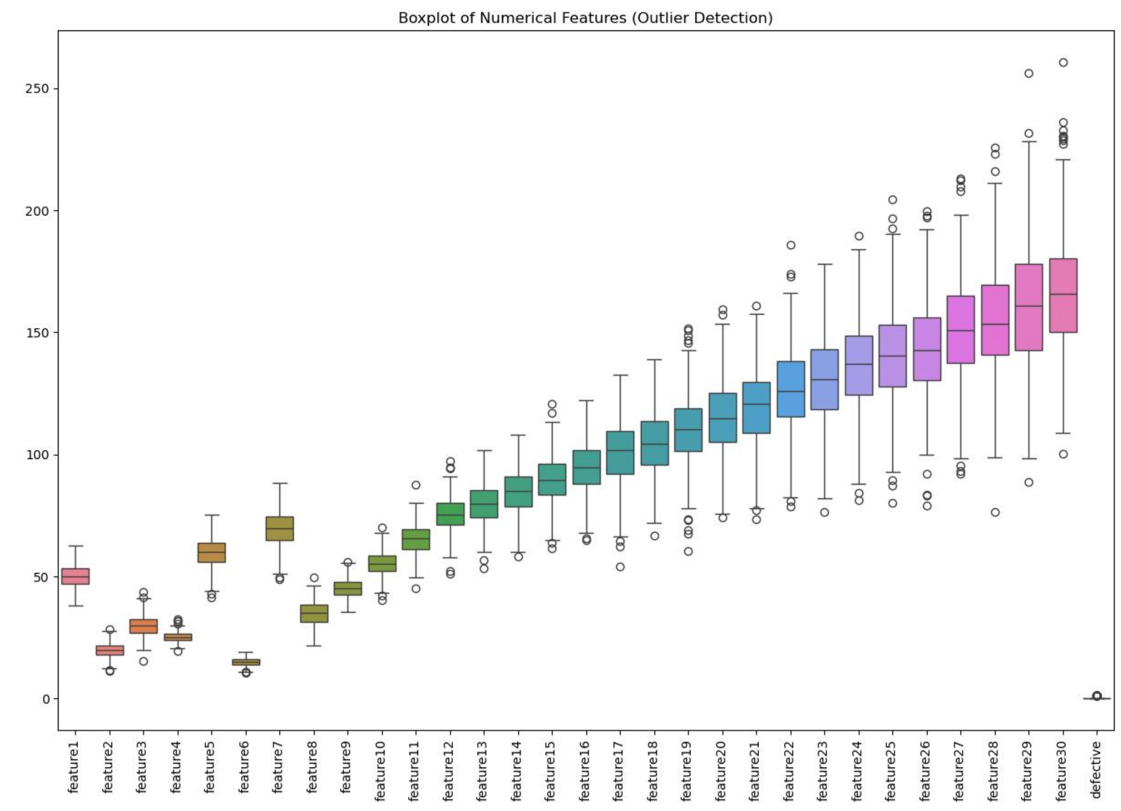
```
print(df.describe())
```

Provides key statistics such as mean, standard deviation, min, max, and quartiles for each feature.

EDA Visualizations

1. Boxplot for Outlier Detection:

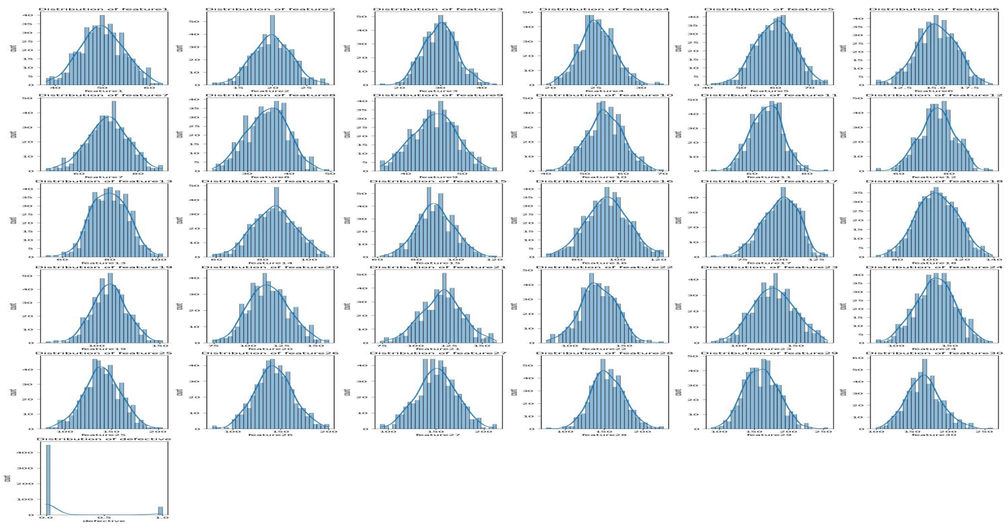
```
plt.figure(figsize=(15, 10))

sns.boxplot(data=df)

plt.xticks(rotation=90)

plt.title("Boxplot of Numerical Features (Outlier Detection)")

plt.show()
```



Boxplot helps identify outliers in the features. Outliers are data points that deviate significantly from the rest of the data.

2. Histograms for Feature Distributions:

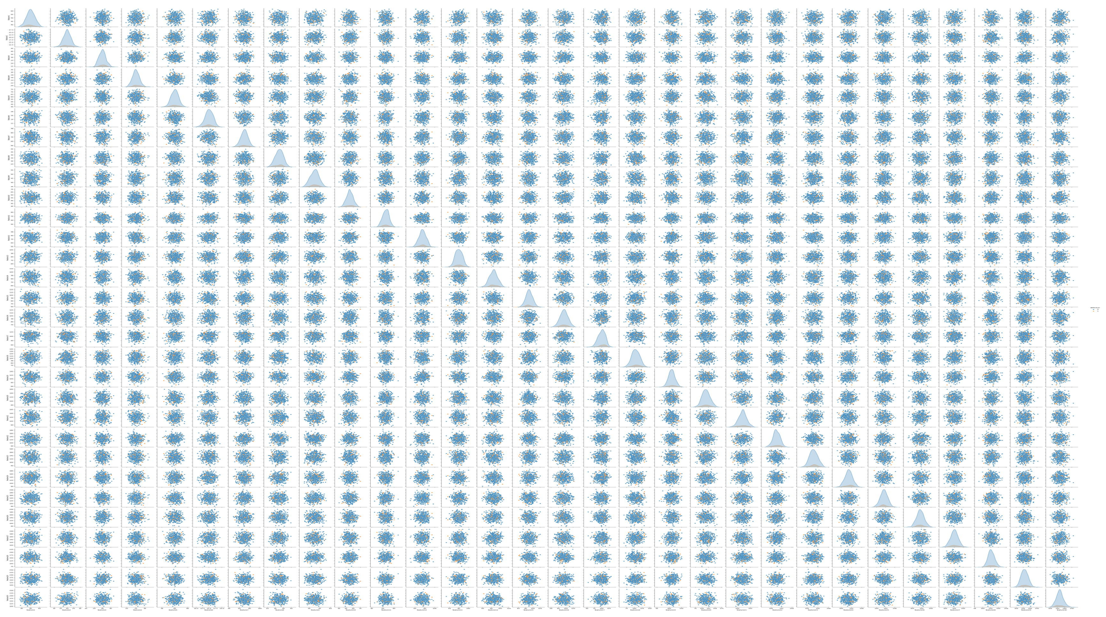
```
plt.figure(figsize=(15, 5 * num_rows), constrained_layout=True)

for i, column in enumerate(df.columns):

    plt.subplot(num_rows, num_cols, i + 1)

    sns.histplot(df[column].dropna(), bins=30, kde=True)

    plt.title(f"Distribution of {column}")

    plt.xlabel(column)

    plt.ylabel('count')

plt.show()
```



Histograms show the distribution of each feature, indicating whether the data follows a normal distribution and highlighting any skewness or anomalies.

3. Pairplot for Relationships:

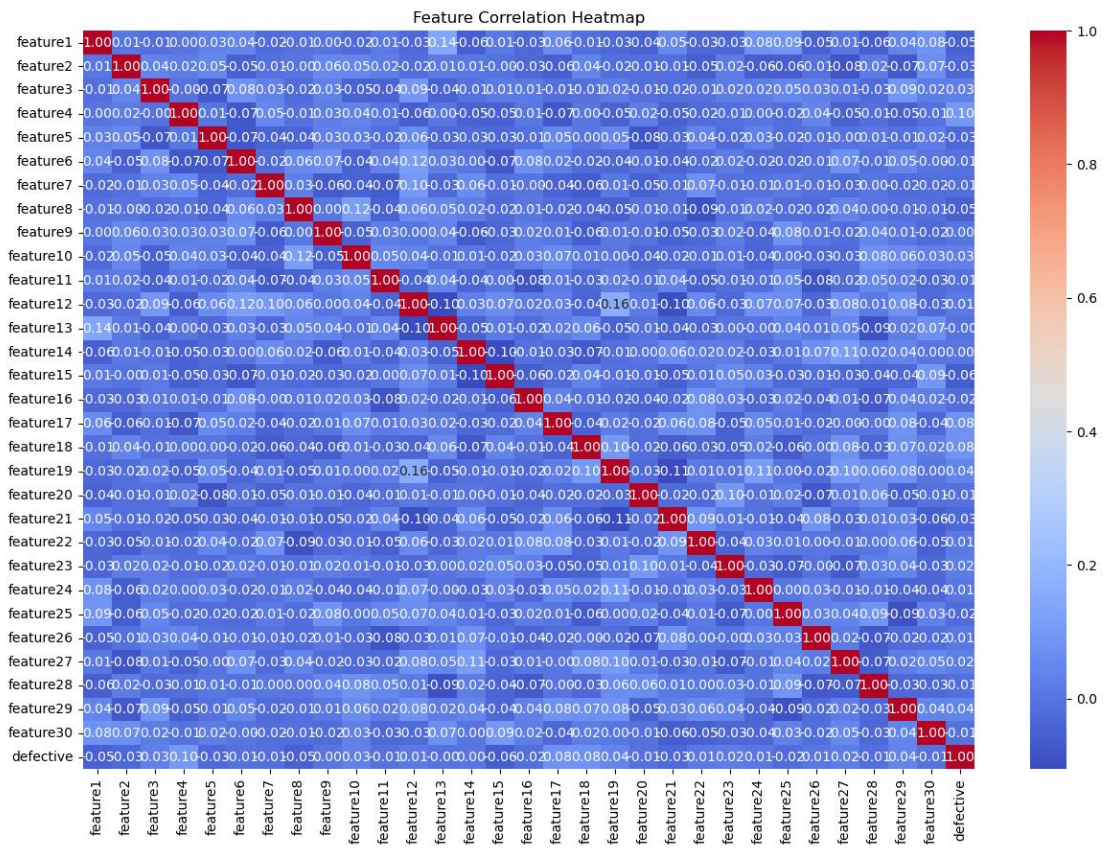
```
if 'defective' in df.columns:

        sns.pairplot(df, hue='defective')

plt.show()
```



Pairplot visualizes the relationships between features, coloring the data points by the target variable (defective). This helps identify any feature interactions and separability between classes.

4. **Correlation Heatmap:**

```
plt.figure(figsize=(15, 10))

sns.heatmap(df.corr(), annot=True, cmap="coolwarm", fmt=".2f")

plt.title("Feature Correlation Heatmap")

plt.show()
```



Correlation heatmap shows the correlation coefficients between features, helping identify multicollinearity and feature dependencies.

4. Data Preprocessing

Handling Missing Values:

```
print(df.isnull().sum())
```

No missing values were found in the synthetic data. However, in a real-world scenario, appropriate methods such as imputation or deletion would be used to handle missing values.

Scaling Features:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

df[df.columns[:-1]] = scaler.fit_transform(df[df.columns[:-1]])
```

Standardization ensures features have a mean of 0 and a standard deviation of 1, which is crucial for many machine learning algorithms to perform optimally.

Outlier Handling:

```
for col in df.columns[:-1]:    # Exclude the target column

        lower_percentile = df[col].quantile(0.01)

        upper_percentile = df[col].quantile(0.99)

        df[col] = df[col].clip(lower_percentile, upper_percentile)
```

Outliers were clipped to the 1st and 99th percentiles to prevent them from skewing the analysis and model training. Clipping helps maintain the majority of the data while reducing the impact of extreme values.

Justification of Preprocessing Choices:

Scaling: Standardization is crucial for models that rely on distance metrics or gradient-based optimization, such as logistic regression, SVM, and neural networks.

Feature Scaling: Standardized using StandardScaler to normalize values.

Train-Test Split: 80% training, 20% testing.

5. Model Selection and Training

Chosen Model: Random Forest Classifier due to its robustness with tabular data.

```
from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier



X = df.drop(columns=['defective'])

y = df['defective']

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

model = RandomForestClassifier(random_state=42)

model.fit(X_train, y_train)
```

Data was split into training and testing sets with an 80-20 ratio. The model was trained on the training set (X_train, y_train) and tested on the test set (X_test, y_test).

Hyperparameter Tuning:

```
from sklearn.model_selection import GridSearchCV



param_grid = {

    'n_estimators': [50, 100, 200],

    'max_depth': [None, 10, 20, 30],

    'min_samples_split': [2, 5, 10]

}

grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5,
scoring='accuracy', n_jobs=-1)

grid_search.fit(X_train, y_train)

best_model = grid_search.best_estimator_

print("Best Parameters:", grid_search.best_params_)
```

Trained an initial model and then optimized it using GridSearchCV.

6. Model Evaluation

Metrics:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
classification_report, confusion_matrix
```

7. Deployment and Reproducibility

To ensure that the trained model can be used in real-world applications, I try to deployed it using FastAPI. FastAPI is a modern, fast (high-performance), web framework for building APIs with Python 3.7+ based on standard Python type hints.

Steps to Deploy the Model

1. Save the Model and Scaler:

Save the trained model and scaler using joblib.

2 Set Up FastAPI:

3.Create the API Endpoint

* POST /predict: Accepts JSON input with feature data and returns a prediction.

4. Run the FastAPI Server:

* Use uvicorn to run the FastAPI application.

```
Python -m uvicorn quality:app --reload
```

For more detail code you can see quality.py file

8. Potential Limitations and Future Improvements

Limitations:

Data Authenticity: Since the dataset is synthetic, real-world validation is necessary to ensure the model's effectiveness.

Feature Selection: Further domain knowledge is needed to refine feature engineering and ensure the most relevant features are used.

Handling Class Imbalance: Techniques like SMOTE (Synthetic Minority Over-sampling Technique) can be explored to improve recall for the minority class.

Future Improvements:

Alternative Models: Deep learning-based approaches could be tested for more complex defect patterns.

Real-World Data: Collect real-world data for training and validation to improve model robustness and applicability.

Integration: Deploy the model in an IoT-based quality control system or a real-time manufacturing line for automated defect detection.