# Technical Documentation

**1. Introduction**

This document provides a detailed overview of the **end-to-end process** undertaken to build, evaluate, and deploy a machine learning model aimed at classifying data into two distinct classes. The project follows a systematic approach starting from **data collection**, **preprocessing**, **exploratory data analysis (EDA)**, **model development**, and concluding with the deployment using **Streamlit** for real-time predictions.

**2. Project Overview**

The dataset used in this project contains various features related to **industrial production**, such as material quantities, energy usage, time-based variables, and chemical compositions. The goal of the project was to apply machine learning algorithms to predict a target class based on these features.

Once the best-performing model was identified, **Streamlit** was used for deployment, enabling users to interact with the model through a **web interface** for real-time predictions.

**3. Data Exploration & Preprocessing**

**3.1 Initial Data Exploration:** The first step involved **Exploratory Data Analysis (EDA)** to understand the dataset better. We examined:

* The **distribution** of key features
* **Missing values** and how to handle them
* **Outliers** and the nature of data distribution (e.g., skewness, kurtosis)
* Relationships between features using **correlation analysis**

**3.2 Data Preprocessing:** The preprocessing steps included:

* **Handling Missing Values**: Imputed missing values using **forward-fill** and **backward-fill** strategies.
* **Feature Engineering**: Created additional time-based features (e.g., year, month, day) and computed time differences between important timestamps.
* **Duplicate Removal**: Checked and removed duplicates to ensure data integrity.
* **Zero Variance Columns**: Removed columns with no variance, such as constant features, that do not contribute useful information.
* **Feature Encoding**: One-hot encoded categorical variables to transform them into numerical format.
* **Data Scaling**: Used **RobustScaler** for scaling numerical data to handle outliers effectively.

**4. Model Development**

**4.1 Model Selection:** Several machine learning models were evaluated:

* **Logistic Regression**
* **SVM (Support Vector Machine)**
* **Random Forest**
* **Gradient Boosting**
* **AdaBoost**
* **K-Nearest Neighbors (KNN)**
* **Decision Tree**
* **Naive Bayes**
* **Ridge Classifier**
* **XGBoost (Chosen Model)**

**4.2 Model Training and Evaluation:** The models were trained and tested using the same dataset, and their performance was evaluated based on accuracy, precision, recall, F1-score, and confusion matrix. After evaluation, **XGBoost** was selected as the final model for the following reasons:

* **Highest test accuracy**: 97.09%
* **Balanced performance** across both classes with high precision and recall
* **Low overfitting** potential compared to models like Random Forest and Decision Trees

**5. Model Evaluation Metrics**

**5.1 Key Performance Metrics for XGBoost:**

* **Accuracy**: 97.09%
* **Precision**: High for both classes
* **Recall**: Balanced for both classes
* **F1-Score**: Excellent balance between precision and recall
* **Confusion Matrix**: Low number of false positives and false negatives

**6. Streamlit Deployment**

**6.1 Overview of Streamlit Deployment:** **Streamlit** was used to deploy the **XGBoost model** and provide an **interactive web interface** for real-time predictions. Streamlit is an open-source Python library that enables rapid development of machine learning web applications.

**6.2 Deployment Workflow:**

1. **Model Serialization**: After training the XGBoost model, we saved it using **joblib** to serialize the trained model so that it could be loaded into the Streamlit app for inference.
2. **Streamlit App**: The Streamlit app was built with:
   * An **input form** for users to enter relevant feature values
   * A **real-time prediction** function that loads the trained XGBoost model and makes predictions based on user input
   * **Visualizations** to display the input data and model results in an interactive manner
3. **Deployment**: The Streamlit app was deployed on a cloud service (e.g., **Streamlit Sharing**, **Heroku**, or **AWS**) to provide access to users for predictions.

**6.3 Streamlit Application Features:**

* **User Input**: Users can input values for the features (such as COKE\_REQ, INJ1\_QTY, etc.) through textboxes or sliders.
* **Prediction**: The model provides a real-time prediction of the target class based on the user’s input.
* **Results Display**: After the model prediction, results are displayed to the user, showing predicted class labels and probability scores.
* **Visualizations**: Visual plots, such as feature importance and predictions distribution, are provided to help users understand model behavior.

**7. Model Monitoring and Maintenance**

**7.1 Continuous Monitoring:** Once the model is deployed, it is essential to continuously monitor its performance. We will:

* Track **prediction accuracy** and ensure that the model performs well on new, unseen data.
* **Log any performance degradation** over time and retrain the model as needed.

**7.2 Retraining and Updates:** As new data becomes available, the model may need to be retrained to adapt to evolving trends in the data. This will involve:

* **Retraining the model periodically** using fresh data.
* Implementing a **continuous integration pipeline** for easy retraining and deployment.

**8. Recommendations for Future Improvements**

**8.1 Data Augmentation:** To further improve model performance, especially in cases of imbalanced data, we recommend exploring **data augmentation** techniques such as **SMOTE (Synthetic Minority Over-sampling Technique)** to balance the classes.

**8.2 Model Interpretability:** To enhance the transparency of the model’s decision-making, tools like **SHAP (SHapley Additive exPlanations)** can be implemented to provide detailed insights into the importance of different features in the model’s predictions.

**8.3 Hyperparameter Tuning:** Although the default parameters for **XGBoost** performed well, further **hyperparameter optimization** using grid search or random search could help fine-tune the model and achieve even better performance.

**9. Conclusion**

The **XGBoost model** has been successfully developed and deployed using **Streamlit**, providing an interactive and efficient platform for real-time predictions. Streamlit's user-friendly interface allows users to input features and receive predictions immediately. Continuous monitoring and retraining of the model will ensure that it remains effective as new data is collected.

The model has been selected based on its superior performance, handling of overfitting, and ease of deployment with **Streamlit**. With future improvements in interpretability and performance tuning, this system is expected to offer valuable insights and accurate predictions for the target classification task.