Table of Contents

Task description

The goal is to develop a model capable of accurately predicting the likelihood of different types of defects in steel plates. This task utilizes a dataset synthesized from a deep learning model trained on the UCI Steel Plates Faults dataset.

Evaluation Criteria: The models will be evaluated based on the Area Under the ROC Curve (AUC) metric. The final score is the average of the individual AUCs for each of the seven predicted defect categories.

IMPORTS

In [1]:
```
import numpy as np
import pandas as pd
import catboost as cb
import lightgbm as lgb
import seaborn as sns
from scipy.stats import skew
import matplotlib.pyplot as plt
from sklearn.preprocessing import RobustScaler
from sklearn.metrics import accuracy_score, roc_auc_score, f1_score, precision_score, re
from sklearn.model_selection import train_test_split
from imblearn.combine import SMOTETomek
from imblearn.under_sampling import TomekLinks
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
pd.pandas.set_option('display.max_columns',None)
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/playground-series-s4e3/sample_submission.csv
/kaggle/input/playground-series-s4e3/train.csv
/kaggle/input/playground-series-s4e3/test.csv
/kaggle/input/steel-plate-faults-orig/faults.csv
```

In [2]:
```
train=pd.read_csv("/kaggle/input/playground-series-s4e3/train.csv")
```

```
test=pd.read_csv("/kaggle/input/playground-series-s4e3/test.csv")
orig=pd.read_csv("/kaggle/input/steel-plate-faults-orig/faults.csv")

train = train.drop(['id'], axis=1)
```

Dataset overview

In [3]:
```
print('Train shape:', train.shape)
print('Original shape:', orig.shape)
print('Test shape: ', test.shape)
```

```
Train shape: (19219, 34)
Original shape: (1941, 34)
Test shape:  (12814, 28)
```

In [4]:
```
train.head()
```

Out[4]:

	X_Minimum	X_Maximum	Y_Minimum	Y_Maximum	Pixels_Areas	X_Perimeter	Y_Perimeter	Sum_of_Luminosit
0	584	590	909972	909977	16	8	5	227
1	808	816	728350	728372	433	20	54	4447
2	39	192	2212076	2212144	11388	705	420	131139
3	781	789	3353146	3353173	210	16	29	320
4	1540	1560	618457	618502	521	72	67	4823

In [5]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19219 entries, 0 to 19218
Data columns (total 34 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   X_Minimum              19219 non-null  int64  
 1   X_Maximum              19219 non-null  int64  
 2   Y_Minimum              19219 non-null  int64  
 3   Y_Maximum              19219 non-null  int64  
 4   Pixels_Areas           19219 non-null  int64  
 5   X_Perimeter            19219 non-null  int64  
 6   Y_Perimeter            19219 non-null  int64  
 7   Sum_of_Luminosity      19219 non-null  int64  
 8   Minimum_of_Luminosity  19219 non-null  int64  
 9   Maximum_of_Luminosity  19219 non-null  int64  
 10  Length_of_Conveyer     19219 non-null  int64  
 11  TypeOfSteel_A300       19219 non-null  int64  
 12  TypeOfSteel_A400       19219 non-null  int64  
 13  Steel_Plate_Thickness  19219 non-null  int64  
 14  Edges_Index            19219 non-null  float64
 15  Empty_Index            19219 non-null  float64
 16  Square_Index           19219 non-null  float64
 17  Outside_X_Index        19219 non-null  float64
 18  Edges_X_Index          19219 non-null  float64
 19  Edges_Y_Index          19219 non-null  float64
 20  Outside_Global_Index   19219 non-null  float64
 21  LogOfAreas             19219 non-null  float64
 22  Log_X_Index            19219 non-null  float64
 23  Log_Y_Index            19219 non-null  float64
 24  Orientation_Index      19219 non-null  float64
```

```
 25  Luminosity_Index        19219 non-null  float64
 26  SigmoidOfAreas          19219 non-null  float64
 27  Pastry                  19219 non-null  int64
 28  Z_Scratch               19219 non-null  int64
 29  K_Scatch                19219 non-null  int64
 30  Stains                  19219 non-null  int64
 31  Dirtiness               19219 non-null  int64
 32  Bumps                   19219 non-null  int64
 33  Other_Faults            19219 non-null  int64
dtypes: float64(13), int64(21)
memory usage: 5.0 MB
```

In [6]:
```
train.describe().T
```

Out[6]:

	count	mean	std	min	25%	50%	75
X_Minimum	19219.0	7.098547e+02	5.315442e+02	0.0000	49.00000	7.770000e+02	1.152000e+
X_Maximum	19219.0	7.538576e+02	4.998366e+02	4.0000	214.00000	7.960000e+02	1.165000e+
Y_Minimum	19219.0	1.849756e+06	1.903554e+06	6712.0000	657468.00000	1.398169e+06	2.368032e+
Y_Maximum	19219.0	1.846605e+06	1.896295e+06	6724.0000	657502.00000	1.398179e+06	2.362511e+
Pixels_Areas	19219.0	1.683988e+03	3.730320e+03	6.0000	89.00000	1.680000e+02	6.530000e+
X_Perimeter	19219.0	9.565466e+01	1.778214e+02	2.0000	15.00000	2.500000e+01	6.400000e+
Y_Perimeter	19219.0	6.412410e+01	1.010542e+02	1.0000	14.00000	2.300000e+01	6.100000e+
Sum_of_Luminosity	19219.0	1.918467e+05	4.420247e+05	250.0000	9848.00000	1.823800e+04	6.797800e+
Minimum_of_Luminosity	19219.0	8.480842e+01	2.880034e+01	0.0000	70.00000	9.000000e+01	1.050000e+
Maximum_of_Luminosity	19219.0	1.286474e+02	1.419698e+01	39.0000	124.00000	1.270000e+02	1.350000e+
Length_of_Conveyer	19219.0	1.459351e+03	1.455687e+02	1227.0000	1358.00000	1.364000e+03	1.652000e+
TypeOfSteel_A300	19219.0	4.026744e-01	4.904490e-01	0.0000	0.00000	0.000000e+00	1.000000e+
TypeOfSteel_A400	19219.0	5.963370e-01	4.906442e-01	0.0000	0.00000	1.000000e+00	1.000000e+
Steel_Plate_Thickness	19219.0	7.621312e+01	5.393196e+01	40.0000	40.00000	6.900000e+01	8.000000e+
Edges_Index	19219.0	3.529394e-01	3.189760e-01	0.0000	0.05860	2.385000e-01	6.561000e-
Empty_Index	19219.0	4.093095e-01	1.241435e-01	0.0000	0.31750	4.135000e-01	4.946000e-
Square_Index	19219.0	5.745204e-01	2.594359e-01	0.0083	0.37575	5.454000e-01	8.182000e-
Outside_X_Index	19219.0	3.060936e-02	4.730194e-02	0.0015	0.00660	9.500000e-03	1.910000e-
Edges_X_Index	19219.0	6.147495e-01	2.223913e-01	0.0144	0.45160	6.364000e-01	7.857000e-
Edges_Y_Index	19219.0	8.316521e-01	2.209660e-01	0.1050	0.65520	9.643000e-01	1.000000e+
Outside_Global_Index	19219.0	5.918986e-01	4.820500e-01	0.0000	0.00000	1.000000e+00	1.000000e+
LogOfAreas	19219.0	2.473475e+00	7.605751e-01	0.7782	1.94940	2.227900e+00	2.814900e+
Log_X_Index	19219.0	1.312667e+00	4.678477e-01	0.3010	1.00000	1.146100e+00	1.431400e+
Log_Y_Index	19219.0	1.389737e+00	4.055493e-01	0.0000	1.07920	1.322200e+00	1.707600e+
Orientation_Index	19219.0	1.027423e-01	4.876805e-01	-0.9884	-0.27270	1.111000e-01	5.294000e-
Luminosity_Index	19219.0	-1.383818e-01	1.203440e-01	-0.8850	-0.19250	-1.426000e-01	-8.40000C
SigmoidOfAreas	19219.0	5.719022e-01	3.322186e-01	0.1190	0.25320	4.729000e-01	9.994000e-
Pastry	19219.0	7.627868e-02	2.654504e-01	0.0000	0.00000	0.000000e+00	0.000000e+

Z_Scratch	19219.0	5.983662e-02	2.371901e-01	0.0000	0.00000	0.000000e+00	0.000000e+
K_Scatch	19219.0	1.785733e-01	3.830046e-01	0.0000	0.00000	0.000000e+00	0.000000e+
Stains	19219.0	2.955409e-02	1.693580e-01	0.0000	0.00000	0.000000e+00	0.000000e+
Dirtiness	19219.0	2.523544e-02	1.568435e-01	0.0000	0.00000	0.000000e+00	0.000000e+
Bumps	19219.0	2.478277e-01	4.317625e-01	0.0000	0.00000	0.000000e+00	0.000000e+
Other_Faults	19219.0	3.412248e-01	4.741330e-01	0.0000	0.00000	0.000000e+00	1.000000e+

In [7]:

```
print('Train NONE elements:', train.isnull().sum().sum())
print('Orig NONE elements:', orig.isnull().sum().sum())
print('Test NONE elements:', test.isnull().sum().sum())
```

```
Train NONE elements: 0
Orig NONE elements: 0
Test NONE elements: 0
```

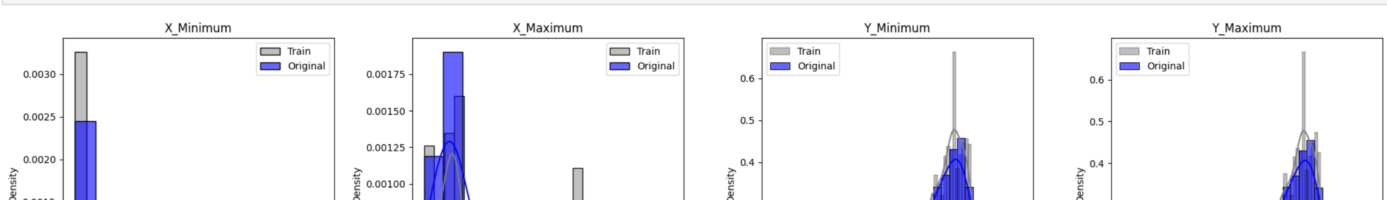
The dataset is prepped for exploratory data analysis (EDA) with no missing values and only one categorical feature: Steel Type.

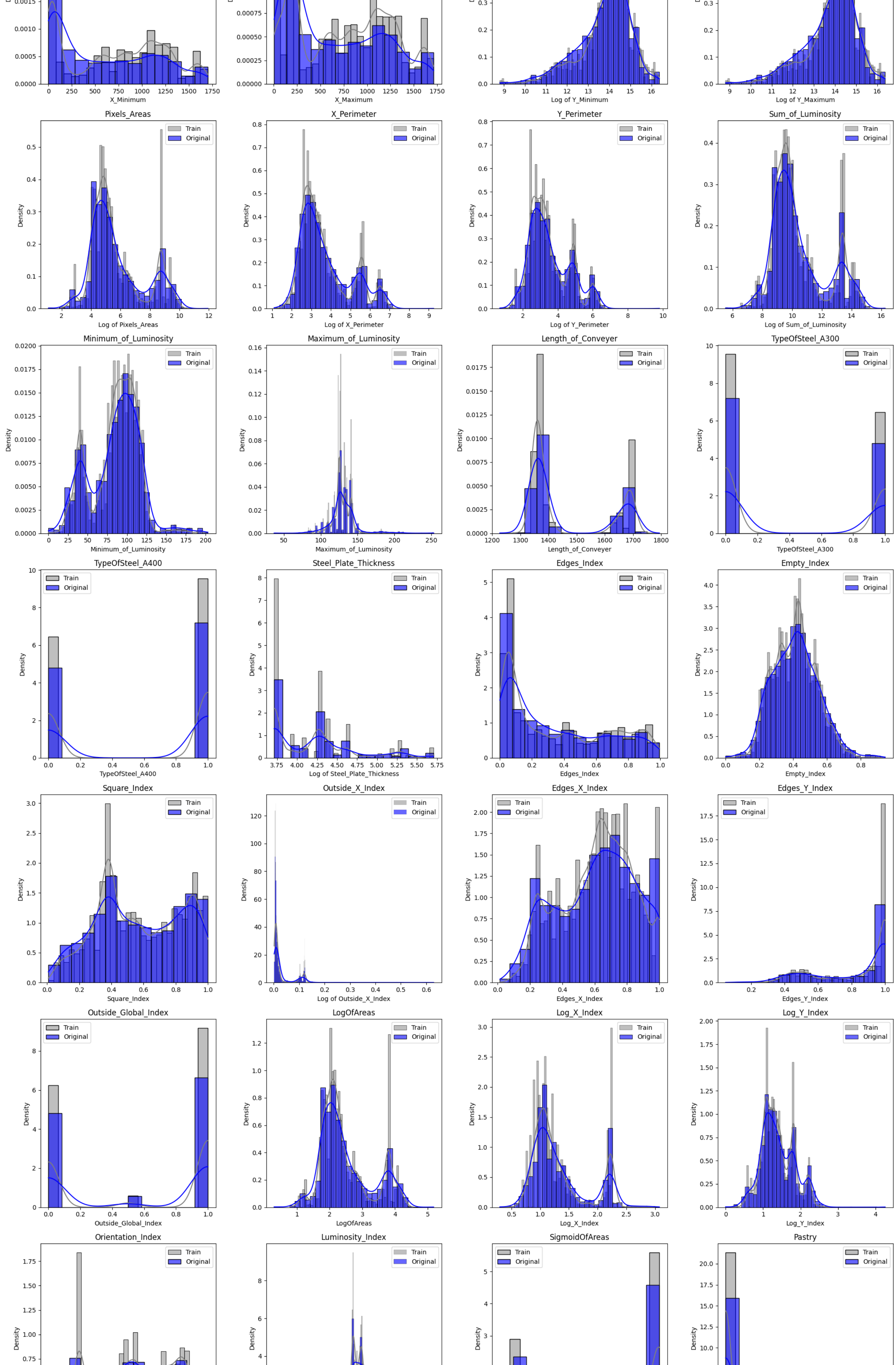
Dataset analysis (EDA)

Let's start from comparation a training dataset with the original.

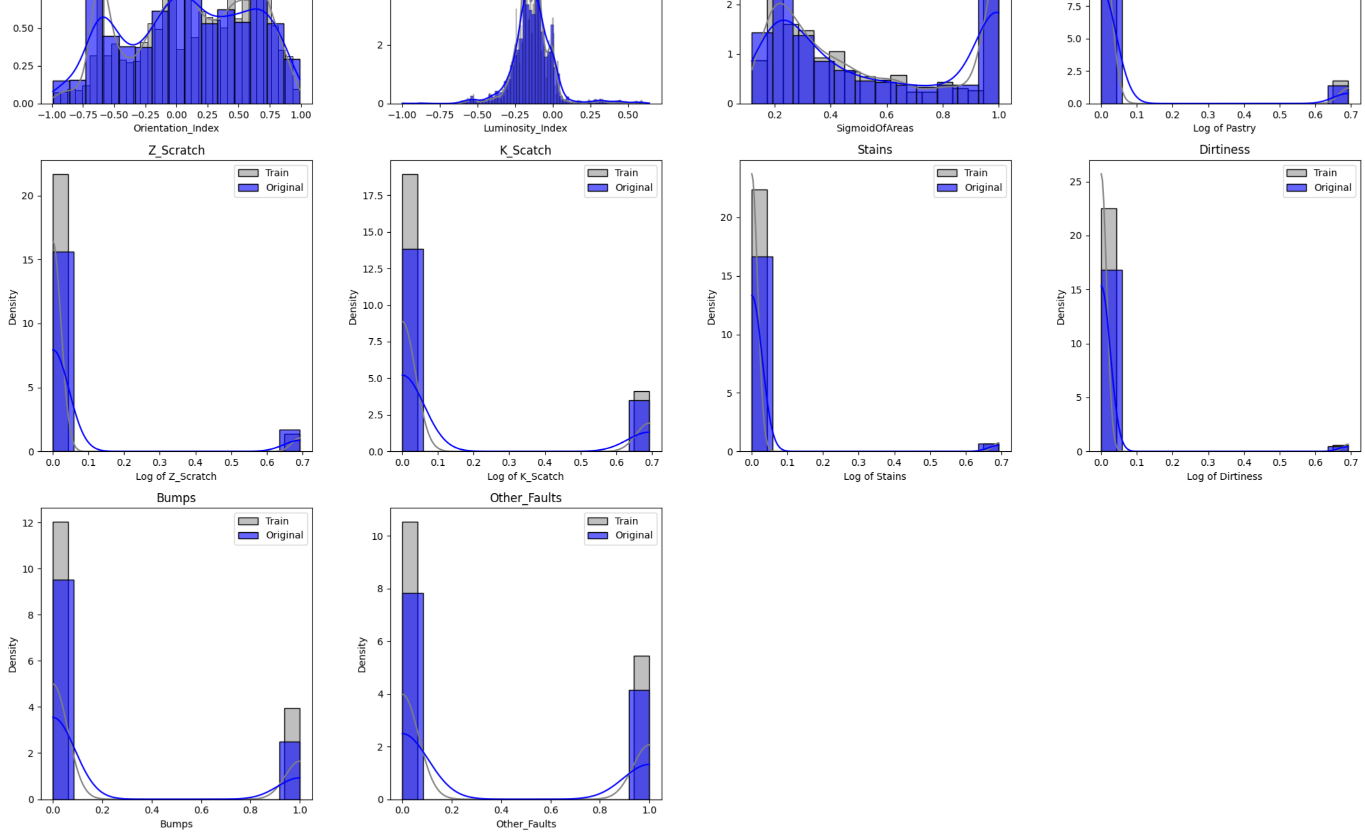
In [8]:

```
columns = train.columns
n_cols = 4
n_rows = int(np.ceil(len(columns) / n_cols))
skewness_threshold = 1.5

fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, 5*n_rows))
axes = axes.flatten()

for i, col in enumerate(columns):
    if i < len(axes):
        if skew(train[col].dropna()) > skewness_threshold:
            train_data_transformed = np.log1p(train[col])
            orig_data_transformed = np.log1p(orig[col])
            xlabel = f'Log of {col}'
        else:
            train_data_transformed = train[col]
            orig_data_transformed = orig[col]
            xlabel = col
        sns.histplot(train_data_transformed, color="grey", label="Train", kde=True, ax=a
        sns.histplot(orig_data_transformed, color="blue", label="Original", kde=True, ax
        axes[i].set_title(col)
        axes[i].legend()
        axes[i].set_xlabel(xlabel)
for i in range(len(columns), len(axes)):
    axes[i].axis('off')
plt.tight_layout()
plt.show()
```



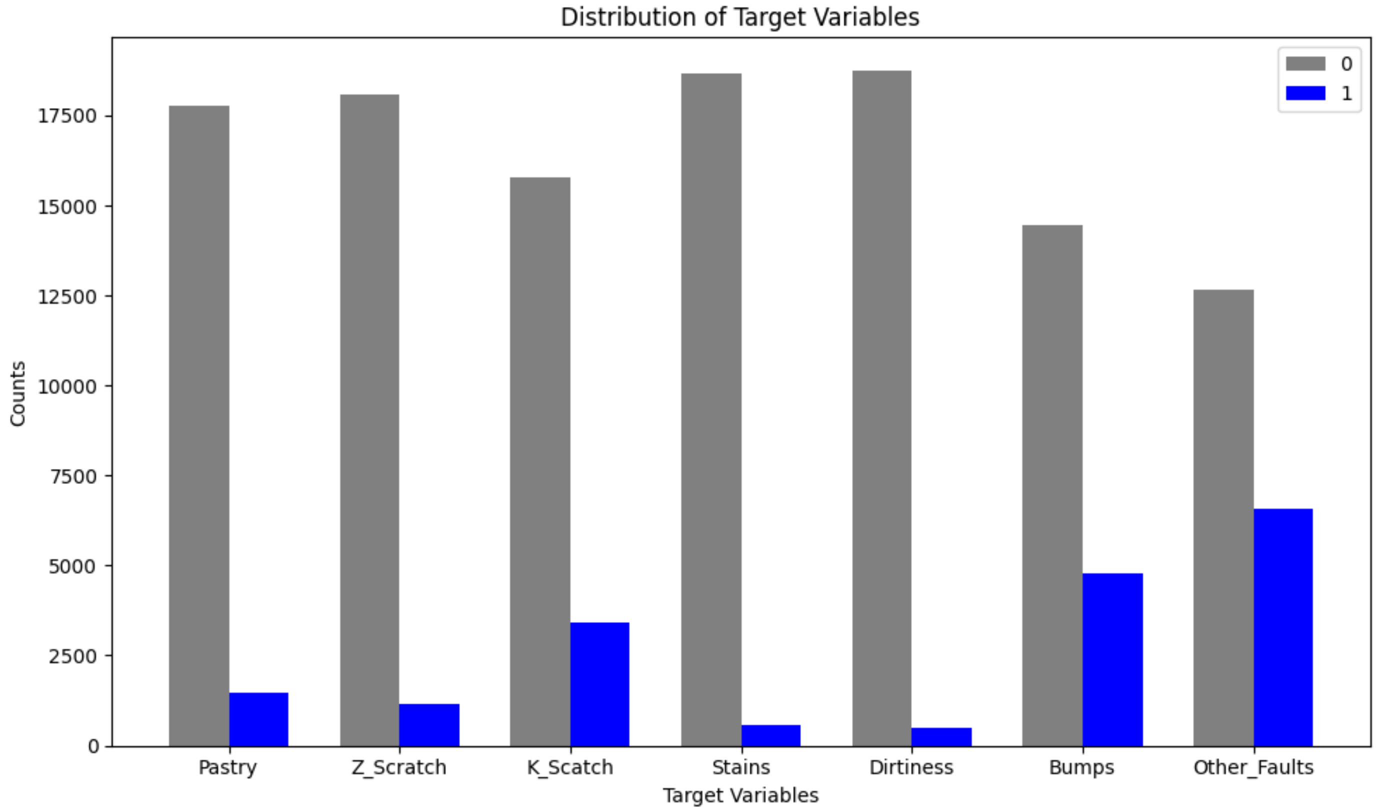


X_Minimum
X_Maximum
Log of Y_Minimum
Log of Y_Maximum
Pixels_Areas
Log of Pixels_Areas
X_Perimeter
Log of X_Perimeter
Y_Perimeter
Log of Y_Perimeter
Sum_of_Luminosity
Log of Sum_of_Luminosity
Minimum_of_Luminosity
Maximum_of_Luminosity
Length_of_Conveyer
TypeOfSteel_A300
TypeOfSteel_A400
Steel_Plate_Thickness
Log of Steel_Plate_Thickness
Edges_Index
Empty_Index
Square_Index
Outside_X_Index
Log of Outside_X_Index
Edges_X_Index
Edges_Y_Index
Outside_Global_Index
LogOfAreas
Log_X_Index
Log_Y_Index
Orientation_Index
Luminosity_Index
SigmoidOfAreas
Pastry
Density
Train
Original



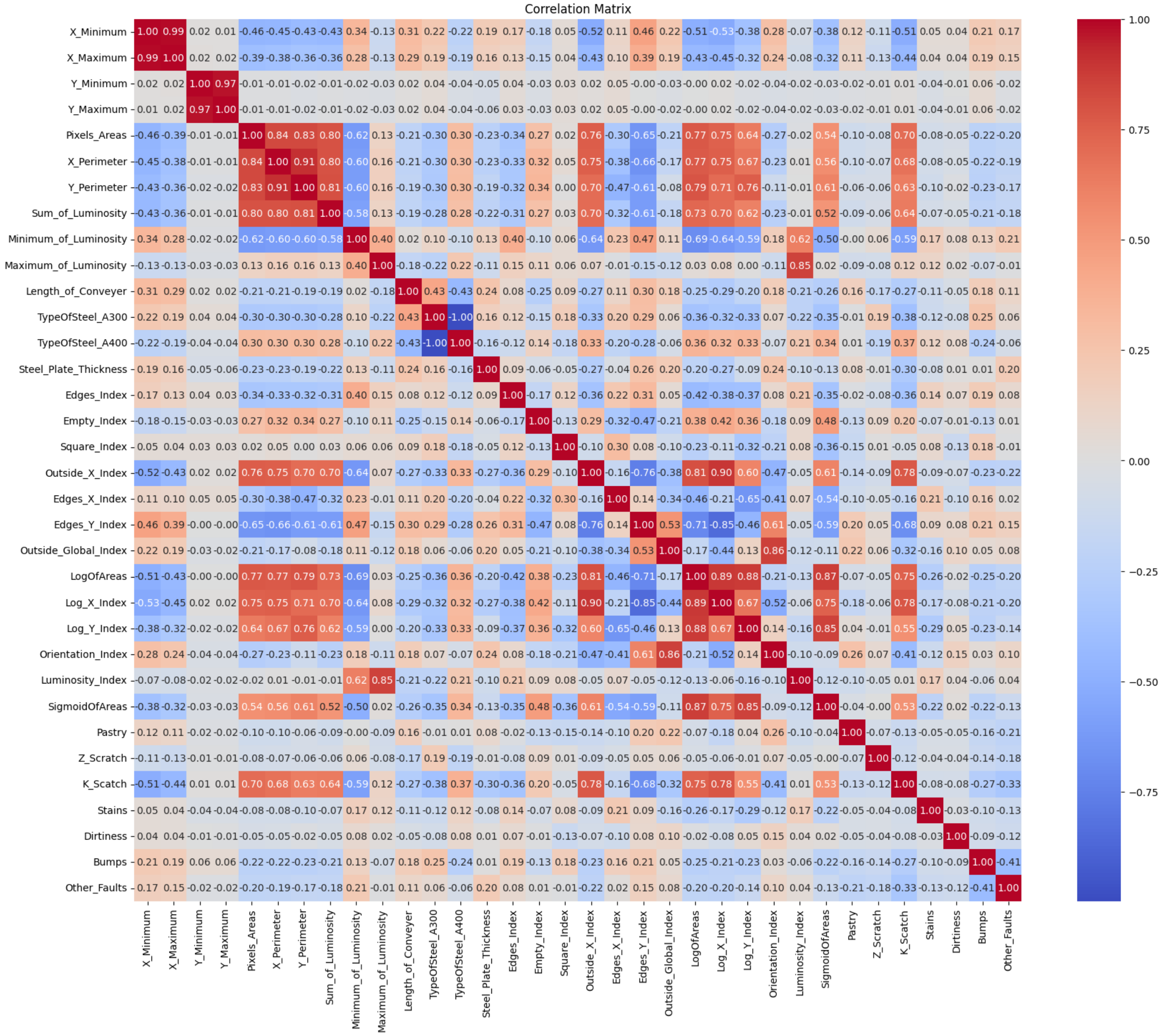
Comparison of the training dataset with the original reveals that, while the training set often displays higher maximum density values, the patterns remain strikingly similar, justifying the combination of both datasets.

In [9]:
```
target_col = ['Pastry', 'Z_Scratch', 'K_Scatch', 'Stains', 'Dirtiness', 'Bumps', 'Other_

fig, ax = plt.subplots(figsize=(10, 6))
bar_width = 0.35
index = np.arange(len(target_col))

values_0 = []
values_1 = []

for col in target_col:
    counts = train[col].value_counts(normalize=False)
    values_0.append(counts.get(0, 0))
    values_1.append(counts.get(1, 0))

bars_0 = ax.bar(index - bar_width/2, values_0, bar_width, label='0', color='gray')
bars_1 = ax.bar(index + bar_width/2, values_1, bar_width, label='1', color='blue')

ax.set_xlabel('Target Variables')
ax.set_ylabel('Counts')
ax.set_title('Distribution of Target Variables')
ax.set_xticks(index)
ax.set_xticklabels(target_col)
ax.legend()

plt.tight_layout()
plt.show()
```



A notable imbalance in target distribution necessitates minority class augmentation.

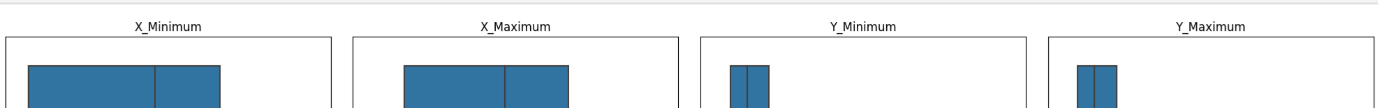
In [10]:
```
numerical_features = train.select_dtypes(include=['int64', 'float64']).columns.tolist()
plt.figure(figsize=(20, 16))
sns.heatmap(train[numerical_features].corr(), annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```

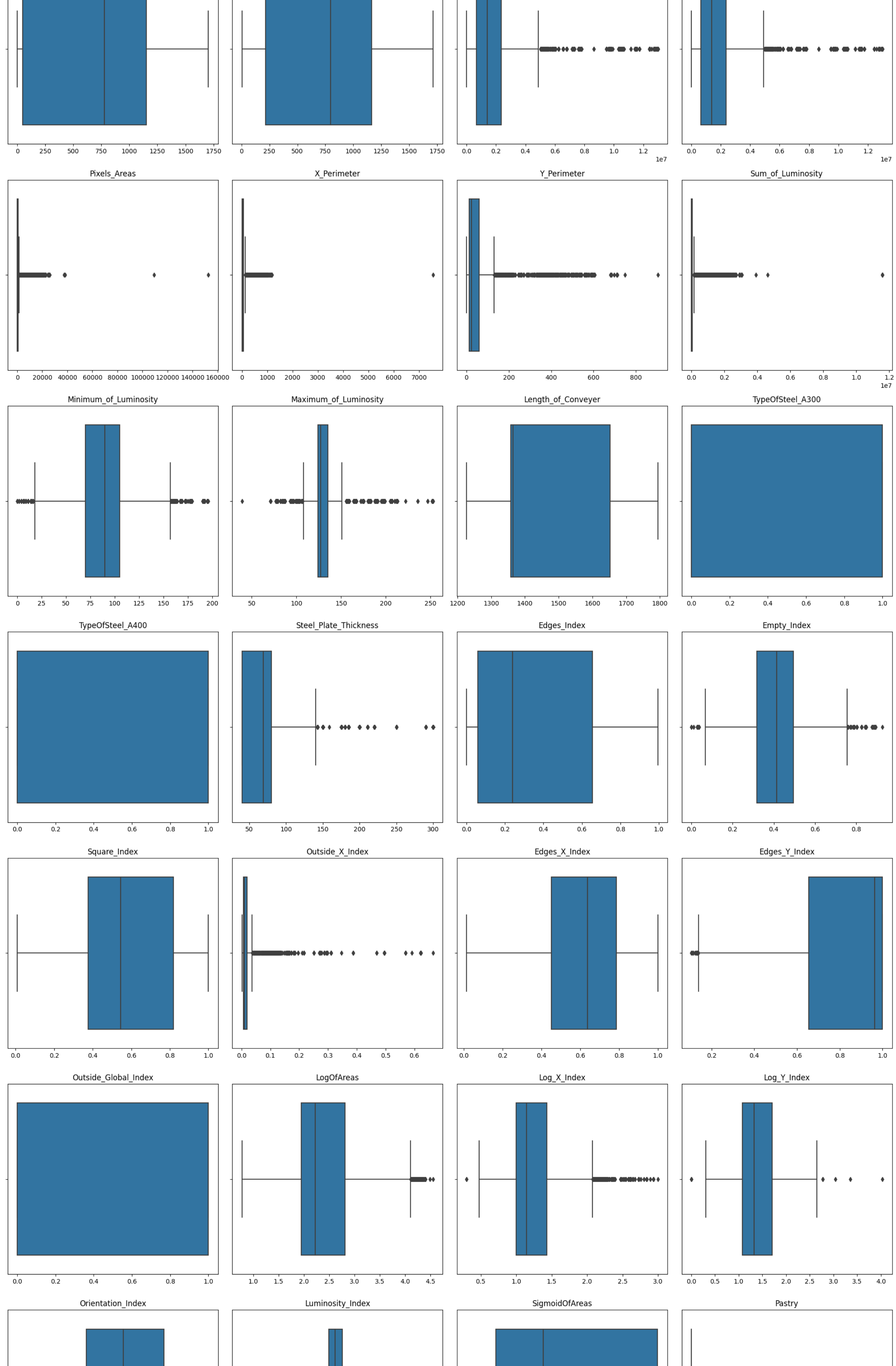


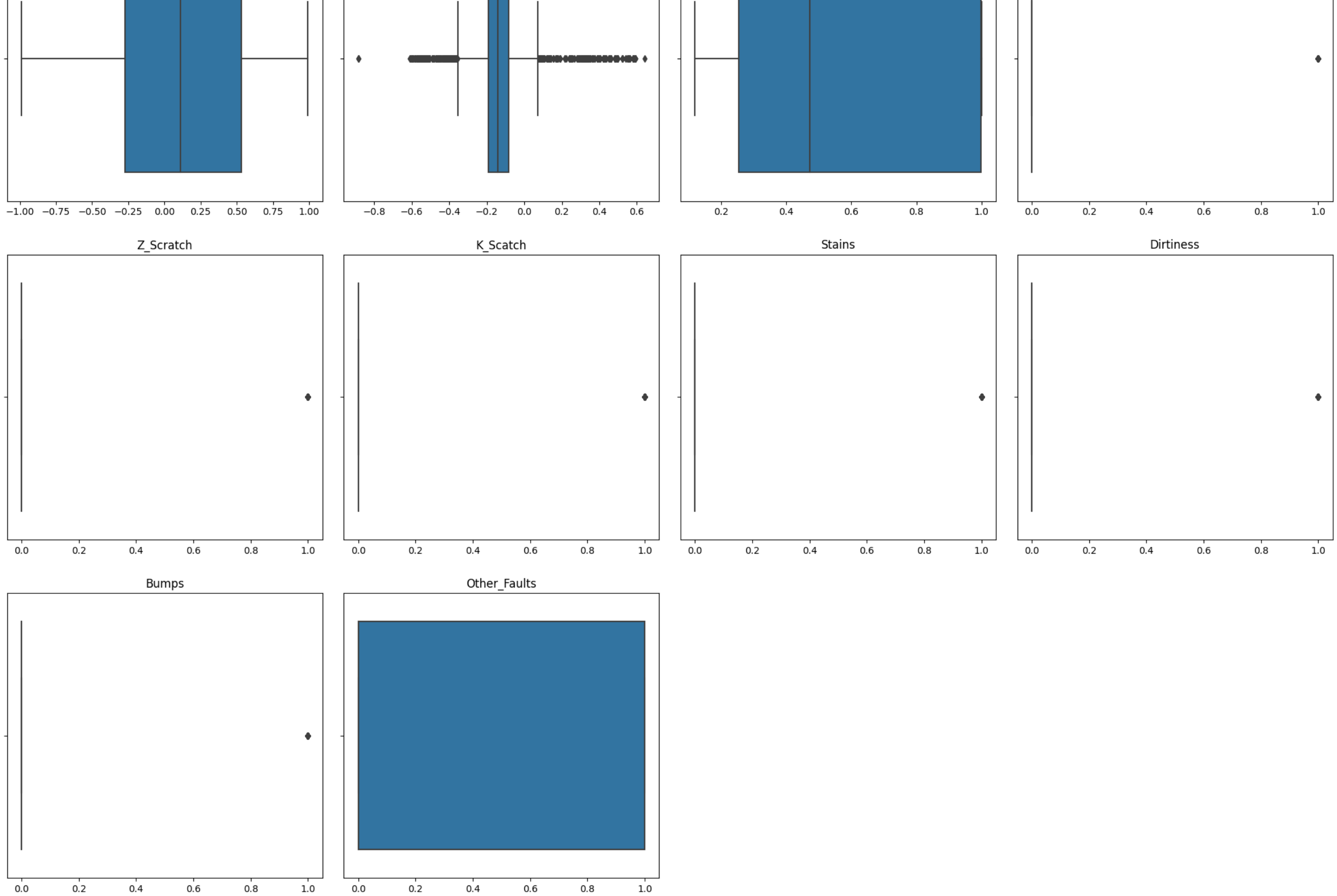
Significant correlations among many features warrant attention.

In [11]:

```
columns = train.columns
n_cols = 4
n_rows = int(np.ceil(len(columns) / n_cols))

fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, 5 * n_rows))
axes = axes.flatten()

for i, col in enumerate(columns):
    if i < len(axes):
        sns.boxplot(data=train, x=col, ax=axes[i])
        axes[i].set_title(col, fontsize=12)
        axes[i].tick_params(labelsize=10)
        axes[i].set_xlabel('')
        axes[i].set_ylabel('')

for i in range(len(columns), len(axes)):
    axes[i].set_visible(False)

plt.tight_layout()
plt.show()
```




Pixels_Areas
X_Perimeter
Y_Perimeter
Sum_of_Luminosity
Minimum_of_Luminosity
Maximum_of_Luminosity
Length_of_Conveyer
TypeOfSteel_A300
TypeOfSteel_A400
Steel_Plate_Thickness
Edges_Index
Empty_Index
Square_Index
Outside_X_Index
Edges_X_Index
Edges_Y_Index
Outside_Global_Index
LogOfAreas
Log_X_Index
Log_Y_Index
Orientation_Index
Luminosity_Index
SigmoidOfAreas
Pastry



Outliers present in numerous features suggest the need for a non-standard scaling approach to mitigate their impact.

Preprocessing, data transformation, features engineering

Informed by EDA insights, the preprocessing strategy will include:

- Merging the original dataset with the training data.
- Implementing Robust Scaler to address outliers.
- Feature reduction to improve model efficiency.
- Employing SMOTE to balance class distribution.
- Opting for various boosting algorithms known for their efficacy in classification tasks.
- Tailoring individual models for each target label, necessitating a comprehensive approach to manage dataset specifics for each model.

In [12]:
```
validation_ids = test['id']
test = test.drop(['id', 'X_Maximum', 'Y_Maximum', 'Y_Perimeter', 'Sum_of_Luminosity', 'T

train = pd.concat([train, orig], axis=0)
train = train.drop(['X_Maximum', 'Y_Maximum', 'Y_Perimeter', 'Sum_of_Luminosity', 'TypeO

X = train.drop(target_col, axis=1)
train_y = train[target_col]
```

In [13]:

```
smt = SMOTETomek(random_state=42)
scaler = RobustScaler()

# 1
X_train_Pastry, X_test_Pastry, y_train_Pastry, y_test_Pastry = train_test_split(X, train
X_train_Pastry_scaled = scaler.fit_transform(X_train_Pastry)
X_test_Pastry_scaled = scaler.transform(X_test_Pastry)
X_train_Pastry_smote, y_train_Pastry_smote = smt.fit_resample(X_train_Pastry_scaled, y_t
print(y_train_Pastry_smote.value_counts())

# 2
X_train_Z_Scratch, X_test_Z_Scratch, y_train_Z_Scratch, y_test_Z_Scratch = train_test_sp
X_train_Z_Scratch_scaled = scaler.fit_transform(X_train_Z_Scratch)
X_test_Z_Scratch_scaled = scaler.transform(X_test_Z_Scratch)
X_train_Z_Scratch_smote, y_train_Z_Scratch_smote = smt.fit_resample(X_train_Z_Scratch_sc
print(y_train_Z_Scratch_smote.value_counts())

# 3
X_train_K_Scatch, X_test_K_Scatch, y_train_K_Scatch, y_test_K_Scatch = train_test_split(
X_train_K_Scatch_scaled = scaler.fit_transform(X_train_K_Scatch)
X_test_K_Scatch_scaled = scaler.transform(X_test_K_Scatch)
X_train_K_Scatch_smote, y_train_K_Scatch_smote = smt.fit_resample(X_train_K_Scatch_scale
print(y_train_K_Scatch_smote.value_counts())

# 4
X_train_Stains, X_test_Stains, y_train_Stains, y_test_Stains = train_test_split(X, train
X_train_Stains_scaled = scaler.fit_transform(X_train_Stains)
X_test_Stains_scaled = scaler.transform(X_test_Stains)
X_train_Stains_smote, y_train_Stains_smote = smt.fit_resample(X_train_Stains_scaled, y_t
print(y_train_Stains_smote.value_counts())

# 5
X_train_Dirtiness, X_test_Dirtiness, y_train_Dirtiness, y_test_Dirtiness = train_test_sp
X_train_Dirtiness_scaled = scaler.fit_transform(X_train_Dirtiness)
X_test_Dirtiness_scaled = scaler.transform(X_test_Dirtiness)
X_train_Dirtiness_smote, y_train_Dirtiness_smote = smt.fit_resample(X_train_Dirtiness_sc
print(y_train_Dirtiness_smote.value_counts())

# 6
X_train_Bumps, X_test_Bumps, y_train_Bumps, y_test_Bumps = train_test_split(X, train_y['
X_train_Bumps_scaled = scaler.fit_transform(X_train_Bumps)
X_test_Bumps_scaled = scaler.transform(X_test_Bumps)
X_train_Bumps_smote, y_train_Bumps_smote = smt.fit_resample(X_train_Bumps_scaled, y_trai
print(y_train_Bumps_smote.value_counts())

# 7
X_train_Other_Faults, X_test_Other_Faults, y_train_Other_Faults, y_test_Other_Faults = t
X_train_Other_Faults_scaled = scaler.fit_transform(X_train_Other_Faults)
X_test_Other_Faults_scaled = scaler.transform(X_test_Other_Faults)
X_train_Other_Faults_smote, y_train_Other_Faults_smote = smt.fit_resample(X_train_Other_
print(y_train_Other_Faults_smote.value_counts())
```

```
Pastry
0    15612
1    15612
Name: count, dtype: int64
Z_Scratch
0    15844
1    15844
Name: count, dtype: int64
K_Scatch
0    13841
1    13841
Name: count, dtype: int64
Stains
```

```
0    16420
1    16420
Name: count, dtype: int64
Dirtiness
1    16485
0    16485
Name: count, dtype: int64
Bumps
0    12654
1    12654
Name: count, dtype: int64
Other_Faults
0    10845
1    10845
Name: count, dtype: int64
```

Model training

Model hyperparameters were finely tuned using Optuna, ensuring optimal performance during the tests.

In [14]:
```
lgbm_model_Pastry = lgb.LGBMClassifier(objective ='binary',
                                       boosting_type ='gbdt',
                                       data_sample_strategy ="goss",
                                       metric="auc",
                                       class_weight="balanced",
                                       colsample_bytree=0.1,
                                       subsample=0.7,
                                       learning_rate=0.16,
                                       max_depth=11,
                                       n_estimators=3000,
                                       num_leaves=230,
                                       reg_alpha=0.8,
                                       reg_lambda=0.9,
                                       verbose=-1,
                                       min_child_samples=24,
                                       random_state=42)

lgbm_model_Pastry.fit(X_train_Pastry_smote, y_train_Pastry_smote)
y_pred_Pastry = lgbm_model_Pastry.predict(X_test_Pastry_scaled)
accuracy_Pastry = accuracy_score(y_test_Pastry, y_pred_Pastry)
print(f"Accuracy: {accuracy_Pastry}")
print(classification_report(y_test_Pastry, y_pred_Pastry))

y_train_Pastry_pred_proba = lgbm_model_Pastry.predict_proba(X_test_Pastry_scaled)[:, 1]
roc_auc_Pastry = roc_auc_score(y_test_Pastry, y_train_Pastry_pred_proba)
print(f"AUC-ROC score for the training dataset: {roc_auc_Pastry:.2f}")
```

```
Accuracy: 0.915406427221172
              precision    recall  f1-score   support

           0       0.94      0.97      0.96      3910
           1       0.39      0.20      0.27       322

    accuracy                           0.92      4232
   macro avg       0.66      0.59      0.61      4232
weighted avg       0.90      0.92      0.90      4232

AUC-ROC score for the training dataset: 0.84
```

In [15]:
```
lgbm_model_Z_Scratch = lgb.LGBMClassifier(objective ='binary',
                                       boosting_type ='gbdt',
```

```
                                                   data_sample_strategy ="goss",
                                                   metric="auc",
                                                   colsample_bytree=0.20,
                                                   subsample=0.25,
                                                   learning_rate=0.10,
                                                   max_depth=12,
                                                   n_estimators=3000,
                                                   num_leaves=120,
                                                   reg_alpha=0.15,
                                                   reg_lambda=0.90,
                                                   verbose=-1,
                                                   random_state=42)

lgbm_model_Z_Scratch.fit(X_train_Z_Scratch_smote, y_train_Z_Scratch_smote)
y_pred_Z_Scratch = lgbm_model_Z_Scratch.predict(X_test_Z_Scratch_scaled)
accuracy_Z_Scratch = accuracy_score(y_test_Z_Scratch, y_pred_Z_Scratch)
print(f"Accuracy: {accuracy_Z_Scratch}")
print(classification_report(y_test_Z_Scratch, y_pred_Z_Scratch))

y_train_Z_Scratch_pred_proba = lgbm_model_Z_Scratch.predict_proba(X_test_Z_Scratch_scale
roc_auc_Z_Scratch = roc_auc_score(y_test_Z_Scratch, y_train_Z_Scratch_pred_proba)
print(f"AUC-ROC score for the training dataset: {roc_auc_Z_Scratch:.2f}")
```

```
Accuracy: 0.9525047258979206
              precision    recall  f1-score   support

           0       0.97      0.98      0.97      3974
           1       0.64      0.49      0.56       258

    accuracy                           0.95      4232
   macro avg       0.81      0.74      0.77      4232
weighted avg       0.95      0.95      0.95      4232

AUC-ROC score for the training dataset: 0.95
```

In [16]:

```
lgbm_model_K_Scatch = lgb.LGBMClassifier(objective ='binary',
                                   boosting_type ='gbdt',
                                   data_sample_strategy ="goss",
                                   metric="auc",
                                   colsample_bytree=0.20,
                                   subsample=0.25,
                                   learning_rate=0.10,
                                   max_depth=12,
                                   n_estimators=3000,
                                   num_leaves=120,
                                   reg_alpha=0.15,
                                   reg_lambda=0.90,
                                   verbose=-1,
                                   random_state=42)

lgbm_model_K_Scatch.fit(X_train_K_Scatch_smote, y_train_K_Scatch_smote)
y_pred_K_Scatch = lgbm_model_K_Scatch.predict(X_test_K_Scatch_scaled)
accuracy_K_Scatch = accuracy_score(y_test_K_Scatch, y_pred_K_Scatch)
print(f"Accuracy: {accuracy_K_Scatch}")
print(classification_report(y_test_K_Scatch, y_pred_K_Scatch))

y_train_K_Scatch_pred_proba = lgbm_model_K_Scatch.predict_proba(X_test_K_Scatch_scaled)[
roc_auc_K_Scatch = roc_auc_score(y_test_K_Scatch, y_train_K_Scatch_pred_proba)
print(f"AUC-ROC score for the training dataset: {roc_auc_K_Scatch:.2f}")
```

```
Accuracy: 0.9629017013232514
              precision    recall  f1-score   support

           0       0.98      0.97      0.98      3479
           1       0.88      0.91      0.90       753
```

```
    accuracy                           0.96      4232
   macro avg       0.93      0.94      0.94      4232
weighted avg       0.96      0.96      0.96      4232

AUC-ROC score for the training dataset: 0.98
```

In [17]:

```python
lgbm_model_Stains = lgb.LGBMClassifier(objective ='binary',
                                   boosting_type ='gbdt',
                                   data_sample_strategy ="goss",
                                   metric="auc",
                                   colsample_bytree=0.20,
                                   subsample=0.25,
                                   learning_rate=0.10,
                                   max_depth=12,
                                   n_estimators=3000,
                                   num_leaves=120,
                                   reg_alpha=0.15,
                                   reg_lambda=0.90,
                                   verbose=-1,
                                   random_state=42)

lgbm_model_Stains.fit(X_train_Stains_smote, y_train_Stains_smote)
y_pred_Stains = lgbm_model_Stains.predict(X_test_Stains_scaled)
accuracy_Stains = accuracy_score(y_test_Stains, y_pred_Stains)
print(f"Accuracy: {accuracy_Stains}")
print(classification_report(y_test_Stains, y_pred_Stains))

y_train_Stains_pred_proba = lgbm_model_Stains.predict_proba(X_test_Stains_scaled)[:, 1]
roc_auc_Stains = roc_auc_score(y_test_Stains, y_train_Stains_pred_proba)
print(f"AUC-ROC score for the training dataset: {roc_auc_Stains:.2f}")
```

```
Accuracy: 0.9848771266540642
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      4100
           1       0.74      0.79      0.76       132

    accuracy                           0.98      4232
   macro avg       0.87      0.89      0.88      4232
weighted avg       0.99      0.98      0.99      4232

AUC-ROC score for the training dataset: 0.99
```

In [18]:

```python
lgbm_model_Dirtiness = lgb.LGBMClassifier(objective ='binary',
                                   boosting_type ='gbdt',
                                   data_sample_strategy ="goss",
                                   metric="auc",
                                   colsample_bytree=0.2,
                                   subsample=0.2,
                                   learning_rate=0.14,
                                   max_depth=16,
                                   n_estimators=1000,
                                   num_leaves=290,
                                   reg_alpha=0.4,
                                   reg_lambda=0.9,
                                   verbose=-1,
                                   min_child_samples=24,
                                   random_state=42)

lgbm_model_Dirtiness.fit(X_train_Dirtiness_smote, y_train_Dirtiness_smote)
y_pred_Dirtiness = lgbm_model_Dirtiness.predict(X_test_Dirtiness_scaled)
accuracy_Dirtiness = accuracy_score(y_test_Dirtiness, y_pred_Dirtiness)
print(f"Accuracy: {accuracy_Dirtiness}")
print(classification_report(y_test_Dirtiness, y_pred_Dirtiness))

y_train_Dirtiness_pred_proba = lgbm_model_Dirtiness.predict_proba(X_test_Dirtiness_scale
```

```
roc_auc_Dirtiness = roc_auc_score(y_test_Dirtiness, y_train_Dirtiness_pred_proba)
print(f"AUC-ROC score for the training dataset: {roc_auc_Dirtiness:.2f}")
```

```
Accuracy: 0.9773156899810964
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      4135
           1       0.51      0.21      0.29        97

    accuracy                           0.98      4232
   macro avg       0.75      0.60      0.64      4232
weighted avg       0.97      0.98      0.97      4232

AUC-ROC score for the training dataset: 0.88
```

In [19]:

```
catboost_model_Bumps = cb.CatBoostClassifier(verbose=0,
                                    loss_function='Logloss',
                                    eval_metric='AUC',
                                    iterations=300,
                                    learning_rate=0.03,
                                    depth=12,
                                    l2_leaf_reg=2,
                                    border_count=220,
                                    bagging_temperature=0.2,
                                    random_strength=0.7,
                                    scale_pos_weight=0.9)

catboost_model_Bumps.fit(X_train_Bumps_smote, y_train_Bumps_smote)
y_pred_Bumps = catboost_model_Bumps.predict(X_test_Bumps_scaled)
accuracy_Bumps = accuracy_score(y_test_Bumps, y_pred_Bumps)
print(f"Accuracy: {accuracy_Bumps}")
print(classification_report(y_test_Bumps, y_pred_Bumps))

y_train_Bumps_pred_proba = catboost_model_Bumps.predict_proba(X_test_Bumps_scaled)[:, 1]
roc_auc_Bumps = roc_auc_score(y_test_Bumps, y_train_Bumps_pred_proba)
print(f"AUC-ROC score for the training dataset: {roc_auc_Bumps:.2f}")
```

```
Accuracy: 0.775047258979206
              precision    recall  f1-score   support

           0       0.85      0.86      0.85      3249
           1       0.52      0.51      0.51       983

    accuracy                           0.78      4232
   macro avg       0.68      0.68      0.68      4232
weighted avg       0.77      0.78      0.77      4232

AUC-ROC score for the training dataset: 0.81
```

In [20]:

```
catboost_model_Other_Faults = cb.CatBoostClassifier(verbose=0,
                                    loss_function='Logloss',
                                    eval_metric='AUC',
                                    iterations=200,
                                    learning_rate=0.07,
                                    depth=7,
                                    l2_leaf_reg=2,
                                    border_count=260,
                                    bagging_temperature=0.9,
                                    random_strength=0.1,
                                    scale_pos_weight=0.5)

catboost_model_Other_Faults.fit(X_train_Other_Faults_smote, y_train_Other_Faults_smote)
y_pred_Other_Faults = catboost_model_Other_Faults.predict(X_test_Other_Faults_scaled)
accuracy_Other_Faults = accuracy_score(y_test_Other_Faults, y_pred_Other_Faults)
print(f"Accuracy: {accuracy_Other_Faults}")
print(classification_report(y_test_Other_Faults, y_pred_Other_Faults))
```

```
y_train_Other_Faults_pred_proba = catboost_model_Other_Faults.predict_proba(X_test_Other
roc_auc_Other_Faults = roc_auc_score(y_test_Other_Faults, y_train_Other_Faults_pred_prob
print(f"AUC-ROC score for the training dataset: {roc_auc_Other_Faults:.2f}")
```

```
Accuracy: 0.6755671077504726
              precision    recall  f1-score   support

           0       0.67      0.96      0.79      2707
           1       0.71      0.17      0.28      1525

    accuracy                           0.68      4232
   macro avg       0.69      0.57      0.53      4232
weighted avg       0.68      0.68      0.61      4232

AUC-ROC score for the training dataset: 0.73
```

In [21]:
```
overall_roc_auc = (roc_auc_Other_Faults+roc_auc_Bumps+roc_auc_Dirtiness+roc_auc_Stains+r
print(f"Overall AUC-ROC score for the training dataset: {overall_roc_auc:.2f}")
```

```
Overall AUC-ROC score for the training dataset: 0.88
```

Submission and conclusion

In [22]:
```
X_test_scaled = scaler.transform(test)
```

In [23]:
```
y_val_Pastry = lgbm_model_Pastry.predict_proba(X_test_scaled)[:, 1]
y_val_Z_Scratch = lgbm_model_Z_Scratch.predict_proba(X_test_scaled)[:, 1]
y_val_K_Scatch = lgbm_model_K_Scatch.predict_proba(X_test_scaled)[:, 1]
y_val_Stains = lgbm_model_Stains.predict_proba(X_test_scaled)[:, 1]
y_val_Dirtiness = lgbm_model_Dirtiness.predict_proba(X_test_scaled)[:, 1]
y_val_Bumps = catboost_model_Bumps.predict_proba(X_test_scaled)[:, 1]
y_val_Other_Faults = catboost_model_Other_Faults.predict_proba(X_test_scaled)[:, 1]
```

In [24]:
```
submission = pd.DataFrame({
    "id": validation_ids,
    "Pastry": y_val_Pastry,
    "Z_Scratch": y_val_Z_Scratch,
    "K_Scatch": y_val_K_Scatch,
    "Stains": y_val_Stains,
    "Dirtiness": y_val_Dirtiness,
    "Bumps": y_val_Bumps,
    "Other_Faults":y_val_Other_Faults})
submission.to_csv("s4e03_2503_final.csv", index=False)
```

In [25]:
```
submission.head()
```

Out[25]:

	id	Pastry	Z_Scratch	K_Scatch	Stains	Dirtiness	Bumps	Other_Faults
0	19219	0.758933	2.672571e-07	3.916142e-05	3.934458e-07	0.000809	0.160991	0.220146
1	19220	0.049082	3.090268e-04	6.535804e-07	2.911253e-06	0.049543	0.103068	0.208759
2	19221	0.000065	1.597733e-03	2.031185e-04	1.120500e-06	0.000154	0.306067	0.402969
3	19222	0.064993	4.182642e-07	1.174328e-07	4.032868e-05	0.000784	0.432290	0.311818
4	19223	0.000074	7.828435e-06	4.186211e-07	7.501313e-06	0.000967	0.735086	0.281195

The model achieved an 88.273% score upon submission, aligning closely with the validation set performance. While slightly behind top Kaggle competitors, the consistent metric underscores the model's robustness. This result is considered satisfactory, given the challenge's complexity and competition caliber.