WeldOptim: Advanced Precision in Welding Enhancement

Project Description

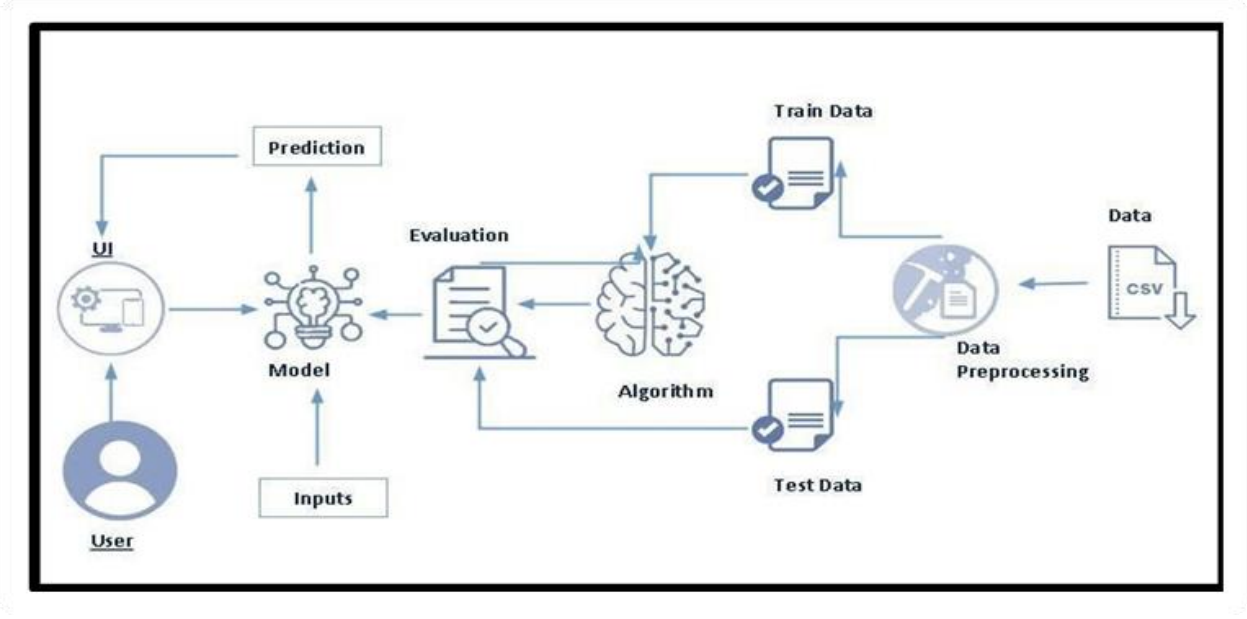
Achieving optimal welding quality requires precise control over parameters like pressure, welding time, and material properties. Variations in these factors can lead to defects, affecting structural integrity and performance. Traditional methods lack efficiency in predicting optimal welding conditions. This project aims to leverage machine learning to analyze a dataset comprising pressure, welding time, and other parameters to develop predictive models for achieving superior welding quality.

Scenario 1: Automated Welding Process Enhancement
In the automotive industry, where high-volume production demands consistent quality, the model assists in optimizing welding parameters for robotic welding systems. Analyzing historical data and real-time feedback, ensures precise control over the welding process, reducing defects and enhancing productivity.

Scenario 2: Material Compatibility Optimization
In the mechanical industry, where diverse materials are welded together, the model identifies optimal welding parameters considering material compatibility. It recommends adjustments in pressure, welding time, and other variables to achieve strong and reliable welds, ensuring the integrity of mechanical structures and components.

Scenario 3: Quality Assurance in Structural Welding
For structural welding applications, such as in shipbuilding or bridge construction, the model provides insights into achieving welds with superior strength and durability. Analyzing factors like pressure, welding time, and material properties, aids in the development of welding procedures that meet stringent quality standards and regulatory requirements.

Technical Architecture



Project Flow:

- User interacts with the UI to enter the input.
- Entered input is analyzed by the model which is integrated.
- Once the model analyzes the input the prediction is showcased on the UI.

To accomplish this, we have to complete all the activities listed below:

- **Data Collection & Preparation**
  - Collect the dataset
  - Data Preparation
- **Exploratory Data Analysis and Train Test Split**
  - Descriptive statistical
  - Visual Analysis
  - Splitting data into input and target variable (X and y split)
  - Data Scaling
  - Train Test Split
  - Balancing Training Data
- **Model Building**
  - Training and testing the model in multiple algorithms
- **Performance Testing**
  - Comparing models with multiple evaluation metrics

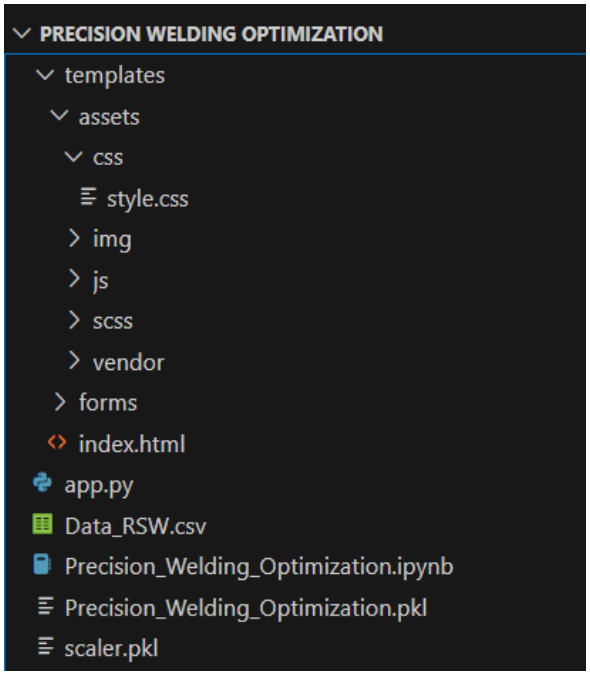
    - Selecting the best model
    - Testing the model
- **Model Deployment**
    - Save the best model
    - Integrate with Web Framework

Prior Knowledge:

You must have the prior knowledge of the following topics to complete this project.

- ML Concepts
- Supervised Learning: https://www.javatpoint.com/supervised-machine-learning
- Logistic Regression: https://www.geeksforgeeks.org/understanding-logistic-regression/
- Decision Tree: https://www.geeksforgeeks.org/decision-tree/
- Random Forest: https://www.geeksforgeeks.org/random-forest-algorithm-in-machine-learning/
- SVC: https://www.geeksforgeeks.org/support-vector-machine-algorithm/
- Evaluation Metrics: https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/
- Flask Basics: https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Structure



- We are building a Flask application requiring HTML pages stored in the Template folder. CSS and JS scripts reside in the Assets folder, while images are in the Img folder within Assets. Additionally, we have a Python script named app.py for scripting.

- Precision_Welding_Optimization.pkl is our saved model and scale.pkl contains our scaled data. These files will be utilized for Flask integration.

- Data_RSW.csv is the dataset and Precision_Welding_Optimization.ipynb is the model training file.

Milestone 1: Data Collection & Preparation

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So, this section allows you to download the required dataset.

Activity 1: Collect the Dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc. In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

The Resistance Spot Welding Insights dataset provides a comprehensive record of crucial welding spot process parameters. It includes monitoring of factors like current and force during nugget formation, along with input parameters such as welding time, applied force, and material characteristics. Additionally, the dataset offers valuable welding outputs like mechanical resistance and nugget diameter, along with their classifications. This dataset is a valuable resource for optimizing resistance spot welding processes and improving welding quality.

Link: https://www.kaggle.com/datasets/warcoder/resistance-spot-welding-insights

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualization techniques and some analyzing techniques.

Note: There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Activity 1.1: Importing the libraries

Import the necessary libraries as shown in the image.

```
# import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, roc_auc_score
import pickle
```

Activity 1.2: Read the Dataset

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas. In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of the csv file.

```
df = pd.read_csv('Data_RSW.csv')
df.head()
```

	Sample ID	Pressure (PSI)	Welding Time (ms)	Angle (Deg)	Force (N)	Current (A)	Thickness A (mm)	Thickness B (mm)	Material	PullTest (N)	NuggetDiameter (mm)	Category	Comments
0	1	35	200	0	0.00	1315.41	0.922	0.920	SS	2127.7	2.63	Bad	DOE
1	1	35	200	0	3.41	1337.45	0.922	0.920	SS	2127.7	2.63	Bad	DOE
2	1	35	200	0	6.82	1081.47	0.922	0.920	SS	2127.7	2.63	Bad	DOE
3	2	35	1500	0	0.00	1819.13	0.920	0.925	SS	5346.4	3.34	Good	DOE
4	2	35	1500	0	3.41	2016.44	0.920	0.925	SS	5346.4	3.34	Good	DOE

Activity 2: Data Preparation

As we have understood how the data is, let's pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results.

This activity includes the following steps.

- Removal of Unwanted Columns
- Handling Missing Values
- Handling Categorical Values
- Handling Outliers

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

Activity 2.1: Removal of Unwanted Columns

We remove unwanted columns that do not contribute to identifying patterns essential for building models. These irrelevant columns, such as serial numbers, customer IDs, or roll numbers, are extraneous to the modeling process and only add noise to the dataset.

For our specific case, we remove the following columns - Sample ID, Material and Comments as they do not provide valuable insights or contribute significantly to the predictive modeling of welding optimization. This streamlining ensures that our dataset contains only relevant features essential for accurate model training and prediction.

```
df.drop(['Sample ID', 'Comments', 'Material'], axis=1, inplace=True)
df.head()
```

	Pressure (PSI)	Welding Time (ms)	Angle (Deg)	Force (N)	Current (A)	Thickness A (mm)	Thickness B (mm)	PullTest (N)	NuggetDiameter (mm)	Category
0	35	200	0	0.00	1315.41	0.922	0.920	2127.7	2.63	Bad
1	35	200	0	3.41	1337.45	0.922	0.920	2127.7	2.63	Bad
2	35	200	0	6.82	1081.47	0.922	0.920	2127.7	2.63	Bad
3	35	1500	0	0.00	1819.13	0.920	0.925	5346.4	3.34	Good
4	35	1500	0	3.41	2016.44	0.920	0.925	5346.4	3.34	Good

Activity 2.1: Handling Missing Values

Let's find the shape of our dataset first. To find the shape of our data, the df.shape method is used. There are 4186 records and 10 columns after removing the unwanted columns.

```
df.shape
```

```
(4186, 10)
```

For checking the null values, df.isnull() function is used. To sum those null values, we use .sum() function. Another way is by using df.isnull().any(). From the below image we found that there no null values present in our dataset:

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

```
Pressure (PSI)          0
Welding Time (ms)       0
Angle (Deg)             0
Force (N)               0
Current (A)             0
Thickness A (mm)        0
Thickness B (mm)        0
PullTest (N)            0
NuggetDiameter (mm)     0
Category                0
dtype: int64
```

```
df.isnull().any()
```

```
Pressure (PSI)          False
Welding Time (ms)       False
Angle (Deg)             False
Force (N)               False
Current (A)             False
Thickness A (mm)        False
Thickness B (mm)        False
PullTest (N)            False
NuggetDiameter (mm)     False
Category                False
dtype: bool
```

Activity 2.1: Handling Categorical Values

To find the data type, df.info() function is used. All columns in the dataset are in numerical format, except for 'Category'.

```
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4186 entries, 0 to 4185
Data columns (total 10 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Pressure (PSI)       4186 non-null   int64
 1   Welding Time (ms)    4186 non-null   int64
 2   Angle (Deg)          4186 non-null   int64
 3   Force (N)            4186 non-null   float64
 4   Current (A)          4186 non-null   float64
 5   Thickness A (mm)     4186 non-null   float64
 6   Thickness B (mm)     4186 non-null   float64
 7   PullTest (N)         4186 non-null   float64
 8   NuggetDiameter (mm)  4186 non-null   float64
 9   Category             4186 non-null   object
dtypes: float64(6), int64(3), object(1)
memory usage: 327.2+ KB
```

Since 'Category' is our target variable, there's no need to encode it. Encoding is only necessary for input variables if they are categorical.

To encode categorical data, there are three methods available:

- LabelEncoder: Converts categorical labels into numerical values.
- One-Hot Encoding: Represents categorical variables as binary vectors.
- Manual Encoding: Custom encoding method applied manually to categorical data.

Activity 2.1: Handling Outliers

An outlier is an observation that significantly deviates from other observations in a dataset, potentially indicating a data entry error, measurement variability, or a genuine anomaly in the data.

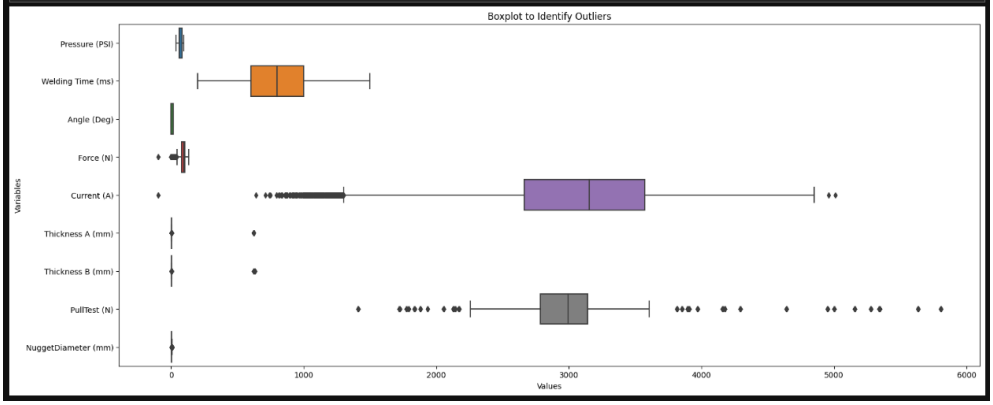
To deal with Outliers, there are three methods available:

- IQR (Interquartile Range): A measure of statistical dispersion that represents the range between the first quartile (25th percentile) and the third quartile (75th percentile) of a dataset, useful for detecting outliers.
- Z Score: A statistical measure that quantifies how many standard deviations a data point is from the mean of the dataset, providing a way to identify outliers based on their deviation from the mean.

- Percentile Method: A method of outlier detection that involves identifying data points that fall above or below a certain percentile threshold in the dataset, such as the 95th or 99th percentile, indicating extreme values relative to the rest of the data.

Let's create a horizontal boxplot using Seaborn to visualize the distribution of data across different variables in the DataFrame. The boxplot helps identify outliers and provides a summary of the data's central tendency and dispersion.

```
plt.figure(figsize=(20, 8))
sns.boxplot(data=df, orient='h')
plt.title('Boxplot to Identify Outliers')
plt.xlabel('Values')
plt.ylabel('Variables')
plt.show()
```



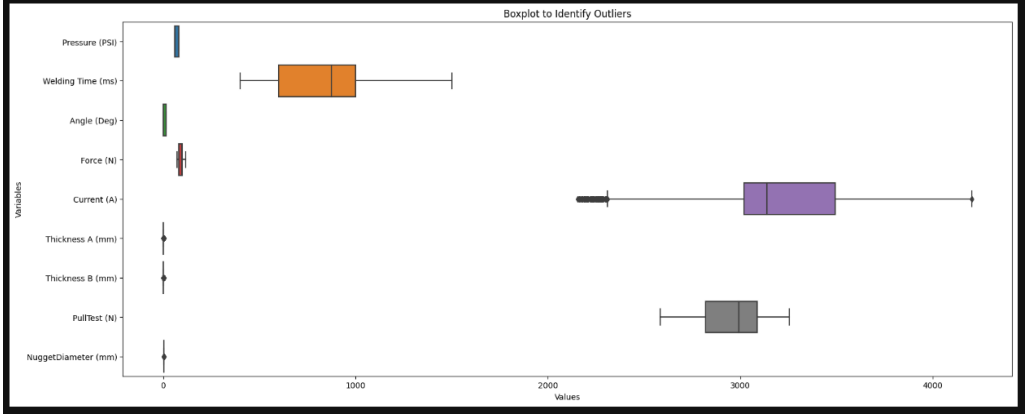
It's evident that numerous outliers are present in the dataset.

To handle outliers, we employ the percentile method. Here, we define a function to replace outliers in numerical columns of a DataFrame with their respective column means, using percentiles to establish outlier boundaries. Outliers falling below the 5th percentile or above the 95th percentile are identified and replaced with the mean of their respective columns.

```
# Define a function to replace outliers with column means using percentiles
def replace_outliers_with_mean_percentile(df, lower_percentile=5, upper_percentile=95):
    for column in df.select_dtypes(include='number').columns:
        # Calculate lower and upper percentiles
        lower_bound = np.percentile(df[column], lower_percentile)
        upper_bound = np.percentile(df[column], upper_percentile)

        # Identify outliers
        outliers = (df[column] < lower_bound) | (df[column] > upper_bound)

        # Replace outliers with column mean
        df.loc[outliers, column] = df[column].mean()

# Call the function to replace outliers with column means using percentiles
replace_outliers_with_mean_percentile(df)
```

Let’s check the boxplots again.

```
plt.figure(figsize=(20, 8))
sns.boxplot(data=df, orient='h')
plt.title('Boxplot to Identify Outliers')
plt.xlabel('Values')
plt.ylabel('Variables')
plt.show()
```



While we successfully addressed many outliers, a few still remain unresolved.

Milestone 2: Exploratory Data Analysis

Activity 1: Descriptive Analysis

Descriptive analysis involves examining fundamental characteristics of data using statistical methods. Pandas offers a valuable function known as 'describe' for this purpose. Utilizing the 'describe' function enables us to uncover unique, top, and frequently occurring values within categorical features. Moreover, it provides insights into the mean, standard deviation, minimum, maximum, and percentile values of continuous features.

```
df.describe()
```

	Pressure (PSI)	Welding Time (ms)	Angle (Deg)	Force (N)	Current (A)	Thickness A (mm)	Thickness B (mm)	PullTest (N)	NuggetDiameter (mm)
count	4186.000000	4186.000000	4186.000000	4186.000000	4186.000000	4186.000000	4186.000000	4186.00000	4186.000000
mean	65.590062	875.298614	7.514333	91.217286	3019.032711	2.289362	3.501072	3032.96204	3.577721
std	11.780475	303.025712	7.500882	19.300206	844.557376	31.916085	42.217941	521.44465	0.393631
min	35.000000	200.000000	0.000000	-99.000000	-99.000000	0.610000	0.608000	1410.30000	1.900000
25%	60.000000	600.000000	0.000000	78.890000	2663.300000	0.623000	0.623000	2783.60000	3.350000
50%	60.000000	800.000000	15.000000	95.010000	3155.460000	0.631000	0.630000	2994.10000	3.570000
75%	80.000000	1000.000000	15.000000	103.677500	3571.340000	0.638000	0.635000	3141.80000	3.757500
max	95.000000	1500.000000	15.000000	133.530000	5009.430000	624.000000	632.000000	5806.50000	4.720000

Using the value_counts() function, we'll determine the frequency of each unique class in the Category (Target variable) column.

```
df['Category'].value_counts()
```

```
Good       3814
Explode     284
Bad          88
Name: Category, dtype: int64
```

Activity 2: Visual analysis

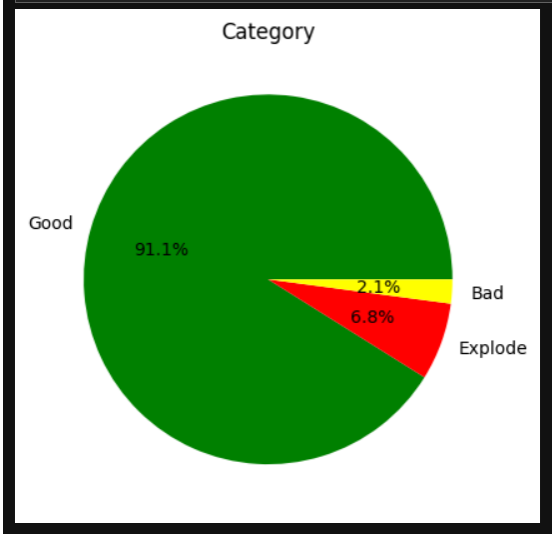
Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

Activity 2.1: Univariate analysis

In simple words, univariate analysis is understanding the data with a single feature. Here we have displayed two different graphs such as pie chart and displot.

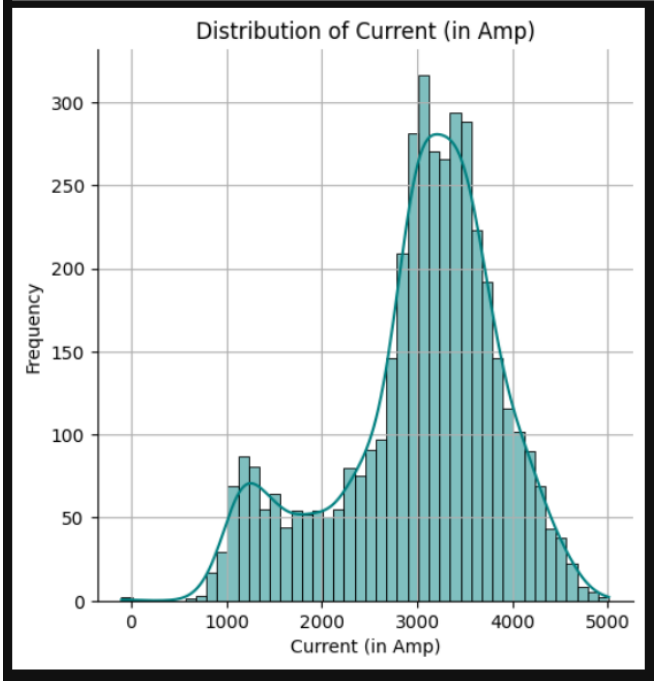
We have used matplotlib library to create a pie chart visualization. We created a pie chart displaying the distribution of categories in the "Category" column of the DataFrame. Each category (Good, Explode, Bad) is represented by a slice of the pie chart, with the percentage of occurrences labeled on each slice.

```
plt.pie(df['Category'].value_counts(),
        labels = ['Good','Explode', 'Bad'],
        autopct = '%1.1f%%',
        colors = ['green','red','yellow'])
plt.title('Category')
plt.show()
```



The following code snippet uses Seaborn to create a distribution plot (histogram) of the "Current (A)" column from the DataFrame. The plot visualizes the frequency distribution of current values, with a kernel density estimation (KDE) overlayed. The plot title, axis labels, and grid are added for better clarity.

```
sns.displot(df['Current (A)'], kde=True, color='teal')
plt.title('Distribution of Current (in Amp)')
plt.xlabel('Current (in Amp)')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```

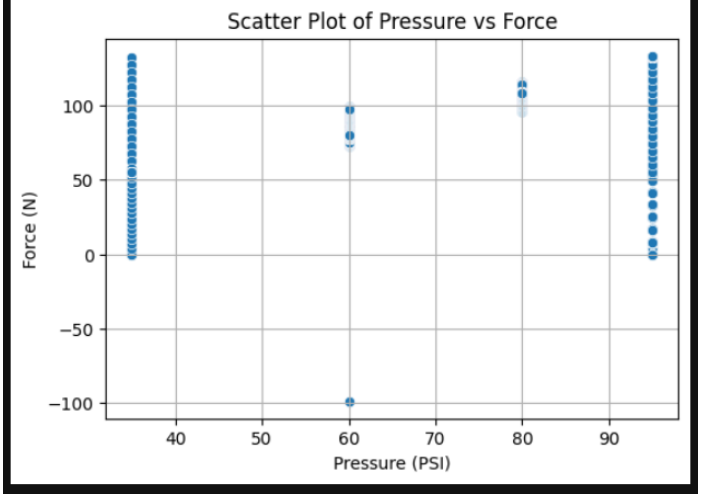


Activity 2.2: Bivariate analysis

Bivariate analysis is employed to explore the relationship between two features. Here we have displayed two different graphs such as scatter plot and line plot.

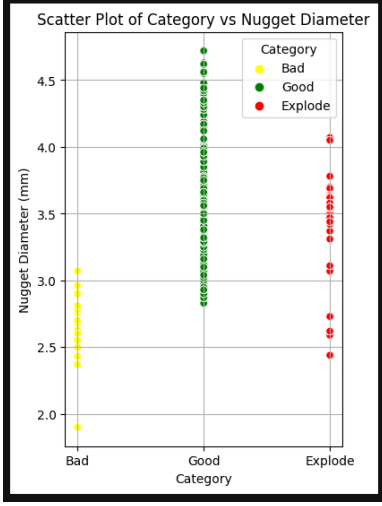
We are generating a scatter plot using the Seaborn library, illustrating the relationship between pressure (in PSI) and force (in N). By plotting pressure on the x-axis and force on the y-axis, we can observe patterns or trends in how changes in pressure affect the force exerted during the welding process. This graphical representation aids in identifying any potential correlations or relationships between pressure and force, which are essential for understanding the welding process dynamics.

```
plt.figure(figsize=(6, 4))
sns.scatterplot(data=df, x='Pressure (PSI)', y='Force (N)')
plt.title('Scatter Plot of Pressure vs Force')
plt.xlabel('Pressure (PSI)')
plt.ylabel('Force (N)')
plt.grid(True)
plt.show()
```



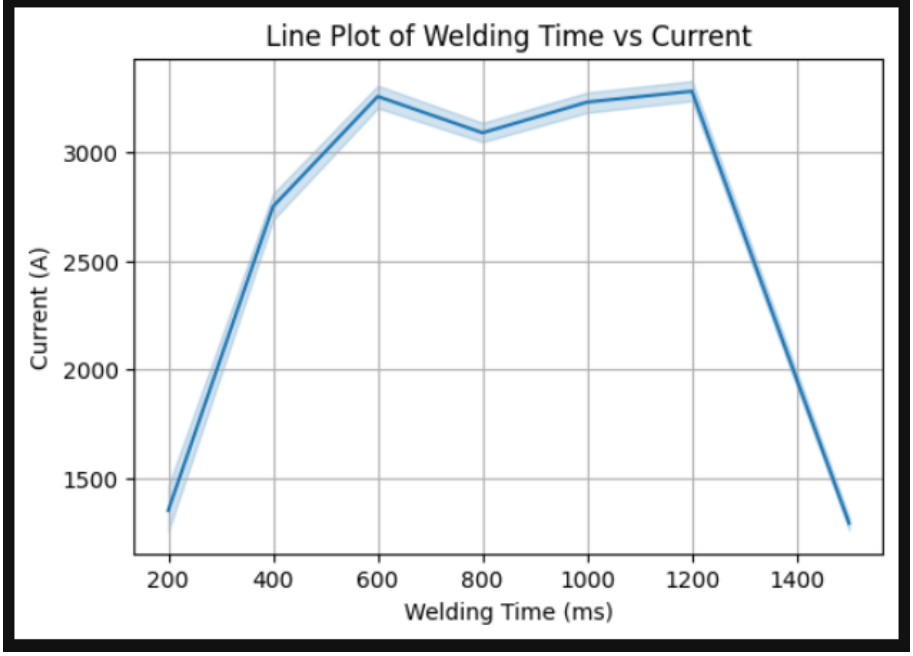
We are generating a scatter plot using the Seaborn library, illustrating the relationship between welding category and nugget diameter (in mm). Each data point represents a specific welding instance categorized as 'Good', 'Bad', or 'Explode', color-coded accordingly. This visualization helps to understand how different welding categories correlate with the resulting nugget diameter, providing insights into the welding quality based on the diameter of the weld nugget.

```
plt.figure(figsize=(4, 6))
colors = {'Good': 'green', 'Bad': 'yellow', 'Explode':'red'}
sns.scatterplot(data=df, x='Category', y='NuggetDiameter (mm)',hue='Category', palette=colors)
plt.title('Scatter Plot of Category vs Nugget Diameter')
plt.xlabel('Category')
plt.ylabel('Nugget Diameter (mm)')
plt.legend(title='Category', loc='upper right')
plt.grid(True)
plt.show()
```



We are generating a line plot using the Seaborn library, depicting the relationship between welding time (in milliseconds) and current (in amperes). This visualization helps to understand how welding time influences the current, providing insights into the welding process dynamics.

```
plt.figure(figsize=(6, 4))
sns.lineplot(data=df, x='Welding Time (ms)', y='Current (A)')
plt.title('Line Plot of Welding Time vs Current')
plt.xlabel('Welding Time (ms)')
plt.ylabel('Current (A)')
plt.grid(True)
plt.show()
```



Activity 2.3: Multivariate analysis

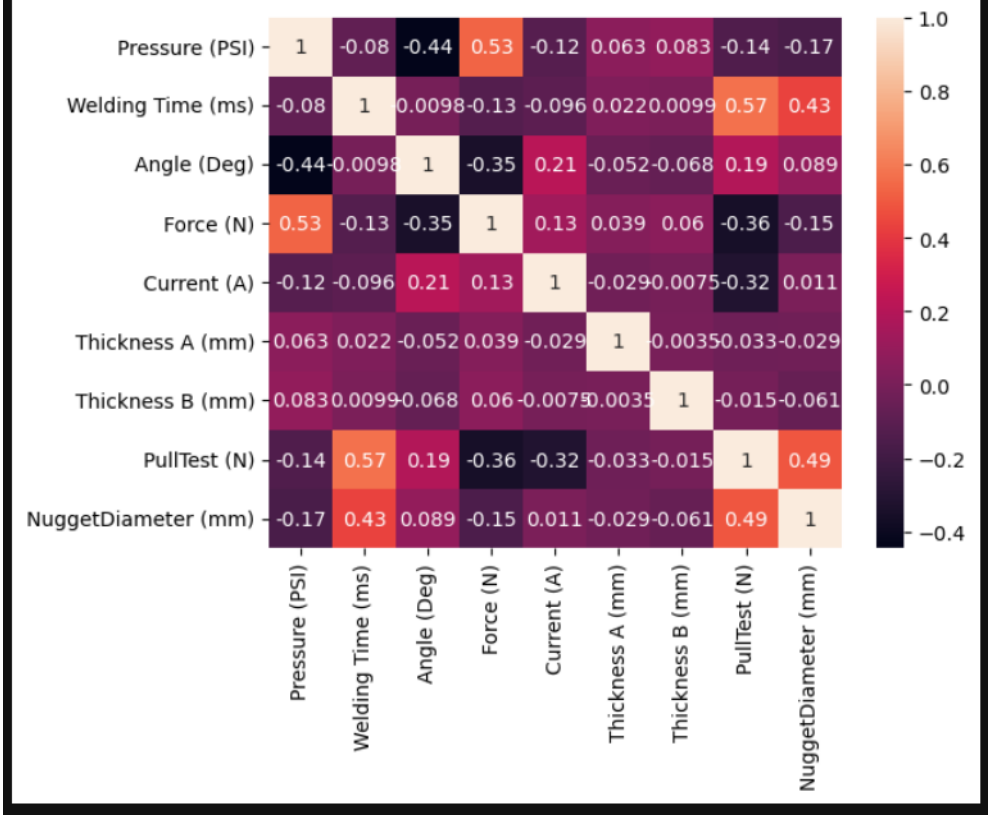
Multivariate analysis is employed to explore the relationship between more than two features. Here we have displayed two different graphs such as pairplot and correlation matrix.

We are utilizing the Seaborn library to generate a pair plot, which displays pairwise relationships between different variables in the DataFrame. Each scatterplot in the grid represents the relationship between two variables, while the diagonal plots show the distribution of individual variables. This visualization helps to identify patterns and correlations between variables in the dataset.

We are using the Seaborn library to create a heatmap of the correlation matrix of the DataFrame. Each cell in the heatmap represents the correlation coefficient between two variables, with color intensity indicating the strength and direction of the correlation. The 'annot=True' parameter adds numerical annotations to the heatmap, showing the correlation values. This visualization helps to identify patterns and relationships between different variables in the dataset.

```
sns.heatmap(df.corr(), annot = True)
```

```
<AxesSubplot:>
```



Activity 3: Splitting data into input and target variable

The code is splitting the dataset into input variables (X) and the target variable (y). Input variables (X) contain all columns except the 'Category' column, which is set as the target variable (y). This separation facilitates the training of machine learning models, where X represents the features used for prediction and y represents the target variable to be predicted.

```
X = df.drop(columns = ['Category'], axis = 1)
y = df['Category']
```

Activity 4: Data Scaling

The code is performing feature scaling on the input data using Min-Max scaling, which transforms the data to a specific range (usually between 0 and 1). This scaling is crucial because it ensures that all input features contribute equally to the model training process, preventing any particular feature from dominating due to its larger magnitude.

```
scale = MinMaxScaler()
X_scaled = pd.DataFrame(scale.fit_transform(X), columns = X.columns)
X_scaled.head()
```

	Pressure (PSI)	Welding Time (ms)	Angle (Deg)	Force (N)	Current (A)	Thickness A (mm)	Thickness B (mm)	PullTest (N)	NuggetDiameter (mm)
0	0.279503	0.43209	0.0	0.42483	0.435911	0.182865	0.105680	0.671374	0.446854
1	0.279503	0.43209	0.0	0.42483	0.435911	0.182865	0.105680	0.671374	0.446854
2	0.279503	0.43209	0.0	0.42483	0.422008	0.182865	0.105680	0.671374	0.446854
3	0.279503	1.00000	0.0	0.42483	0.490775	0.181670	0.107412	0.671374	0.272059
4	0.279503	1.00000	0.0	0.42483	0.490775	0.181670	0.107412	0.671374	0.272059

Now we will save the MinMaxScaler object named 'scale' as a pickle file named 'scaler.pkl'. We will use this pickle file for web integration.

```
with open('scaler.pkl', 'wb') as file:
    pickle.dump(scale, file)
```

Activity 5: Train Test Split

We are performing a train-test split on the scaled input features (X_scaled) and target variable (y). It splits the data into training and testing sets, with 75% of the data used for training and 25% for testing. The random_state parameter is set to 0 for reproducibility.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size = 0.25, random_state = 0)

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(3139, 9)
(1047, 9)
(3139,)
(1047,)
```

Activity 6: Balancing Training Data

An imbalanced dataset may bias the model towards the majority class, leading to poor predictions for minority classes. Balancing ensures each class is adequately represented during training, improving the model's ability to generalize and predict accurately across all classes. So let's check whether our data is balanced or not.

```
y_train.value_counts()
```

```
Good       2870
Explode     211
Bad          58
Name: Category, dtype: int64
```

It is evident that our data is highly imbalanced. So, to deal with it, we use the Synthetic Minority Over-sampling Technique (SMOTE) method to balance the training data by generating synthetic samples for the minority class. This helps address class imbalance issues and ensures equal representation of all classes in the training dataset.

```
smote = SMOTE()
X_train, y_train = smote.fit_resample(X_train, y_train)
y_train.value_counts()
```

```
Good       2870
Explode    2870
Bad        2870
Name: Category, dtype: int64
```

Milestone 3: Model Building

Activity 1: Training and testing the model using multiple algorithms

Now our data is cleaned and it’s time to build the model. We can train our data on different algorithms. For this project we are applying four classification algorithms. The best model is saved based on its performance.

Activity 1.1 Logistic Regression

We import the LogisticRegression class from the sklearn.linear_model module. Then, we initialize a logistic regression model named model1. Next, we train the model using the fit method with the training data X_train and corresponding labels y_train. After training, the model makes predictions on the test data X_test using the predict method, and the predicted labels are stored in y_pred1.

```
from sklearn.linear_model import LogisticRegression
model1 = LogisticRegression()
model1.fit(X_train, y_train)
```

```
LogisticRegression()
```

```
y_pred1 = model1.predict(X_test)
y_pred1
```

```
array(['Good', 'Good', 'Bad', ..., 'Explode', 'Good', 'Good'],
      dtype=object)
```

The following code evaluates the logistic regression model's performance by computing testing and training accuracies using accuracy_score, ROC AUC score using roc_auc_score, and generating a confusion matrix and classification report to assess class-wise prediction metrics.

```
test_acc1 = accuracy_score(y_test, y_pred1)
train_acc1 = accuracy_score(y_train, y_pred_train1)
print('Testing Accuracy = ', test_acc1)
print('Training Accuracy = ', train_acc1)
```

```
Testing Accuracy =  0.6361031518624641
Training Accuracy =  0.7723577235772358
```

```
probability = model1.predict_proba(X_test)
roc_auc1 = roc_auc_score(y_test, probability, multi_class='ovr')
print("ROC AUC score:", roc_auc1)
```

```
ROC AUC score: 0.8263597353469674
```

```
pd.crosstab(y_test,y_pred1)
```

col_0	Bad	Explode	Good
Category			
Bad	26	3	1
Explode	8	51	14
Good	131	224	589

```
# classification report
print(classification_report(y_test,y_pred1))
```

```
              precision    recall  f1-score   support

         Bad       0.16      0.87      0.27        30
     Explode       0.18      0.70      0.29        73
        Good       0.98      0.62      0.76       944

    accuracy                           0.64      1047
   macro avg       0.44      0.73      0.44      1047
weighted avg       0.90      0.64      0.71      1047
```

Activity 1.2 Decision Tree

We use the DecisionTreeClassifier from the scikit-learn library to build a decision tree model. It sets the maximum depth of the tree to 4, uses the 'best' strategy for choosing the split at each node, and employs the entropy criterion to measure the quality of splits. This classifier is then trained on the balanced training data (X_train, y_train), and used to predict the classes of the testing data (X_test).

```
from sklearn.tree import DecisionTreeClassifier
model2 = DecisionTreeClassifier(max_depth = 4, splitter = 'best', criterion = 'entropy')
model2.fit(X_train, y_train)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

```
y_pred2 = model2.predict(X_test)
y_pred2
```

```
array(['Good', 'Good', 'Good', ..., 'Explode', 'Good', 'Explode'],
      dtype=object)
```

The following code evaluates the decision tree model's performance by computing testing and training accuracies using accuracy_score, ROC AUC score using roc_auc_score, and generating a confusion matrix and classification report to assess class-wise prediction metrics.

```
test_acc2 = accuracy_score(y_test, y_pred2)
train_acc2 = accuracy_score(y_train, y_pred_train2)
print('Testing Accuracy = ', test_acc2)
print('Training Accuracy = ', train_acc2)
```

```
Testing Accuracy =  0.725883476599809
Training Accuracy =  0.8777003484320558
```

```
probability = model2.predict_proba(X_test)
roc_auc2 = roc_auc_score(y_test, probability, multi_class='ovr')
print("ROC AUC score:", roc_auc2)
```

```
ROC AUC score: 0.9237757410748658
```

```
pd.crosstab(y_test,y_pred2)
```

col_0	Bad	Explode	Good
Category			
Bad	30	0	0
Explode	0	68	5
Good	53	229	662

```
# classification report
print(classification_report(y_test,y_pred2))
```

```
              precision    recall  f1-score   support

         Bad       0.36      1.00      0.53        30
     Explode       0.23      0.93      0.37        73
        Good       0.99      0.70      0.82       944

    accuracy                           0.73      1047
   macro avg       0.53      0.88      0.57      1047
weighted avg       0.92      0.73      0.78      1047
```

Activity 1.3 Random Forest

We use the RandomForestClassifier from scikit-learn's ensemble module to construct a random forest classifier. This ensemble method creates multiple decision trees during training and combines their predictions to improve accuracy and control overfitting. It trains the model on the balanced training data (X_train, y_train) and uses it to predict the classes of the testing data (X_test) using the predict method.

```
from sklearn.ensemble import RandomForestClassifier
model3 = RandomForestClassifier()
model3.fit(X_train, y_train)
```

```
RandomForestClassifier()
```

```
y_pred3 = model3.predict(X_test)
y_pred3
```

```
array(['Good', 'Good', 'Good', ..., 'Explode', 'Good', 'Good'],
      dtype=object)
```

The following code evaluates the random forest model's performance by computing testing and training accuracies using accuracy_score, ROC AUC score using roc_auc_score, and generating a confusion matrix and classification report to assess class-wise prediction metrics.

```
test_acc3 = accuracy_score(y_test, y_pred3)
train_acc3 = accuracy_score(y_train, y_pred_train3)
print('Testing Accuracy = ', test_acc3)
print('Training Accuracy = ', train_acc3)
```

```
Testing Accuracy =  0.9990448901623686
Training Accuracy =  1.0
```

```
probability = model3.predict_proba(X_test)
roc_auc3 = roc_auc_score(y_test, probability, multi_class='ovr')
print("ROC AUC score:", roc_auc3)
```

```
ROC AUC score: 1.0
```

```
pd.crosstab(y_test,y_pred3)
```

col_0	Bad	Explode	Good
Category			
Bad	29	1	0
Explode	0	73	0
Good	0	0	944

```
# classification report
print(classification_report(y_test,y_pred3))
```

```
              precision    recall  f1-score   support

         Bad       1.00      0.97      0.98        30
     Explode       0.99      1.00      0.99        73
        Good       1.00      1.00      1.00       944

    accuracy                           1.00      1047
   macro avg       1.00      0.99      0.99      1047
weighted avg       1.00      1.00      1.00      1047
```

Activity 1.4 SVM Classifier

We import the Support Vector Classifier (SVC) from scikit-learn's SVM module. SVC is a supervised learning algorithm that analyzes data for classification tasks. Here, the model is instantiated with the parameter `probability=True`, which enables probability estimates. The `fit` method is then called to train the model on the balanced training data (X_train, y_train), and the `predict` method is used to predict the classes of the testing data (X_test).

```
from sklearn.svm import SVC
model4 = SVC(probability=True)
model4.fit(X_train, y_train)
```

```
SVC(probability=True)
```

```
y_pred4 = model4.predict(X_test)
y_pred4
```

```
array(['Good', 'Good', 'Good', ..., 'Explode', 'Good', 'Good'],
      dtype=object)
```

The following code evaluates the SVM classifier model's performance by computing testing and training accuracies using accuracy_score, ROC AUC score using roc_auc_score, and generating a confusion matrix and classification report to assess class-wise prediction metrics.

```
test_acc4 = accuracy_score(y_test, y_pred4)
train_acc4 = accuracy_score(y_train, y_pred_train4)
print('Testing Accuracy = ', test_acc4)
print('Training Accuracy = ', train_acc4)
```

```
Testing Accuracy =  0.8194842406876791
Training Accuracy =  0.9171893147502903
```

```
probability = model4.predict_proba(X_test)
roc_auc4 = roc_auc_score(y_test, probability, multi_class='ovr')
print("ROC AUC score:", roc_auc4)
```

```
ROC AUC score: 0.9497981110866732
```

```
pd.crosstab(y_test,y_pred4)
```

col_0	Bad	Explode	Good
Category			
Bad	29	1	0
Explode	0	67	6
Good	9	173	762

```
# classification report
print(classification_report(y_test,y_pred4))
```

```
              precision    recall  f1-score   support

         Bad       0.76      0.97      0.85        30
     Explode       0.28      0.92      0.43        73
        Good       0.99      0.81      0.89       944

    accuracy                           0.82      1047
   macro avg       0.68      0.90      0.72      1047
weighted avg       0.94      0.82      0.86      1047
```

Milestone 4: Performance Testing

Activity 1: Comparing models with multiple evaluation metrics

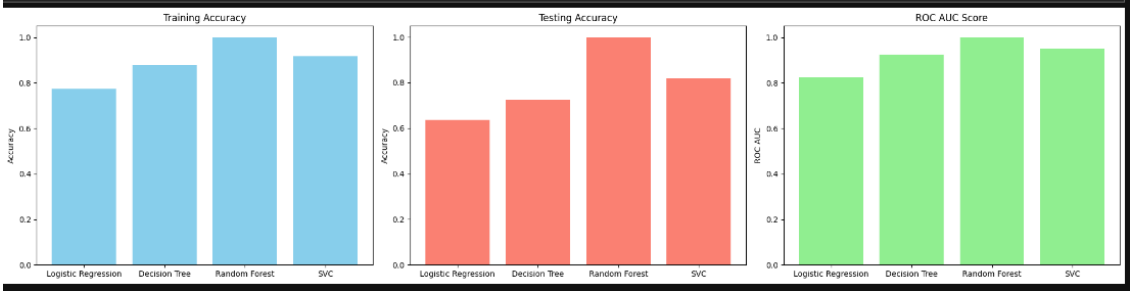
To compare all the models, we create a DataFrame containing the performance metrics such as training accuracy, testing accuracy, and ROC AUC score for each classification model.

```
models = ['Logistic Regression', 'Decision Tree', 'Random Forest', 'SVC']
train_acc = [train_acc1, train_acc2, train_acc3, train_acc4]
test_acc = [test_acc1, test_acc2, test_acc3, test_acc4]
roc_auc = [roc_auc1, roc_auc2, roc_auc3, roc_auc4]

df_results = pd.DataFrame({
    'Model Name': models,
    'Training Accuracy': train_acc,
    'Testing Accuracy': test_acc,
    'ROC AUC Score': roc_auc
})

df_results
```

	Model Name	Training Accuracy	Testing Accuracy	ROC AUC Score
0	Logistic Regression	0.772358	0.636103	0.826360
1	Decision Tree	0.877700	0.725883	0.923776
2	Random Forest	1.000000	0.999045	1.000000
3	SVC	0.917189	0.819484	0.949798

We also visualize the performance metrics of each classification model for easy comparison.

```
fig, axs = plt.subplots(1, 3, figsize=(20, 5))

# Plot Training Accuracy
axs[0].bar(models, train_acc, color='skyblue')
axs[0].set_title('Training Accuracy')
axs[0].set_ylabel('Accuracy')

# Plot Testing Accuracy
axs[1].bar(models, test_acc, color='salmon')
axs[1].set_title('Testing Accuracy')
axs[1].set_ylabel('Accuracy')

# Plot ROC AUC Score
axs[2].bar(models, roc_auc, color='lightgreen')
axs[2].set_title('ROC AUC Score')
axs[2].set_ylabel('ROC AUC')

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



Activity 2: Selecting the best model

The Random Forest model is the best because it achieved perfect accuracy on the training data and near-perfect accuracy on the testing data, along with a perfect ROC AUC score. This indicates that the model generalizes well to unseen data and has effectively captured the patterns in the dataset without overfitting.

Activity 3: Testing the model

Let's test the model first in the python notebook itself. As we have 9 features in this model, let's check the output by giving all the inputs.

```
sample1 = model3.predict(scale.transform([[35,200,0,0.00,1315.41,0.922,0.920,2127.7,2.63]]))
sample1
```

```
array(['Good'], dtype=object)
```

```
sample2 = model3.predict(scale.transform([[60,600,15,93.37,3637.78,0.622,3.501072,3097.6,3.43]]))
sample2
```

```
array(['Explode'], dtype=object)
```

Milestone 5: Model Deployment

Activity 1: Save the best model

Saving the best model after comparing its performance using different evaluation metrics means selecting the model with the highest performance. This can be useful in avoiding the need to retrain the model every time it is needed and also to be able to use it in the future.

```
import pickle
pickle.dump(model3,open('Precision_Welding_Optimization.pkl','wb'))
print('Pickle model downloaded successfully!!')
```

```
Pickle model downloaded successfully!!
```

Activity 2: Integrate with Web Framework

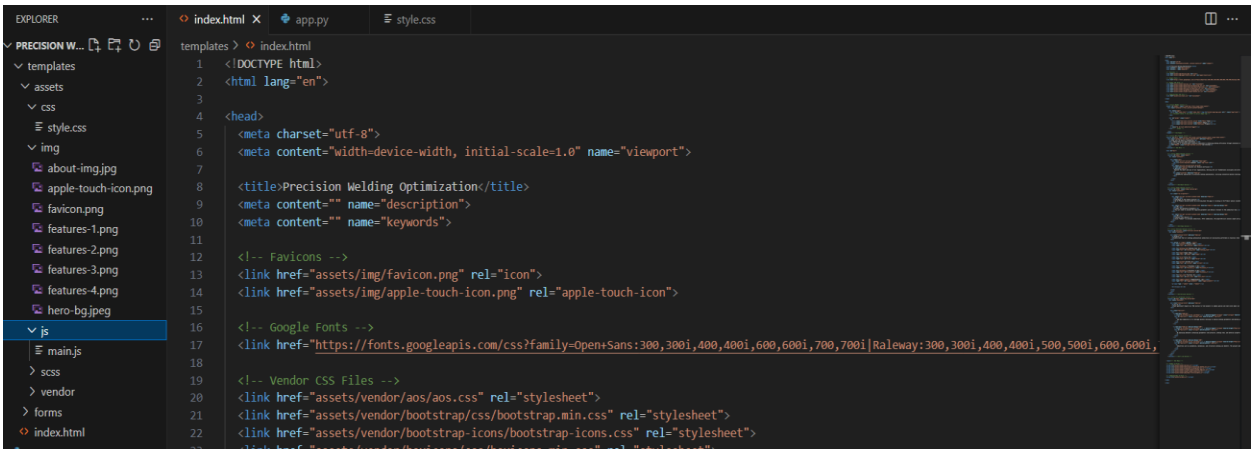
In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the user where he/she has to enter the values for predictions. The entered values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- Building HTML Pages
- Building server-side script
- Run the Web Application

Activity 2.1: Building Html Pages

For this project, an index.html file has been created and stored in the templates folder. Additionally, an assets folder has been created within the templates folder, containing CSS, JS, and image files.



Activity 2.2: Build Python code

Import the required libraries.

```
# import necessary libraries
from flask import render_template, request
from flask import Flask
import pickle
import numpy as np
from sklearn.preprocessing import MinMaxScaler
```

Load the saved model. Importing the flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (__name__) as argument.

```
app = Flask(__name__,static_url_path='/assets', static_folder='templates/assets')

model = pickle.load(open('Precision_Welding_Optimization.pkl','rb'))
scaler = pickle.load(open('scaler.pkl', 'rb'))
```

Render HTML page.

```python
@app.route('/')
def start():
    return render_template('index.html')
```

Here we will be using a declared constructor to route to the HTML page which we have created earlier.

In the above example, '/' URL is bound with the home.html function. Hence, when the home page of the web server is opened in the browser, the html page will be rendered.

Whenever you enter the values from the html page the values can be retrieved using POST and GET Methods.

Retrieving the value from UI:

Here we are routing our app to conditional statements. This will retrieve all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model.predict() function. This function returns the prediction. And this prediction value will be rendered to the text that we have mentioned in the submit.html page earlier.

```python
@app.route('/login', methods = ['POST'])

def login():
    a = request.form["pressure"]
    b = request.form["welding_time"]
    c = request.form["angle"]
    d = request.form["force"]
    e = request.form["current"]
    f = request.form["thickness_a"]
    g = request.form["thickness_b"]
    h = request.form["pull_test"]
    i = request.form["nugget_diameter"]

    t = [[float(a), float(b), float(c),float(d),float(e), float(f), float(g),float(h),float(i)]]
    scaled_t = scaler.transform(t)
    output = model.predict(scaled_t)
    print(output)

    return render_template("index.html", y = 'Expected Welding Category: '+(output[0]))
```

The following code calls the main function.

```python
if __name__ == '__main__':
    app.run(debug = True)
```

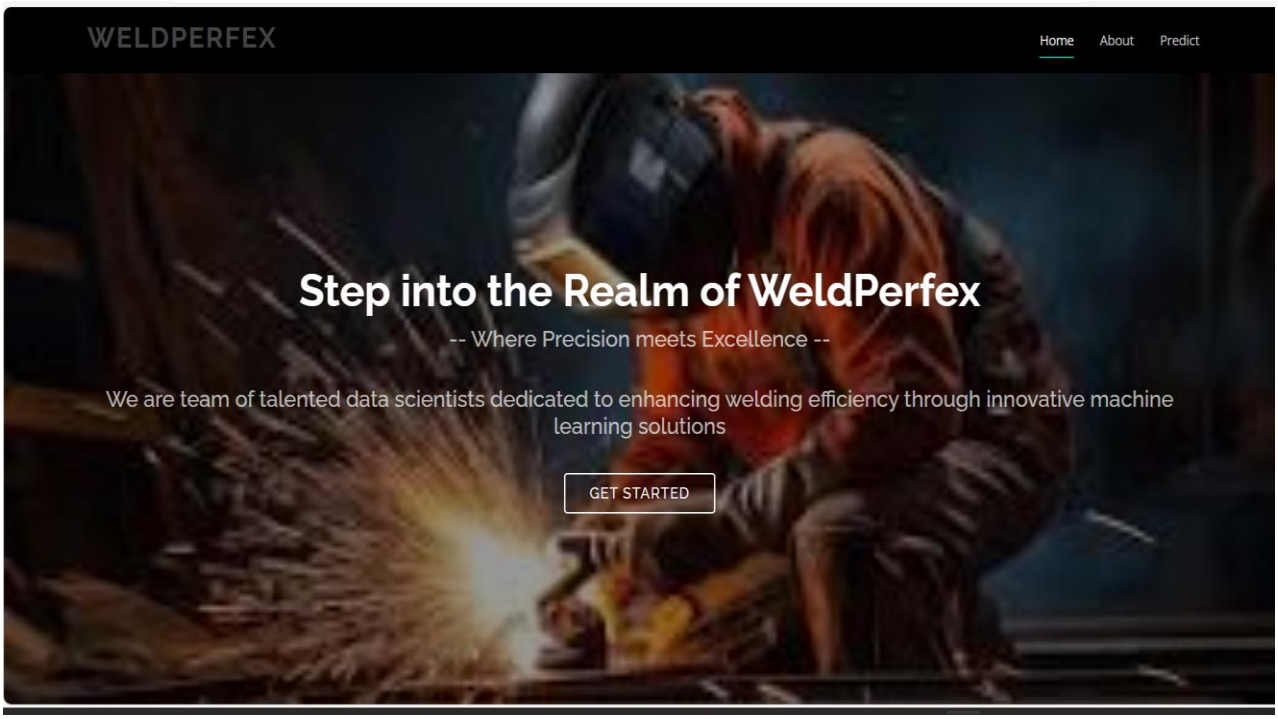
Activity 3.3: Run the web application

Launching the Flask Web Application:

- Open the anaconda prompt from the start menu.
- Navigate to the folder where your python script is.
- Now type “app.py” command .
- Navigate to the localhost where you can view your web page.

```
PS C:\Users\dhurvi patel\Desktop\SCIT academic\Smart_Bridge_Internship\SB_Company work\Major_Project_1\Precision Welding Optimization> & "C:/Users/dhurvi patel/AppData/Local/Programs/Python/Python310/python.exe" "c:/Users/dhurvi patel/Desktop/SCIT academic/Smart_Bridge_Internship/SB_Company work/Major_Project_1/Precision Welding Optimization/app.py"
C:\Users\dhurvi patel\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\base.py:329: UserWarning: Trying to unpickle estimator DecisionTreeClassifier from version 1.0.2
 when using version 1.1.1. This might lead to breaking code or invalid results. Use at your own risk. For more info please refer to:
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
  warnings.warn(
C:\Users\dhurvi patel\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\base.py:329: UserWarning: Trying to unpickle estimator RandomForestClassifier from version 1.0.2
 when using version 1.1.1. This might lead to breaking code or invalid results. Use at your own risk. For more info please refer to:
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
  warnings.warn(
C:\Users\dhurvi patel\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\base.py:329: UserWarning: Trying to unpickle estimator MinMaxScaler from version 1.0.2 when using version 1.1.1. This might lead to breaking code or invalid results. Use at your own risk. For more info please refer to:
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
  warnings.warn(
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
C:\Users\dhurvi patel\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\base.py:329: UserWarning: Trying to unpickle estimator DecisionTreeClassifier from version 1.0.2
 when using version 1.1.1. This might lead to breaking code or invalid results. Use at your own risk. For more info please refer to:
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
  warnings.warn(
C:\Users\dhurvi patel\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\base.py:329: UserWarning: Trying to unpickle estimator RandomForestClassifier from version 1.0.2
 when using version 1.1.1. This might lead to breaking code or invalid results. Use at your own risk. For more info please refer to:
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
```

Now, Go to the web browser and write the localhost url (http://127.0.0.1:5000/) to get the below results.



WELDPERFEX

Home About Predict

Discover our Essence and Vision

Explore the heart and soul of our organization, delving into our foundational principles and overarching mission.

WeldPerfex specializes in precision welding optimization, utilizing innovative machine learning to analyze vital welding parameters. Our advanced algorithms predict welding outcomes, ensuring structural integrity and performance while reducing defects. Trust WeldPerfex for unparalleled welding excellence, efficiently delivered with confidence.

01

Navigate to the Predict Section

Go to Predict section either by scrolling down the page or clicking on the Predict option located in the top right corner of the website.

02

Input the Necessary Parameters

You will need to provide the required parameters and details relevant to the prediction task.

03

Receive Your Results

Click "Submit" to initiate predictions. After submission, the algorithm will process inputs and generate predictions.

WELDPERFEX

Home About Predict

Predict

Kindly note that all welding optimization predictions are exclusively performed on stainless steel (SS) material.

Pressure (PSI): 35

Welding Time (ms): 200

Angle (Deg): 0

Force (N): 3.41

Current (A): 1337.45

Thickness A (mm): 0.922

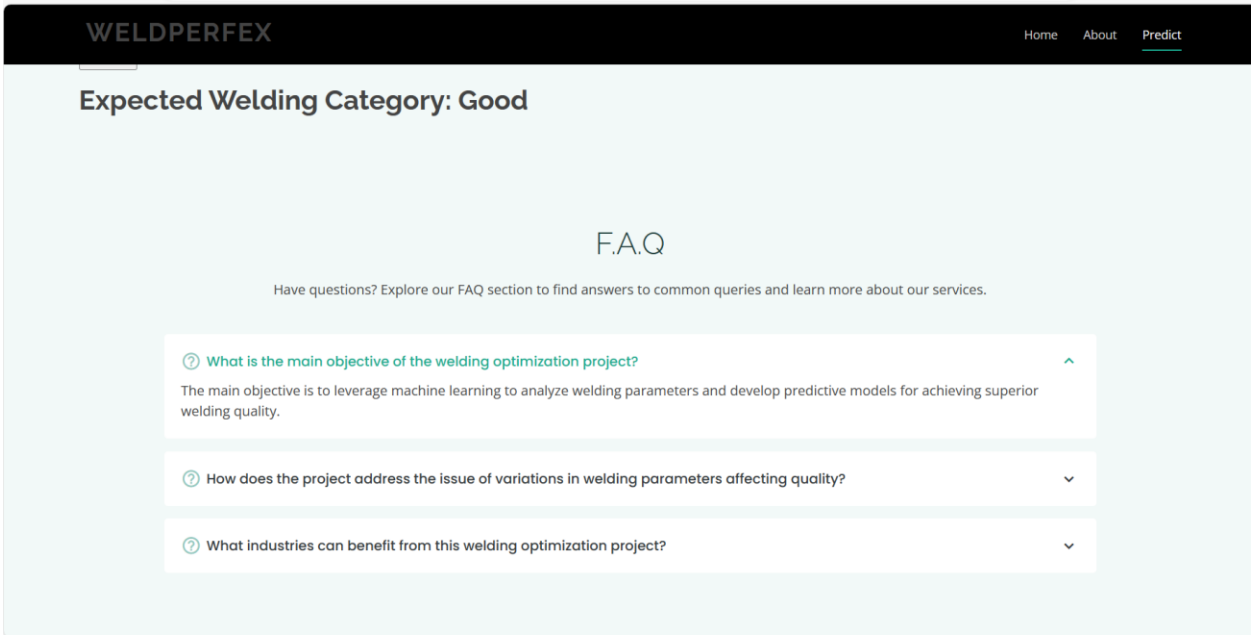
Thickness B (mm): 0.92

PullTest (N): 2127.7

NuggetDiameter (mm): 2.63

Submit

Expected Welding Category: Good


WELDPERFEX
Home
About
Predict
Expected Welding Category: Good
F.A.Q
Have questions? Explore our FAQ section to find answers to common queries and learn more about our services.
What is the main objective of the welding optimization project?
The main objective is to leverage machine learning to analyze welding parameters and develop predictive models for achieving superior welding quality.
How does the project address the issue of variations in welding parameters affecting quality?
What industries can benefit from this welding optimization project?