Defect Rate Prediction

Objective

Predict the defect rate 7 days in advance to improve quality control and reduce production defects. Business Problem: High defect rates in the production process led to increased costs, wasted materials, and reduced customer satisfaction. By accurately predicting defect rates, TDK aims to proactively address potential issues, optimize production processes, and enhance overall product quality. Data: Historical data from various steps in the production process, including measurements and defect rates. Sample data attached, please extrapolate, and come up with the approach.

Data Availability –

The following data sets are utilized in the project:

- Mount Terminals
- Mount Terminals Resin
- Wind Wire
- Peel Wire
- Check Alignment
- Defect Rates

Tools Used

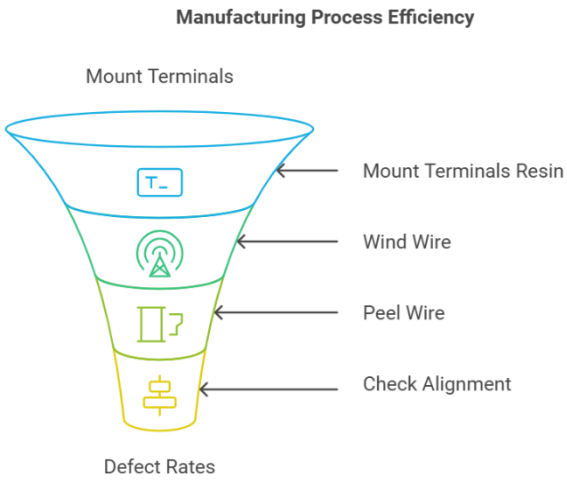
Python (NumPy, pandas, seaborn, matplotlib, sklearn), pytorch

Techniques –

Assumption

- Assume the data set share they followed sequential order with each data set
- In production it follows the steps like

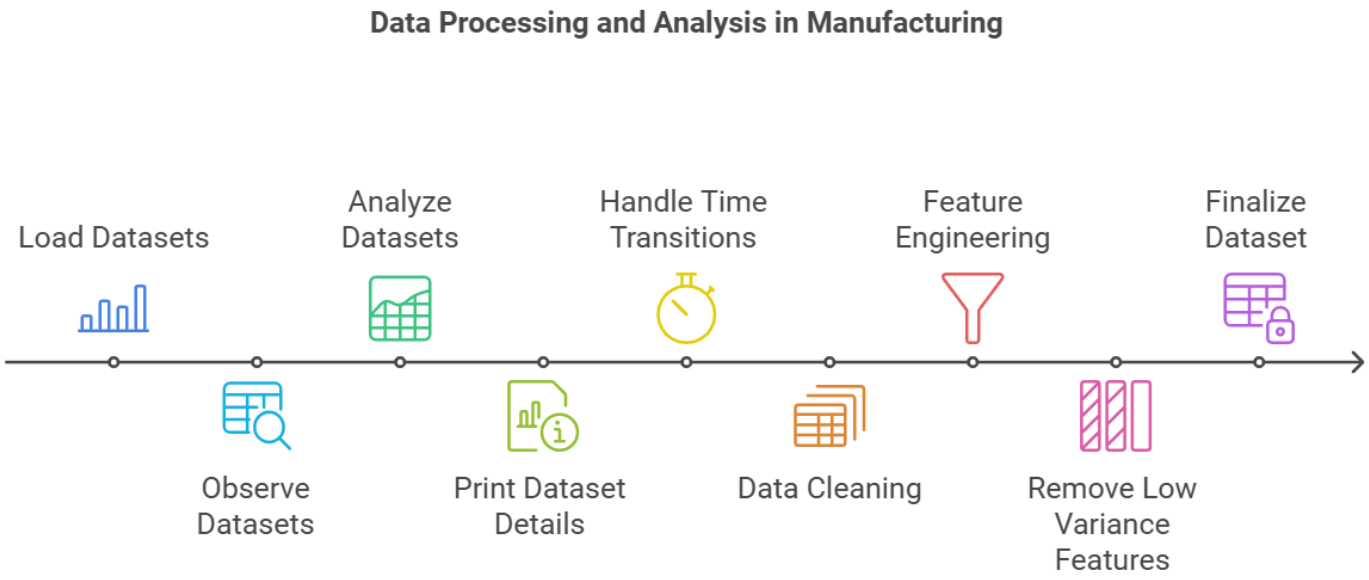
Mount Terminals >Mount Terminals Resin >Wind Wire>Peel Wire >Check Alignment>Defect Rates



Steps followed to solve the problem.

1. Load the whole dataset and analyse the which provide the production process dataset in each steps date, time, measurement count, defect rates etc are given for each step in the production process.
2. Observe that data six datasets representing different steps in a factory process are loaded. These include defect rates, terminal mounting, resin application, wire winding, wire peeling, and alignment checking.

3. Each dataset represents a specific step in the manufacturing process and analysing them collectively helps identify correlations or issues.
4. Prints the size, missing values, and unique counts for each dataset.
5. To understand the structure, check for completeness, and assess the variability of the data.
6. **Handling Time Transition**:
    a. Reverses the defect rate dataset for easier handling of time transitions.
    b. A custom parse_time function converts time strings into minutes and seconds, checking for boundary transitions (e.g., 59:58 to 00:00).
    c. Handling time transitions is critical for maintaining temporal integrity in time-series data.
7. **Data Cleaning and Feature Engineering**:
    a. Combines date and time into a `DateTime` column and extracts useful time-based features (month, year, hour).
    b. `Defect Rate` is converted from a percentage string to a decimal value.
    c. These transformations standardize data formats and create features for downstream analysis.
8. **Low Variance Feature Removal**:
    a. Drops columns with low variance (same values throughout).
    b. Low variance features do not contribute to the model and may add noise.
    c. Merging data enables holistic analysis of the entire production pipeline.
9. **Final Processing and Combining the data.**
    a. Similar transition handling as the defect rate dataset is applied to the combined dataset.
    b. Extracts time features (month, year, hour) and finalizes the dataset by dropping unnecessary columns.
    c. By handling time transitions and ensuring consistency across datasets, the preprocessing step ensures better model performance and fewer errors during analysis.



After performing the data preprocessing step, we store combined_processed.csv and defect_rate_preprocessed.csv data.
We perform EDA on the cleaning data set to understand the trend and seasonality in the dataset and check the data is stationary or not.

Model Building and Evaluation

The train_time_series_model.py script aims to predict a time series metric, specifically the "Defect Rate," using two approaches:

- Long Short-Term Memory (LSTM), a deep learning method
- Auto ARIMA, a statistical forecasting technique

Data Loading and Preprocessing

1. load_and_resample(filepath):

- Loads a CSV file containing time-series data.
- Resamples the "Defect Rate" column at a minute-level granularity and fills missing values forward/backward.

2. train_test_split (data, train_size=0.8):

- Splits the time-series data into training and testing sets, preserving temporal order.

LSTM Data Preparation

prepare_lstm_data (series, seq_length):

- Converts the time series into sequences of fixed length (`seq_length`) for LSTM input.
- Each sequence serves as an input (X), and the next value in the series is the target (y).

LSTM Model Definition and Training

- LSTMModel (nn. Module) :
  - Defines the architecture of the LSTM model.
  - Uses one LSTM layer followed by a fully connected layer for predictions.
  - train_lstm_model (model, dataloader, criterion, optimizer, epochs) `:
  - Trains the LSTM model using the Mean Squared Error (MSE) loss function.
  - Iteratively updates model weights over multiple epochs.

Model Evaluation

evaluate_model (model, dataloader, scaler):

- Evaluates the LSTM model's performance by generating predictions and calculating metrics (MSE, RMSE, MAPE).
- Inversely transforms the predictions and actual values to their original scale for evaluation.

Auto ARIMA Forecasting

auto_arima_forecast (train, test):

- Fits an Auto ARIMA model on the training data.
- Selects the best ARIMA model parameters using stepwise optimization.
- Generates predictions for the test data.

Results Visualization and Saving

plot_and_save (train, test, lstm_pred, arima_pred, filename) `:

- Plots actual vs predicted values from both LSTM and ARIMA models.
- Saves the plot to a specified file path.

Metrics Calculation and Export

1. *Metrics Computation:*
   a. Compares LSTM and ARIMA predictions with actual values over a common timeframe.

b. Calculates MSE, RMSE, and MAPE for both models.

2. *Metrics Export:*
   a. Saves metrics to a CSV file for further analysis.

Workflow

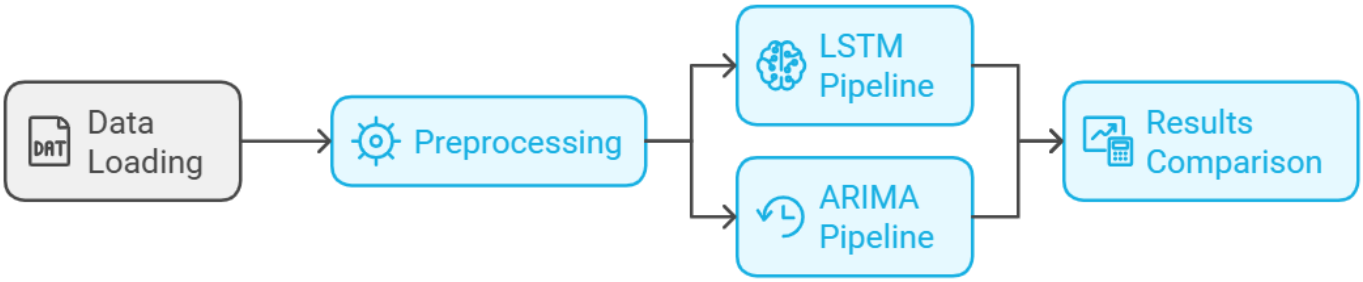
a. Data Loading: Reads data from a file path defined in the configuration file (`cf. FILEPATH`).
b. Preprocessing: Resamples data and splits it into training and testing sets. Scales the data using MinMaxScaler for compatibility with LSTM.
c. LSTM Pipeline: Prepares input data for LSTM. Trains the LSTM model and evaluates its performance on the test set.
d. ARIMA Pipeline: Fits an Auto ARIMA model and generates forecasts for the test set.
e. Results Comparison: Visualizes and compares the performance of both models. Exports performance metrics to a CSV file.

Key Libraries Used

- numpy, pandas: For numerical computations and data manipulation.
- scikit-learn: For scaling and evaluation metrics.
- torch (PyTorch): For defining and training the LSTM model.
- pmdarima: For fitting and forecasting with Auto ARIMA.
- matplotlib: For visualizing results.

Outputs

- Predictions Plot:
- A visual comparison of actual vs predicted values for both models, saved as an image.
- Metrics CSV File:
- A CSV file containing MSE, RMSE, and MAPE metrics for LSTM and ARIMA models.



Defect Rate prediction using Regression method.

The machine learning pipeline for defect rate prediction. It includes data processing, feature selection, model training, hyperparameter tuning, evaluation, and explainability using SHAP. Each step is encapsulated in separate scripts for maintainability and reusability.

1. ***Data Processing (data_processing.py)***

- Purpose: Clean and prepare the dataset for machine learning.
- Key Functions:
- `load_data () `: Loads combined dataset and defect rate file using pandas.
- `preprocess_data () `: Drops irrelevant columns, selects numerical columns for model readiness.
- `plot_correlation_matrix () `: Generates a heatmap of correlations to visualize relationships between features and the target.

- Data preparation ensures the dataset is clean, reduces noise, and highlights key relationships for feature engineering.

2. Feature Selection (feature_selection.py)

- Purpose: Identify the most relevant features for predicting defect rate.
- Key Functions:
    - `select_features () `: Uses Recursive Feature Elimination (RFE) with a linear regression model to select top features.
    - Reducing feature dimensionality improves model efficiency and reduces the risk of overfitting.
    - Use fifty to select top features effecting the target column.

3. Model Training (model_training.py)

- Purpose: Train and evaluate different machine learning models.
- Key Functions:
    - `split_data () `: Splits the data into training and testing sets.
    - `train_linear_regression () `: Fits a linear regression model.
    - `train_random_forest () `: Fits a random forest regressor.
    - `train_xgboost () `: Fits an XGBoost regressor.
    - `evaluate_model () `: Calculates R^2, MSE, RMSE, and MAPE metrics for model performance.
    - Training multiple models allows comparison to identify the best-performing approach.

4. Hyperparameter Tuning (hyperparameter_tuning.py)

- Purpose: Optimize model parameters for improved performance.
- Key Functions:
- `random_search_rf () `: Performs randomized search for random forest hyperparameters.
- `random_search_xgb () `: Performs randomized search for XGBoost hyperparameters.
- Systematic hyperparameter tuning ensures models achieve optimal performance without manual trial-and-error.

5. SHAP Analysis (shap_analysis.py)

- Purpose: Interpret model predictions by analyzing feature importance.
- Key Functions:
- `analyze_shap () `: Generates SHAP summary plots to visualize feature impact on predictions.
- Explainability is critical for trust and decision-making in ML models, especially in high-stakes scenarios.

6. Main Script (main.py)

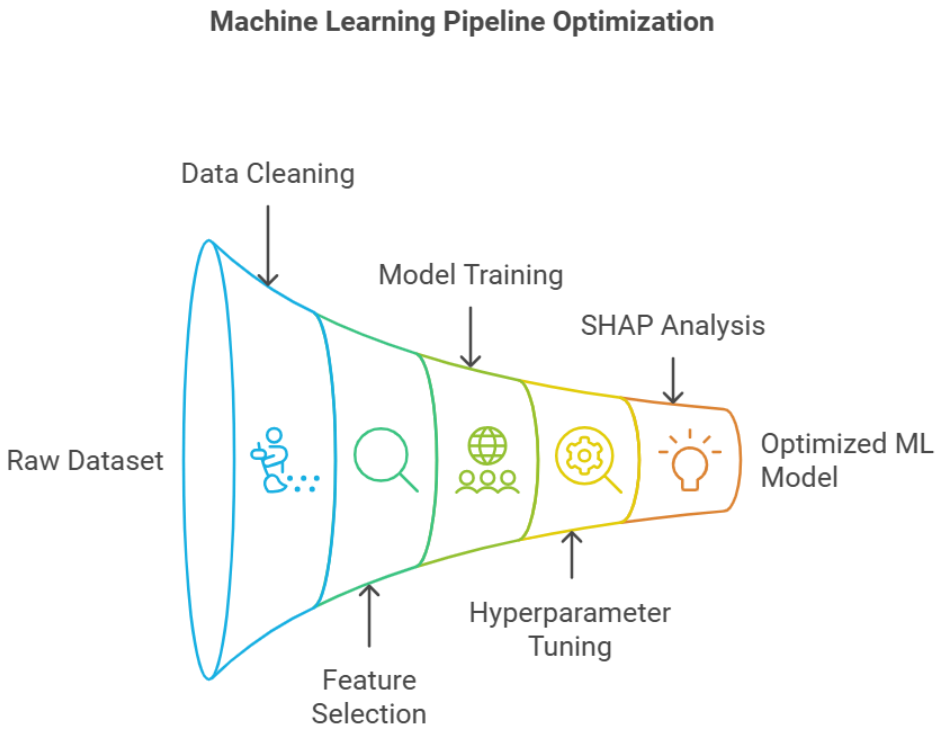
- **Orchestrates the end-to-end ML pipeline**: -Steps:

1. Loads data and preprocesses it.
2. Generates correlation heatmap.
3. Selects features using RFE.
4. Splits data into training and testing sets.
5. Trains three models (linear regression, random forest, XGBoost).
6. Evaluates models using R^2, MSE, RMSE, and MAPE.
7. Performs hyperparameter tuning for random forest and XGBoost.
8. Evaluates tuned models and saves metrics to an Excel file.
9. Dumps best-tuned models into pickle files for reuse.
10. Performs SHAP analysis on the tuned random forest model.

- The pipeline ensures seamless integration and testing of each component. Results are documented, models are saved, and insights are visualized for reproducibility and clarity.

Output Details

- **Metrics**:
  Evaluation metrics are consolidated into an Excel file (`tuned_metrics.xlsx`).
- Includes R^2, MSE, RMSE, and MAPE for tuned models.
- **Models:**
  Tuned models are saved as `best_rf_model.pkl` and `best_xg_model.pkl` for deployment.
- **Visualization:**
  SHAP plots are saved as `shap_feature_importance.png` for feature importance analysis.
- **Correlation Plot:**
  Visual correlation analysis saved as correlationplot.png.



Metrices-

model_metrics.csv

This file contains metrics for two models (LSTM and ARIMA) with the following performance metrics:

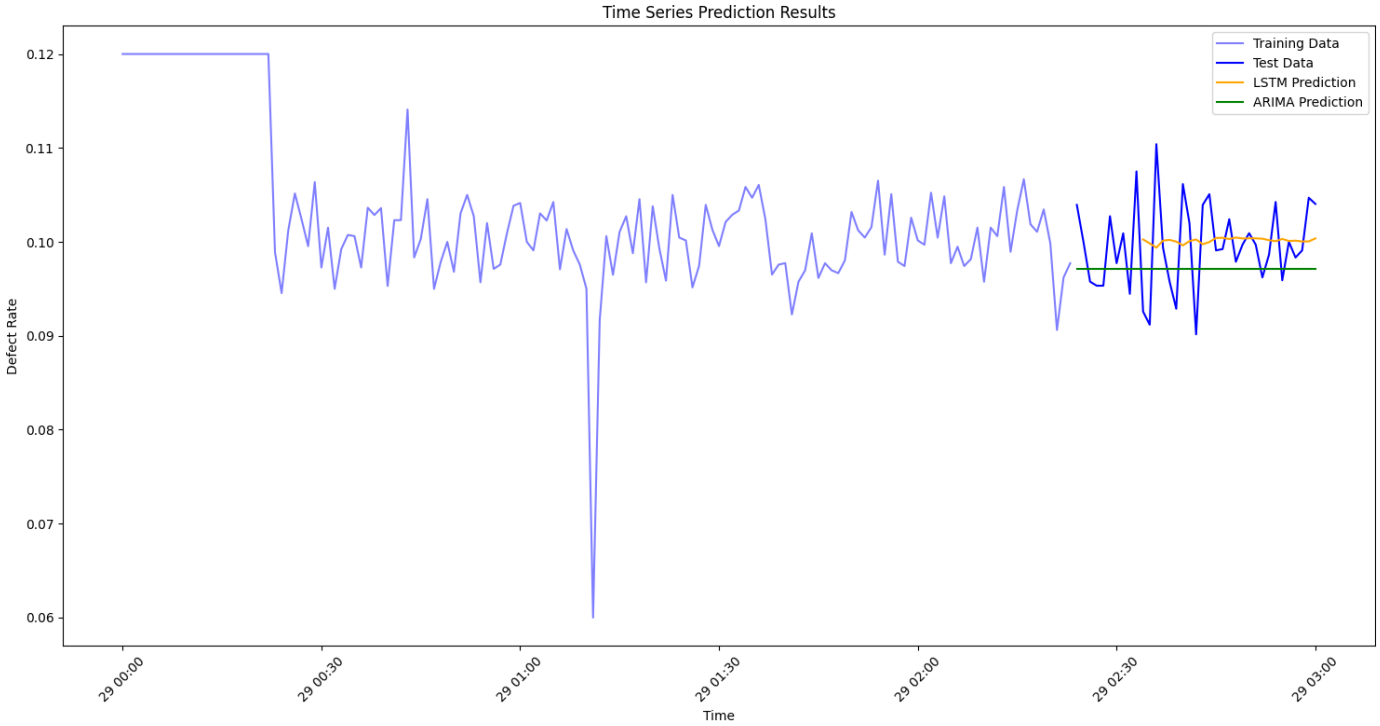
- **MSE (Mean Squared Error)**: Measures the average squared difference between predicted and actual values.
- **RMSE (Root Mean Squared Error)**: The square root of MSE, indicating the standard deviation of prediction errors.
- **MAPE (Mean Absolute Percentage Error)**: Measures prediction accuracy as a percentage.

tuned_model_metrics.csv

This file contains metrics for two tuned models (Random Forest and XGBoost) with additional performance metrics:

- **R^2 (Coefficient of Determination)**: Indicates the proportion of variance explained by the model.
- **MSE, RMSE, and MAPE**: Same as above.

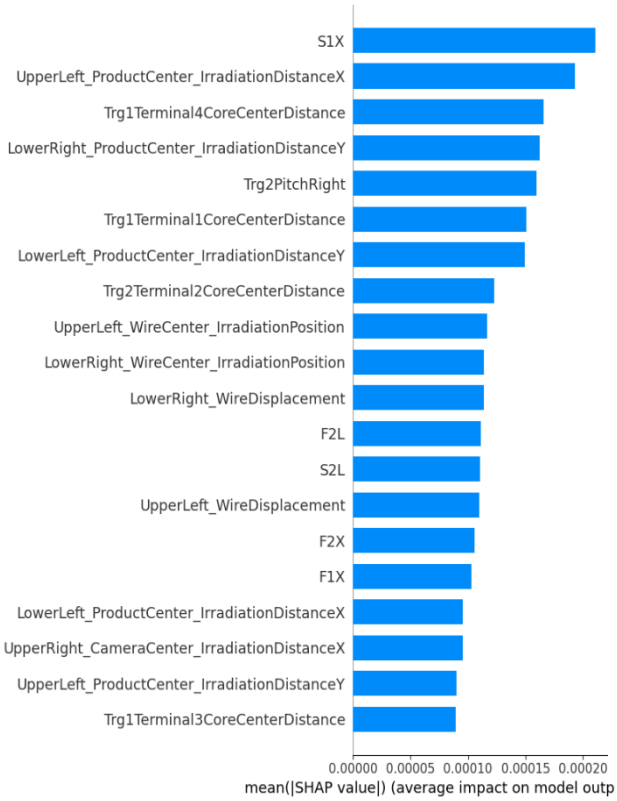
Time Series Model Output



- The training data shows considerable volatility with values fluctuating between approximately 0.09 and 0.11.
- There is a notable dip in the data around 01:00 where the defect rate drops to about 0.06.
- Both LSTM and ARIMA models make future predictions that appear to smooth out the volatility
- The LSTM prediction shows a slight upward trend while the ARIMA prediction maintains a more constant level.

Cause Analysis and Features affecting the Defect Rates

-



key features impacting the defect rate, ordered by their importance:
Top 5 Most Important Features:

1. S1X - Has the highest impact with a mean SHAP value of ~0.0020
2. UpperLeft_ProductCenter_IrradiationDistanceX - Second highest impact
3. Trg1Terminal4CoreCenterDistance
4. LowerRight_ProductCenter_IrradiationDistanceY
5. Trg2PitchRight

These features have the strongest influence on the model's predictions of defect rates. The values shown represent the average magnitude of their impact on the model output.
Key Observations:

- Position-related measurements (like S1X and various Irradiation Distance measurements) appear to be critical factors.
- Terminal distances (particularly Trg1Terminal4CoreCenterDistance and Trg1Terminal1CoreCenterDistance) play significant roles.
- Wire-related parameters (WireCenter_IrradiationPosition and WireDisplacement) have moderate importance.

Less Impactful Features:

- Trg1Terminal3CoreCenterDistance
- UpperLeft_ProductCenter_IrradiationDistanceY
- UpperRight_CameraCenter_IrradiationDistanceX

Advantages of the Approach

- Combines deep learning and statistical methods, offering robust forecasting capabilities.
- Provides a comprehensive evaluation and visualization of results.
- **Modularity:** Easy to debug, extend, or replace components.
- **Reusability:** Scripts can be reused for similar tasks in other projects.
- **Explainability**: SHAP analysis enhances model interpretability.
- **Automation:** Streamlined hyperparameter tuning and result saving processes reduce manual effort.

Limitations

- Hyperparameter tuning uses random search, which may miss global optima. Grid search or Bayesian optimization can be considered for critical tasks.
- Feature selection via RFE relies on linear regression; alternative methods like tree-based selectors can be explored.

Potential Enhancements

- Add hyperparameter tuning for both LSTM and ARIMA models.
- Introduce additional evaluation metrics like R-squared or MAE.
- Implement cross-validation for better performance assessment.
- Experiment with advanced LSTM architectures (e.g., BiLSTM, stacked LSTMs).
- Expand ARIMA functionality to include exogenous variables if applicable.

Conclusion

This pipeline demonstrates a robust, modular approach to machine learning. It combines rigorous preprocessing, systematic evaluation, and interpretability to produce reliable and actionable insights.

The manufacturing defects are most significantly influenced by positioning accuracy, particularly along the X-axis (S1X) and irradiation distances. Any quality improvement efforts should prioritize controlling these parameters to reduce defect rates.