# Wafer Defect Detection

## 1. Aim

The objective is to develop a machine learning model that accurately detects wafer defects based on given sensor data. This helps in identifying faulty wafers early in the production process, reducing manufacturing losses.

## 2. Motivation

Semiconductor manufacturing involves wafer production, where defects can significantly impact yield and quality. Manual inspection is time-consuming and prone to errors; thus, an automated ML-based solution is needed.

## 3. Dataset

We use the "Wafer Fault Detection" dataset from sources like UCI Machine Learning Repository or industry datasets. The dataset contains multiple sensor readings per wafer, labeled as "good" or "bad" wafers.

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

# Load dataset

data = pd.read\_csv('wafer\_data.csv') # Replace with actual dataset path

## 4. Exploratory Data Analysis (EDA)

Checking dataset properties:  
- Display first few rows  
- Summary statistics  
- Checking for missing values  
- Visualization of target variable  
- Correlation heatmap

# Checking dataset properties

print(data.head())

print(data.info())

print(data.describe())

# Checking for missing values

print(data.isnull().sum())

# Distribution of target variable

sns.countplot(x='target', data=data)

plt.title("Distribution of Wafer Defects")

plt.show()

# Correlation heatmap

plt.figure(figsize=(10,8))

sns.heatmap(data.corr(), annot=False, cmap='coolwarm')

plt.title("Feature Correlation Heatmap")

plt.show()

## 5. ML Model Justification

We use the Random Forest classifier due to:  
- Its robustness to noise and missing data  
- Ability to handle high-dimensional datasets  
- Feature importance insights

## 6. ML Model Code (Random Forest Example)

Splitting dataset, training Random Forest model, and making predictions.

# Splitting dataset

X = data.drop(columns=['target']) # Features

y = data['target'] # Target variable

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train model

rf\_model = RandomForestClassifier(n\_estimators=100, random\_state=42)

rf\_model.fit(X\_train, y\_train)

# Predictions

y\_pred = rf\_model.predict(X\_test)

## 7. Metrics for Model Evaluation

Model performance is evaluated using:  
- Accuracy Score  
- Classification Report  
- Confusion Matrix

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("Classification Report:\n", classification\_report(y\_test, y\_pred))

print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

## 8. Self Inference

The model provides good accuracy, but further hyperparameter tuning may improve performance. Feature selection techniques can be applied to reduce computation time. Imbalanced data handling (e.g., SMOTE) can be considered if needed.

## 9. Scope for Enhancement

Possible improvements include:  
- Exploring deep learning models such as CNNs for image-based wafer defect detection.  
- Using more advanced ensemble techniques.  
- Deploying the model into a real-time quality control system.