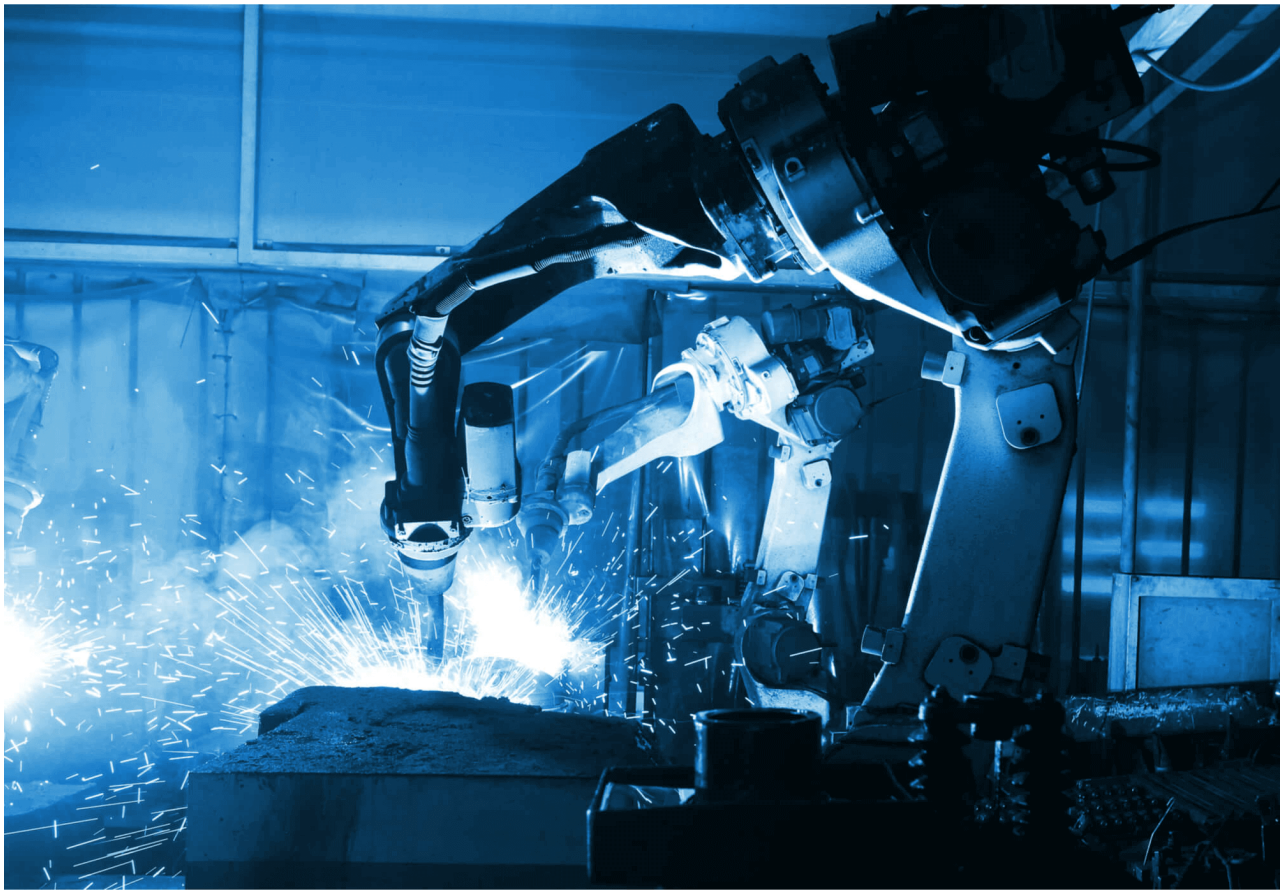
WELDRIGHT - TECHFEST



Team ID - HA-226489

Tharun Vemula

Manikanta Dommati

Abhinav Kalvacherla

Ashutosh Rai

Solution Approach

INTRODUCTION

An overview of the task at hand:

Welding is a critical activity for manufacturing. The threshold for welding error is the bare minimum.

In this challenge, participants will predict the welding defects in materials using data science methods, machine learning, and hyperparameter tuning.

Participants are expected to use ML models to predict welding defects in the materials by developing algorithms using the provided parameters. Participants are free to use any technique provided it is suited for the variety and volume of data provided.

DATA

Here we have a dataset with welding factors and welding defects. The dataset has 827534 Rows and 13 Columns. This is a multi-class classification problem where there are 3 major classes: No Defect, Porosity, and Tungsten Inclusion. The different types of features in the dataset include

1. Categorical variables
2. Continuous variables
3. Discrete variables
4. Binary variables
5. Numerical variables
6. Date - Time variables

- The feature "**Machine"** contains only one value of the machine in all the

rows. So there will be no effect on the target. Hence, this column is dropped.

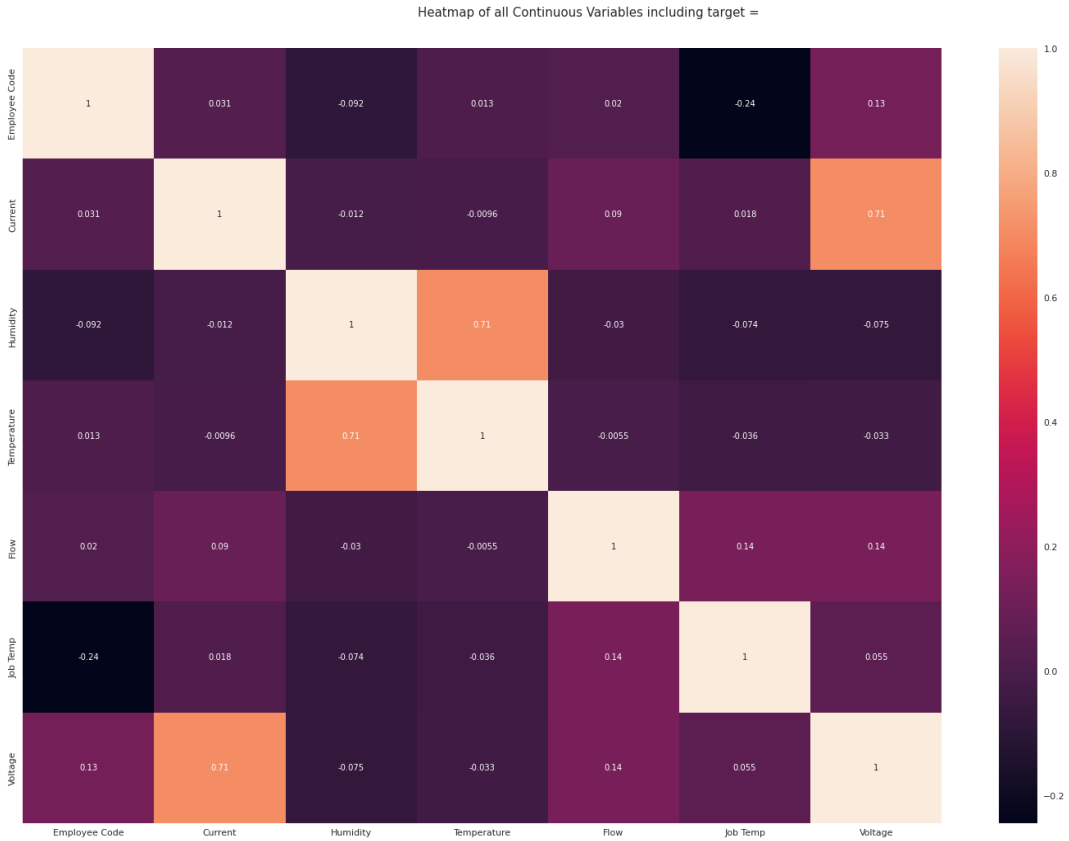
- A new feature is made based on the date time variable's time stamp, specifying if the process was performed during the **day** or **night**.

- The feature "**Production"** contains 278180 null values, constituting 33.6% of the total records. This feature can't be dropped and is, therefore, imputed based on the other features.

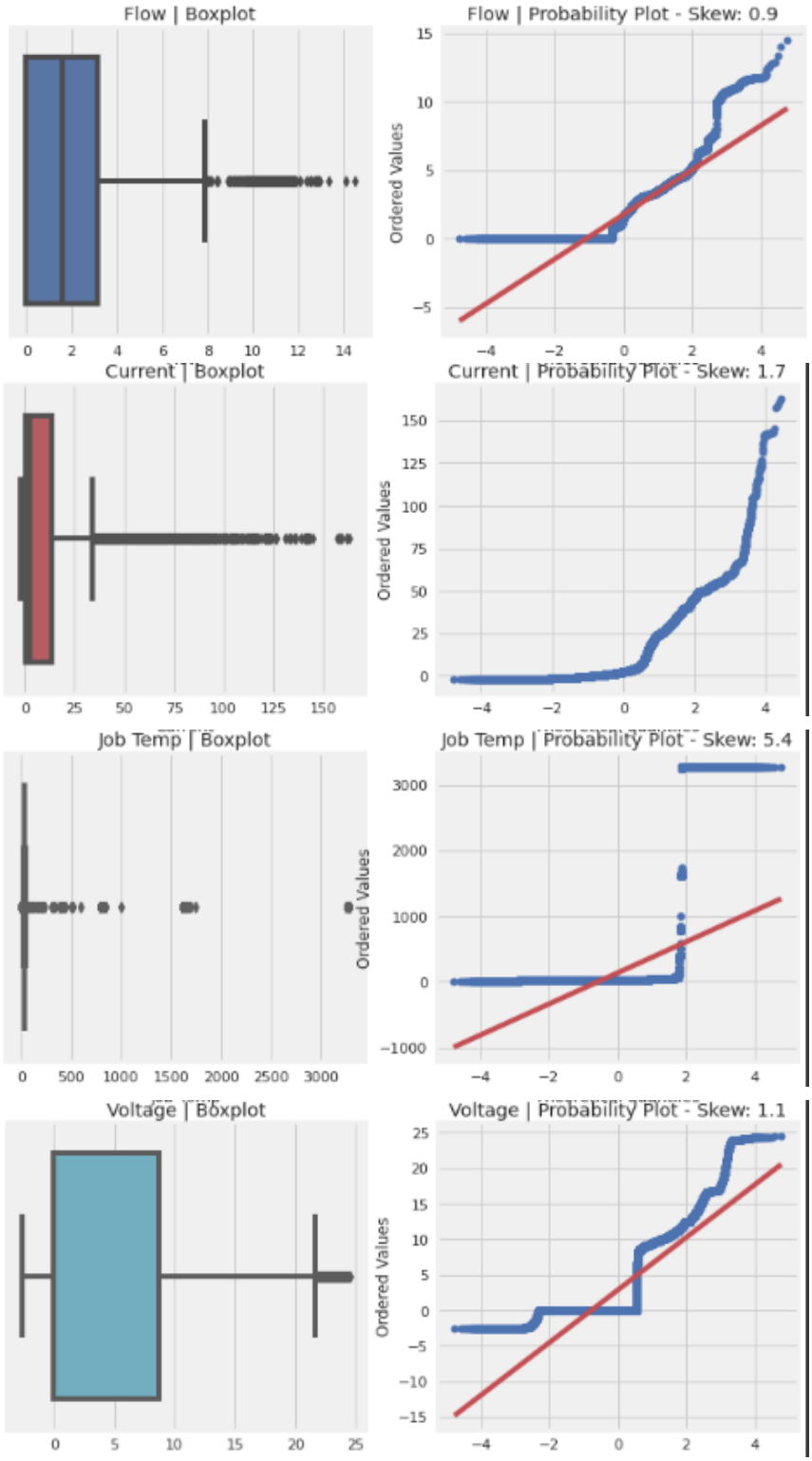
- Corrections are performed on the feature values for **"Order operation no."** and **"Defect"** column values and erroneous values are replaced.

- By observing the correlation heatmap conclude that no two features are highly correlated. So we can include all the features for target prediction.

- Here the target variable is of type string. We use a label encoder to transform the non-numerical labels into numerical labels. We scale the feature values using a MinMaxScaler.



Illustrations depicting the outliers and Non-Gaussian distributions for the numerical variables involved:



Model

The defect is classified under one of the 3 given classes, which are: **no defect**, **tungsten inclusion**, and **porosity**. Based on the number of samples we had and the number of output classes we can choose between the following classification models, we cannot, however, come to a conclusion unless we train and test the various different classifiers:

1. Random Forest Classifier.
2. K nearest Neighbours
3. XGB classifier

We need to try all the above models and should come to a conclusion based on the metric - **f1** score.

After experimenting with all the models, **Random Forest Classifier** comes out on top, outperforming the rest on the test data with an excellent f1.

Hyperparameter Tuning:

Hyperparameter tuning is important to find the best parameters that suit our classifier on the given data. Now hyper parameter tuning can be done using RandomizedSearchCV so we can find the best parameters for the random forest classifier.

RandomizedSearchCV permutates all the combinations of parameters and fits data based on the evaluation metrics best parameters are chosen.

We can tune any parameters that we want and find the best parameters for our model.

```
[ ]  rf_params={
         "max_depth": [5,8,15,None,10],
         "max_features": [5,7,"auto",8,10,'sqrt','log2'],
         "min_samples_split": [2,8,15,20],
         "n_estimators": [100,200,500,1000]
     }

     random_search_cv=[
         ("RF",RandomForestClassifier(), rf_params)
     ]

     model_param={}
     for name,model,params in random_search_cv:
       random= RandomizedSearchCV(
           estimator=model,
           param_distributions=params,
           n_iter=100,
           cv=2,
           verbose=2,
           n_jobs=-1
       )
       random.fit(Xt,Yt)
       model_param[name]=random.best_params_

     for model_name in model_param:
       print(f"Best parameters for {model_name}:\n")
       print(model_param[model_name])
```

```
[ ]  best_params=model_param["RF"]
     model=RandomForestClassifier().set_params(**best_params)
     final_prediction=clf.predict(x)
     print(f"f1 score: {f1_score(y,final_prediction)}")
```

METRICS

- Metrics are used to evaluate our model and the performance of our model can be estimated using metrics. As this data is highly imbalanced and is a classification problem we need to use the f1 score as a metric for model evaluation.

- F1 score is calculated based on the precision score and recall score for every class which will be the correct evaluation metric for the imbalanced dataset.

- We cannot use accuracy as an evaluation metric because there is an accuracy paradox for imbalanced data. We may get high accuracy even if our model is not predicting well or just guessing among classes.

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

	precision	recall	f1-score	support
0	0.90	0.80	0.85	205597
1	0.83	0.93	0.88	204682
2	0.99	0.97	0.98	205502
accuracy			0.90	615781
macro avg	0.91	0.90	0.90	615781
weighted avg	0.91	0.90	0.90	615781

RETURN ON INVESTMENT

Return on Investment (ROI) is a financial ratio of an investment's gain or loss relative to its cost. In its simplest form, the benefits should outweigh the costs when investments are made in AI.

Classification Accuracy is the ratio of the number of correctly predicted to the total number of input samples. The formula below estimates the profit per prediction:

$$\hat{a} = (a - (1 - I) * e)$$

Where $\hat{a}$ denotes adjusted saving (profit per prediction), a represents the expected saving, I is the computed average accuracy (we get that from training a model) and e is the cost of manually fixing a mistake.

To get the adjusted savings $\hat{a}$, we have to account for the ratio from the incorrectly predicted (1 – accuracy) to the cost of making a mistake. The adjusted savings give us the actual savings after removing the number of errors. However, the simplicity of the above equation comes with a high risk, as we rely entirely on the algorithm's performance.

The steps to achieve a more robust estimation algorithm are just slightly more involved than the previous equation, which includes confidence scores as follows,

Our model has $\hat{a} = (((a - (1 - \hat{I}) * e) * \varkappa) - ((1 - \varkappa) * \hat{e}))$ an accuracy of **99.93%**.

Let's assume the following values,

$a = 5\,minutes,\ e = 35\,minutes,\ I = 99.93,\ \varkappa = 0.90,\ \hat{e} = 5\,minutes$

Applying the above values in the equation gives,

â = 4.497

Hence 4.497 minutes would be saved per claim.