DESIGN A PREDICTIVE MAINTENANCE SYSTEM FOR AN INDUSTRIAL EQUIPMENT USING MACHINE LEARNING TECHNIQUES

By

M.K.T.T.D PERERA(E/18/257)

J.S.P.K. RAJAPAKSHA (E/18/270)

R.H.P.M RAJAPAKSHA (E/18/271)

ME 325-MECHANICAL ENGINEERING GROUP PROJECT

Presented in partial fulfillment of the requirements for the degree

of B.Sc. Engineering degree.

University of Peradeniya

06/07/2023

Supervised by:

Dr. D.H.S MAITRIPALA

Department of Mechanical Engineering

Faculty of Engineering

DECLARATION

We hereby declare that the work published in this report is the result of our own investigation, except where due references are made. It has not already been accepted for any other course and is also not being concurrently submitted for any other person.

E/18/257

M.K.T.T.D PERERA

signature of the candidate

_____

E/18/270

J.S.P.K. RAJAPAKSHA

signature of the candidate

_____

E/18/271

R.H.P.M. RAJAPAKSHA

signature of the candidate

_____

CERTIFICATION

Prof. D. A. A. C. Ratnaweera

Head of the Department

_____

signature

Dr. D. H. S. Maithreepala

Supervisor

_____

signature

Signature of the Examiner

_____

signature

Signature of the Examiner

_____

signature

Signature of the Examiner

_____

signature

ACKNOWLEDGEMENT

The group project, which was done to design a predictive maintenance system for industrial equipment using machine learning techniques, was supervised by Dr. D.H.S. Maithreepala from the University of Peradeniya We take this as an opportunity to acknowledge their commitment to the success of the project.

ABSTRACT

Predictive maintenance has emerged as a crucial approach for optimizing the performance and reliability of industrial systems. By leveraging machine learning techniques, predictive maintenance systems aim to accurately predict and prevent unexpected equipment failures, reduce downtime, and improve overall operational efficiency. In recent years, significant advancements have been made in machine learning algorithms, data collection technologies, and computational capabilities, which have enabled the development of sophisticated predictive maintenance systems. This thesis presents an in-depth exploration of various machine learning techniques for predictive maintenance, with a focus on their application in industrial settings. The research aims to investigate the effectiveness of different machine learning approaches for predicting equipment failures, analyze the impact of feature engineering, model selection, and hyperparameter tuning on the predictive performance, and evaluate the scalability and robustness of the developed models. The findings of this research can provide valuable insights for industries seeking to implement predictive maintenance systems and contribute to the advancement of machine learning techniques in the field of industrial maintenance. Traditional maintenance approaches can be inefficient and costly. Predictive maintenance uses machine learning algorithms to predict equipment failure and schedule maintenance accordingly. Predictive maintenance, powered by machine learning algorithms, can address these issues by providing a more accurate and efficient maintenance schedule. By analyzing real-time data from sensors and other sources, predictive maintenance models can predict when equipment is likely to fail and schedule maintenance, accordingly, reducing downtime and optimizing maintenance efforts. Therefore, the motivation for this project is to improve safety, reduce costs, and increase profitability for businesses through the use of predictive maintenance. The main aim of this project is to develop a machine learning model for predictive maintenance that can optimize maintenance schedules, reduce waste, and improve the bottom line of businesses. The sub aims include reducing unplanned downtime, optimizing maintenance schedules, increasing efficiency, and reducing waste.

The project will be divided into four stages: design phase, the implementation phase, the testing phase, and the evaluation phase. In the design phase, we will identify the relevant data sources, select the appropriate machine learning algorithms, and design the predictive maintenance model. The implementation phase involves data collection from various sources, data preprocessing to remove noise and outliers, feature engineering to extract relevant features from the data, model selection and training using the preprocessed data, and model evaluation on a validation set to optimize performance. The testing phase involves verifying model accuracy and completeness, evaluating the effectiveness of preprocessing steps, assessing the performance of engineered features, comparing different machine learning algorithms and hyperparameters, and monitoring the performance of the deployed model over time. The evaluation phase includes assessing the effectiveness of data preprocessing, feature engineering, model selection and training, and model evaluation using appropriate visualization tools and metrics. The justification for choosing evaluation metrics and visualization tools will be based on project objectives and requirements. The evaluation results will demonstrate the effectiveness of the developed predictive maintenance system in improving efficiency and reducing costs.

Table of Contents

List of Figures

Chapter 01: Introduction

1.1 Background

In today's industrial settings, unscheduled downtime and emergency maintenance of critical equipment can result in significant financial losses and operational disruptions. To address this challenge, a project has been initiated to develop a machine learning-based predictive maintenance system. This system aims to leverage the power of data analytics and automation to reduce costs associated with unscheduled downtime and emergency maintenance while improving overall operational efficiency.

The project recognizes the value of harnessing relevant data from industrial equipment to gain insights into its health and performance. By collecting and preprocessing this data, the project team aims to extract valuable information that can be used to predict equipment failures and proactively schedule maintenance activities. This approach shifts maintenance practices from reactive to proactive, allowing for more efficient resource allocation and minimizing costly disruptions caused by unexpected equipment breakdowns.

To ensure the effectiveness of the predictive maintenance system, the project team will employ feature engineering techniques to derive meaningful characteristics from the collected data. These features will capture the equipment's operating conditions, performance trends, and any early warning signs of potential failures. By carefully selecting and engineering these features, the machine learning algorithm can better understand and correlate the data patterns with equipment failures.

The core component of the project involves training a machine learning algorithm using historical data on equipment failures and maintenance records. This training phase will enable the algorithm to learn the underlying patterns and relationships between the collected features and equipment failure events. By leveraging this knowledge, the algorithm will be capable of predicting future failures with a reasonable level of accuracy.

Upon successful training, the machine learning model will be deployed as part of a working system. The system will integrate seamlessly with existing maintenance and repair systems, automating the maintenance process and enhancing overall efficiency. Upon further development and deployment, the system will continuously analyze real-time data from industrial equipment, provide actionable insights, and generate alerts or notifications when potential failures are anticipated. These predictive capabilities will enable maintenance teams to intervene proactively, scheduling repairs or replacements during planned downtime periods, thus reducing the impact on production schedules and minimizing costs.

Overall, the project aims to revolutionize traditional maintenance practices by leveraging machine learning and data analytics to predict equipment failures before they occur. By integrating the predictive maintenance system into existing infrastructure, industries can optimize maintenance processes, reduce unscheduled downtime, lower emergency maintenance costs, and improve operational efficiency.

Figure 1: Milling machine

1.2 Methodologies

1.2.1 Design phase

The proposed solution is to develop a machine learning-based predictive maintenance system for industrial equipment. The system will use a pre-recorded dataset, train a machine learning algorithm, and deploy the model in a production environment. The system will be updated to ensure accurate predictions and will be integrated with existing maintenance and repair systems.

Machine learning-based predictive maintenance provides accurate and reliable predictions, enabling companies to optimize their maintenance schedules and reduce costs associated with equipment failure.

Other maintenance solutions like preventative maintenance, reactive maintenance, and condition-based maintenance do not provide the same level of accuracy and reliability as machine learning-based predictive maintenance.

1.2.2 Implementation phase

1. Data Collection: Relevant data will be collected from sources such as maintenance logs, repair records, online material, and equipment manuals.

2. Data Preprocessing: The collected data will be preprocessed to remove noise and outliers, fill in missing data, normalize and standardize the data, and transform it into a format suitable for machine learning.

3. Feature Engineering: Relevant features will be engineered from the preprocessed data, such as the mean, standard deviation, and trend of sensor readings over a certain time period.

4. Model Selection and Training: A machine learning algorithm will be selected based on the problem domain and trained using the preprocessed and engineered data.

5. Model Evaluation: The trained model will be evaluated on a validation set to ensure it performs well on unseen data. Hyperparameters will be tuned to optimize model performance.

1.2.3 Testing phase

To ensure the effectiveness of the machine learning-based predictive maintenance system, various sections or aspects of the project need to be tested. First, the accuracy and completeness of the collected data should be verified. Secondly, the effectiveness of preprocessing steps, such as noise removal and missing data filling, should be evaluated. Thirdly, the effectiveness of the engineered features should be evaluated. Fourthly, the performance of different machine learning algorithms and hyperparameters should be compared, and the model should be evaluated on a validation set. Fifthly, the performance of the deployed model should be monitored to ensure accurate predictions over time.

1.2.4 Evaluation phase

To evaluate the results of the developed machine learning-based predictive maintenance system, the following aspects of the project can be evaluated:

- Data Preprocessing: The effectiveness of the preprocessing steps, such as noise removal and missing data filling, can be evaluated by comparing the preprocessed data to the raw data. This can be done using visualization tools such as line plots, scatter plots, and histograms.

- Feature Engineering: The effectiveness of the engineered features can be evaluated by comparing the performance of the model with and without the engineered features. This can be done by plotting the prediction accuracy or error rate of the model.

- Model Selection and Training: The performance of different machine learning algorithms and hyperparameters can be compared by evaluating the model on a validation set. This can be done by plotting the prediction accuracy or error rate of the model.

- Model Evaluation: The performance of the model can be evaluated on a test set by calculating the accuracy, precision, recall, and F1 score. This can be done using confusion matrices and ROC curves.

- Model Deployment: The performance of the deployed model can be monitored to ensure it continues to make accurate predictions over time. This can be done by plotting the prediction accuracy or error rate of the model over time.

The evaluation results can be presented using appropriate visualization tools such as line plots, scatter plots, histograms, confusion matrices, and ROC curves. The justification for the chosen evaluation metrics and visualization tools will be based on the project objectives and requirements. The evaluation results will demonstrate the effectiveness of the developed predictive maintenance system in improving efficiency and reducing costs.

1.3 Outlines of Results

- **Features of the Project:**

Real-time Predictive Maintenance is at the forefront of modern industrial practices, revolutionizing equipment maintenance strategies. With this innovative system in place, industries can now benefit from real-time predictions of equipment failures, which empower them to take proactive measures and mitigate potential downtime risks. By continuously monitoring the health of machinery and analyzing the data collected, advanced machine learning techniques play a pivotal role in identifying patterns and early warning signs of equipment failure. This early detection capability proves instrumental in preventing catastrophic breakdowns and costly repairs.

One of the key advantages of this system is its scalability, as it can be easily tailored to suit the specific needs of different industrial setups. Whether it's dealing with a limited number of sensors or managing large-scale equipment, the system efficiently adapts, ensuring consistent and reliable performance across various applications.

Moreover, the cost-effectiveness of the Real-time Predictive Maintenance system is a major driving force behind its widespread adoption. By significantly reducing the frequency of unexpected failures and unplanned maintenance activities, the system substantially minimizes maintenance costs. This leads to enhanced overall productivity, decreased downtime, and ultimately boosts the company's profitability.

- **Benefits of the Project:**

The implementation of a machine learning-based predictive maintenance system promises substantial benefits for industrial operations, particularly in terms of equipment reliability and efficiency. By enabling early detection of potential issues before they escalate into equipment failures, the system significantly reduces downtime, thereby enhancing overall equipment reliability. This improvement translates into increased productivity and improved profitability for the industry.

Another crucial advantage of the system lies in its ability to boost operational efficiency. Automating data collection and processing streamlines maintenance processes, allowing real-time predictions of equipment failure. Consequently, this automation minimizes the need for manual intervention, freeing up valuable human resources for more strategic and value-added tasks.

Cost savings represent another key aspect of this advanced predictive maintenance approach. Through proactive maintenance enabled by the system's timely predictions, industries can minimize unplanned downtime and mitigate expensive emergency repairs. As a result, the overall maintenance and repair expenses are reduced, leading to substantial cost savings in the long run.

The system's scalability further enhances its appeal for various industrial applications. Regardless of the number of sensors or the size of the equipment, the system can easily adapt to different settings, ensuring seamless performance across diverse environments.

1.4 Future Scopes

The development of a machine learning-based predictive maintenance system for industrial equipment presents a wide range of future opportunities and possibilities. Here are some potential areas of future scope for this project:

1. Advanced Analytics: Enhance the existing predictive maintenance system by incorporating more advanced analytical techniques such as deep learning, reinforcement learning, or anomaly detection algorithms. These advanced methods can further improve the accuracy and reliability of equipment failure predictions.

2. IoT Integration: Integrate the predictive maintenance system with Internet of Things (IoT) devices and sensors installed on industrial equipment. This integration would enable real-time data collection and monitoring of equipment health, allowing for more accurate and timely predictions of failures.

3. Prescriptive Maintenance: Expand the system's capabilities to include prescriptive maintenance recommendations. By considering various factors such as cost analysis, inventory availability, and maintenance team schedules, the system could provide optimal recommendations on repair or replacement actions to minimize downtime and cost while maximizing overall operational efficiency.

4. Scalability and Adaptability: Develop the system to handle a broader range of industrial equipment types and industry domains. This would involve creating a scalable architecture that can accommodate different equipment characteristics and adapt to various maintenance requirements across industries such as manufacturing, energy, transportation, and more.

5. Integration with ERP and CMMS Systems: Integrate the predictive maintenance system with Enterprise Resource Planning (ERP) systems and Computerized Maintenance Management Systems (CMMS). This integration would enable seamless data exchange, streamlined workflows, and automated work order generation for maintenance activities based on the system's predictions.

6. Proactive Supply Chain Management: Extend the capabilities of the system to incorporate supply chain management aspects. By predicting equipment failures in advance, the system can facilitate proactive procurement and inventory management, ensuring the availability of spare parts and reducing supply chain disruptions.

7. Continuous Model Monitoring and Updating: Implement mechanisms for continuous model monitoring and updating to ensure the system's predictive capabilities remain accurate and up to date. Regular monitoring of model performance, data drift detection, and retraining cycles will help maintain the system's reliability over time.

8. Integration with Predictive Analytics Platforms: Integrate the predictive maintenance system with broader predictive analytics platforms or frameworks. This integration would enable organizations to leverage the system's predictive insights alongside other data-driven analytics initiatives, fostering a more comprehensive and holistic approach to data-driven decision-making.

9. Predictive Maintenance as a Service: Explore the potential for offering the predictive maintenance system as a service, where industrial organizations can leverage the system's capabilities without the need for extensive in-house development and maintenance. This would allow businesses of all sizes to benefit from predictive maintenance technologies and drive broader adoption across industries.

10. Continuous Improvement and Feedback Loop: Establish a feedback loop to collect data and insights from maintenance activities and actual equipment failures. This feedback loop would enable continuous improvement of the system's predictive algorithms and further refine its capabilities based on real-world performance and outcomes.

These future scopes have the potential to enhance the predictive maintenance system's effectiveness, expand its application areas, and drive innovation in the field of industrial equipment maintenance, leading to significant cost savings, increased productivity, and improved operational efficiency for industries.

Chapter 2: Literature Review

2.1 Machine Learning Applications in Predictive Maintenance for Industry 4.0 and Smart Manufacturing

Predictive maintenance is a key strategy in Industry 4.0 and smart manufacturing, utilizing intelligent sensors and machine learning to identify machinery faults before they become critical. Its objective is to ensure machine availability and prevent machine-down failures. The rise of technologies like automated predictive maintenance and intelligent sensors in smart factories has led to a greater need for efficient data analysis to support decision-making and complex system management.

Numerous studies have focused on predictive maintenance in different areas. In manufacturing, machine learning methods have been successfully employed for fault detection and diagnosis in chemical processes, outperforming traditional maintenance approaches. Machine learning algorithms, including artificial neural networks and decision trees, have also been used to diagnose and predict faults in manufacturing equipment, with empirical results from industrial case studies demonstrating their practicality.

In building facilities, a predictive maintenance framework based on machine learning has been proposed, guiding the implementation of predictive maintenance in building installations. A case study in a sports facility showcased the framework's effectiveness using IoT devices and a building automation system, employing a deep learning model to predict failures.

Similarly, in the machining process and machine tool domain, a data-driven predictive maintenance approach has been implemented, transitioning from preventive to predictive maintenance by visualizing and analyzing the remaining useful life of machining tools. The study demonstrated the feasibility of this methodology in a real machining process.

In summary, the literature review emphasizes the increasing importance of intelligent sensors and data-driven approaches in predictive maintenance. Machine learning techniques have been successfully applied in various domains, including manufacturing, building facilities, and machining processes, improving maintenance practices, reducing downtime, and enhancing equipment reliability in the era of Industry 4.0.

2.2 Artificial Intelligence

2.2.1 Machine learning

Artificial intelligence includes machine learning as a subset. Machine learning is the process of learning from large amounts of data. By training on data sets, machine learning can do predictive analysis more quickly. With larger data sets, predictions can be made more accurately. Although machine learning predictions have a statistical base, machine learning can make different assumptions than statistics, making the forecasts more accurate. Machine learning techniques have a significant impact on artificial intelligence due to their very accurate predictions. When it comes to large projects, machine learning also has limitations. Analyzing large data sets for accurate predictions always necessitates a considerable amount of memory on the machines. Machine processors and speeds must be technologically improved in order to make more accurate predictions.

Machine learning's ultimate goal is to combine technologies and tactics to build a single algorithm that accomplishes a given task. Machines are taught by creating an algorithm that reduces data inputs and expected results to a function in every situation, but the function is specific to the type of task you want the algorithm to accomplish. The purpose of machine learning is to generalize the output function so that the system may produce outcomes that are not limited to the trained set. Algorithms are developed as a function of specified inputs, and the algorithm must produce the appropriate outputs using any input data. This is known as data representation. Similarly, utilizing the training data, a variety of algorithms can be created. The outputs of built algorithms can then be tested using data inputs. This is referred to as evaluation. By observing the results, the best model or algorithm can be selected during the evaluation phase. Evaluation metrics can be used to compare the results. The algorithm's next stage of development is optimization. During this stage, the learner attempts to make more accurate predictions.

The four main classifications of machine learning are supervised learning, semi-supervised learning, unsupervised learning, and reinforcement. During our endeavor, we focused heavily on supervised learning. When supervised learning is used, the machines are trained by examples. The operator delivers known inputs and outputs to the machine, and the computer is tasked with finding an algorithm that recognizes patterns in the data sets. Similarly, the machines are given input and output data sets until they can generate outputs for unknown inputs with greater accuracy. The three main subsets of supervised machine learning are classification, regression, and forecasting. The goal of classification algorithms is to categorize the input training data. For binary classification, classification algorithms are utilized. A regression algorithm is a machine learning technique that predicts outputs as continuous numerical values.

2.2.2 Classification algorithms

Support Vector Machine

Support Vector Machines (SVM) are a powerful and widely used supervised machine learning algorithm for classification tasks. They are particularly effective for dealing with complex and high-dimensional data. SVMs are based on the idea of finding the optimal hyperplane that best separates different classes in the input data. The primary objective of SVM is to create a decision boundary, which is a hyperplane in a multidimensional feature space. This hyperplane aims to maximize the margin or distance between data points of different classes. The data points that are closest to the decision boundary are known as support vectors, and they play a crucial role in determining the hyperplane. One of the key strengths of SVM is its ability to handle non-linearly separable data by using a technique called the kernel trick. The kernel trick allows SVM to implicitly transform the original feature space into a higher-dimensional space, where the data points may become linearly separable. Common kernel functions include polynomial kernels, radial basis function (RBF) kernels, and sigmoid kernels, each suitable for different types of data distributions.

Random Forest

Random Forest is a powerful and widely used ensemble learning algorithm for classification tasks in machine learning. It is an extension of decision trees and combines the predictions from multiple individual decision trees to improve accuracy and reduce overfitting. Random Forest is renowned for its robustness, versatility, and ability to handle high-dimensional data effectively. The Random Forest algorithm works by creating a collection of decision trees during the training phase. Each tree is built using a random subset of the original features and a random subset of the training data, a process known as bagging or bootstrap aggregation. This randomness helps to introduce diversity among the individual trees, which is a key factor in the algorithm's success. During the classification phase, each tree in the Random Forest independently predicts the class label for a given input, and the final classification is determined through a voting mechanism. In a majority vote, the class with the most votes from the individual trees becomes the predicted class label.

Decision Tree

The Decision Tree algorithm is a fundamental and intuitive machine learning model used for classification tasks. It is widely employed in various domains due to its simplicity, interpretability, and ease of understanding. Decision trees are particularly useful when dealing with discrete and categorical data, making them a popular choice for both beginners and experienced machine learning practitioners. The Decision Tree algorithm works by recursively partitioning the input data into subsets based on the values of different features. At each node of the tree, the algorithm selects the feature that best splits the data, optimizing a criterion such as Gini impurity or information gain. This process is performed in a way that maximizes the purity of each subset with respect to the target class labels. During the classification phase, an input sample traverses the decision tree from the root to a leaf node based on the feature values. The leaf node reached by the sample represents the predicted class label for that input. Each path from the root to a leaf corresponds to a decision rule, and the entire tree encapsulates the decision-making process.

CatBoost

CatBoost is a powerful and efficient gradient boosting algorithm designed for handling categorical features in classification tasks. It is an extension of the traditional gradient boosting technique and is known for its ability to automatically handle categorical data without requiring explicit preprocessing. Developed by Yandex, CatBoost has gained popularity among machine learning practitioners due to its high performance, robustness, and ease of use. The CatBoost algorithm uses a combination of gradient boosting and ordered boosting techniques to build an ensemble of weak learners, typically decision trees, in an iterative manner. During the training process, CatBoost handles both numerical and categorical features effectively, as it performs category-specific statistics-based preprocessing and target encoding. This allows the algorithm to naturally handle categorical features without the need for manual label encoding or one-hot encoding. One of the key features of CatBoost is its ability to handle high-cardinality categorical features with large numbers of unique categories, which can be challenging for other gradient boosting algorithms. CatBoost utilizes a combination of ordered boosting and feature discretization techniques to overcome the potential memory and computational issues caused by high cardinality.

K-Nearest Neighbor (KNN)

The k-Nearest Neighbor (k-NN) algorithm is a simple and intuitive machine learning model used for classification tasks. It falls under the category of instance-based learning or lazy learning, as it does not build an explicit model during the training phase. Instead, k-NN makes predictions based on the similarity between the new input and the labeled instances in the training data. The value of k, a hyperparameter, determines the number of neighbors considered for making predictions. A larger k can lead to a smoother decision boundary, reducing the impact of noisy data points, but it can also blur the boundaries between classes. Conversely, a smaller k can result in a more flexible decision boundary but may be sensitive to outliers.

Gradient Boosting

Gradient Boosting is a powerful and widely used ensemble learning technique for classification tasks in machine learning. It is known for its ability to build highly accurate predictive models by combining the predictions of multiple weak learners, typically decision trees, in a sequential manner. Gradient Boosting is particularly effective for both binary and multi-class classification problems and has gained popularity due to its robustness and versatility. One of the key components of Gradient Boosting is the use of a loss function, such as the cross-entropy loss for classification, to quantify the errors made by the model during training. The algorithm seeks to minimize this loss function by adjusting the parameters of the weak learners in each iteration.

Linear Regression

By applying a linear equation to observed data, linear regression aims to illustrate the connection between two variables. One variable is intended to be independent, while the other is supposed to be dependent. For instance, a person's weight is proportional to his height. As a result, there is a linear link between the person's height and weight. The weight of a person increases in proportion to his or her height. It is not required that one variable be reliant on another or that one cause the other, but there must be some essential connection between the two variables. In these instances, a scatter plot is used to indicate the strength of the association

between the variables. The scatter plot does not show any rising or decreasing pattern if there is no relationship or linkage between, the data. The linear regression design is not advantageous to the provided data in such circumstances.

2.2.3 Evaluating metrics

Evaluation is the selection of the model that produces the most accurate results. Model evaluation is a critical aspect of data science analysis. There are numerous evaluation metrics used in data science, however, some are exclusive to regression methods. They are,

- Accuracy
- Precision
- Recall (Sensitivity or true positive rate)
- F1-Score
- Confusion matrix

In multiclass classification problems, various evaluation metrics are utilized to gauge the performance of the model in predicting multiple class labels. These metrics include accuracy, which measures the proportion of correct predictions overall, and precision, which assesses the model's ability to avoid false positives. Recall, also known as sensitivity or true positive rate, evaluates the model's capability to capture all positive instances. The F1-Score, representing the harmonic mean of precision and recall, provides a balanced metric for overall performance. Additionally, the confusion matrix summarizes true positives, false positives, true negatives, and false negatives for each class, aiding visualization of classification results.

2.2.4 Time performance analysis

Time performance analysis of a machine learning model involves measuring and evaluating the time taken for various stages of the model's lifecycle, including data preprocessing, training, hyperparameter tuning, and testing. Analyzing the time performance is crucial to understand the computational complexity and scalability of the model, which can impact its practical usability and deployment.

- Model Training Time: Evaluate the time taken to train the model on the training dataset. This includes the time taken for updating the model's parameters using an optimization algorithm, such as gradient descent, during training.
- Hyperparameter Tuning Time: If hyperparameter tuning is performed, track the time taken to explore different hyperparameter configurations and select the best

combination. Techniques like grid search or random search can be time-consuming, especially for models with a large number of hyperparameters.

- Model Evaluation Time: Measure the time taken to evaluate the model's performance on the test dataset. This involves predicting the output labels and comparing them to the ground truth labels.

2.3 Neural Network

In recent years, neural networks have provided solutions in a variety of sectors. Neural networks have boosted image processing, speech recognition, and other artificial intelligence areas. A neural network is a computational machine learning system that uses a network of functions to output the required data from input data. The neural network learning algorithm learns from training data, which is then used to anticipate the final results. The quantity of examples or training data has a direct influence on the quality of the final outputs, therefore, the more the neural network sees, the better the accuracy. Node layers, input layers, several hidden layers, and an output layer comprise artificial neural networks. The nodes communicate with one another to build a data analysis network. After the input layer is designed, the weights are applied to the nodes. The inputs are multiplied by the set weights, and the resulting output is passed via an activation function. This method of sending data through the tiers is known as a feed forward network. Over time, neural networks' accuracy will improve. Learning algorithms get more powerful with time, culminating in a customized neural network that produces more accurate results.

2.3.1 Optimization of neural networks using hyperparameters

Because of many critical properties, hyperparameter optimization is required in neural network development. To get the optimal balance of bias and variance: It is very easy to achieve very high accuracy while training our data using dense neural networks, but these may not even generalize well to our validation and test sets; additionally, avoiding using deep/complex architectures risks having low accuracy on our data-sets; thus, we must find the sweet spot that generalizes well and has a high accuracy. Each hyper-parameter influences the bias-variance. To avoid the disappearing/exploding gradient problem, hyperparameter tuning is required; the back-propagation step may require the multiplication of numerous gradient values; if the gradient values are small (1), we may experience the vanishing gradient problem; if they are large(1), we may experience the exploding gradient problem. This can be avoided by fine-tuning the learning rate, activation function, and number of layers while modifying the hyperparameters. Some of the most important hyper-parameters utilized in hyper parameter tuning are listed below. These hyper parameters are,

1. Number of Layers

2. Number of hidden units per layer
3. Activation Function
4. Optimizer
5. Learning Rate
6. Initialization
7. Batch Size
8. Number of Epochs
9. Dropout
10. L1/L2 Regularization

Chapter 3: Methodology

3.1 Tools Used

3.1.1 Google colab

Google Colab, short for "Google Collaboratory," is a cloud based Jupyter notebook environment provided by Google. It allows users to run and execute Python code directly in a web browser without the need for any local installation. Google Colab is integrated with Google Drive, enabling users to save and share their notebooks seamlessly.

3.1.2 Python

Python is a high-level, interpreted, and general-purpose programming language. It was created by Guido van Rossum and first released in 1991. Python emphasizes readability, simplicity, and a clean syntax, which makes it a popular choice for beginners and experienced programmers alike. It has been mainly used for analysis and training of the machine learning models and neural networks.

Figure 2: Python logo

3.1.3 Anaconda Package

Anaconda is a popular open-source distribution of the Python and R programming languages used for data science, machine learning, scientific computing, and other related tasks. It is designed to simplify package management and deployment by providing a comprehensive ecosystem of pre-installed libraries, tools, and packages for data analysis and scientific computing.

3.1.4 Keras Library

Keras is an open-source high-level neural networks API written in Python. It is designed to provide a user-friendly and intuitive interface for building and experimenting with deep learning models. Developed with a focus on enabling fast experimentation, Keras allows users to quickly design and train neural networks for tasks such as image classification, natural language processing, and more. While originally a standalone library, Keras has been integrated as part of TensorFlow's official API, making it an integral part of the TensorFlow ecosystem and one of the most widely used deep learning frameworks in the industry.

3.1.5 TensorFlow

TensorFlow is an open-source machine learning library developed by the Google Brain team. It provides a comprehensive ecosystem for building and deploying machine learning models. With a strong focus on deep learning, TensorFlow allows users to create complex neural networks for tasks like image recognition, natural language processing, and reinforcement learning. Its flexible architecture enables easy deployment across different platforms, including CPUs, GPUs, and TPUs, making it a widely adopted and powerful framework for a broad range of machine learning applications.

3.2 Python Libraries

3.2.1 NumPy

NumPy is a Python library for numerical computing, providing support for multi-dimensional arrays and mathematical functions. Its usage in creating a machine learning system involves data preprocessing, conversion of data into arrays, and array-based operations for data transformation and normalization. It also serves as a backbone for implementing various machine learning algorithms, enabling efficient matrix operations for training and optimizing models.

In a machine learning system, NumPy plays a crucial role in handling large datasets, conducting mathematical operations on arrays, and facilitating data manipulation and transformation. Its array-based functionality allows for efficient implementation of machine learning algorithms like linear regression, logistic regression, and neural networks. Overall, NumPy is a vital tool in the Python data science ecosystem, enhancing the efficiency and effectiveness of creating machine learning systems.

Figure 3: NumPy logo

3.2.2 SciPy

SciPy is another fundamental Python library for scientific computing that builds on top of NumPy. It provides a wide range of functions for advanced mathematical, scientific, and engineering computations. SciPy includes modules for optimization, integration, interpolation, linear algebra, signal processing, statistics, and much more, making it a powerful toolkit for various scientific applications, including machine learning.

In creating a machine learning system, SciPy's diverse functionalities come into play for tasks like statistical analysis, feature engineering, and signal processing. The optimization module can be used for fine-tuning machine learning models by finding optimal parameters. SciPy's interpolation capabilities are useful for filling missing data, and its integration module aids in solving differential equations, relevant for certain machine learning models. Overall, SciPy complements NumPy and enhances the capabilities of Python's scientific computing ecosystem, providing a comprehensive toolset for implementing complex machine learning algorithms and conducting in-depth data analysis.

Figure 4: Scipy logo

3.2.3 Matplotlib

Matplotlib is a popular Python library used for creating 2D plots and visualizations. It provides a wide range of plotting options, including line plots, scatter plots, bar plots, histograms, and more. Matplotlib's flexible and intuitive interface allows users to customize and fine-tune various aspects of their plots, such as colors, labels, axes, and annotations, making it a versatile tool for visualizing data in a visually appealing and informative manner.

In creating a machine learning system, Matplotlib plays a vital role in data exploration and analysis. It is used to visualize datasets, helping users understand the distribution of data and identify patterns and trends. For instance, scatter plots can display the relationship between two variables, while histograms can show the distribution of a single variable. During model evaluation, Matplotlib aids in plotting performance metrics like accuracy, precision, recall, and ROC curves to assess the model's performance and make informed decisions about its suitability.

Matplotlib also allows users to create interactive visualizations, which can be helpful when presenting machine learning results or sharing insights with stakeholders. By providing an easy-to-use and flexible plotting interface, Matplotlib enhances the communication of complex information and results, making it an essential tool in the data science and machine learning workflow.

Figure 5: Matplotlib logo

3.2.4 Pandas

Pandas is a powerful Python library for data manipulation and analysis, centered around its primary data structure, the Data Frame. It excels at handling structured data, making it a crucial tool for data preprocessing in machine learning projects. With Pandas, users can efficiently clean, filter, transform, and aggregate data, addressing missing values and duplicates to ensure data quality. Its ability to perform complex data operations simplifies feature engineering, enabling users to derive new meaningful features from existing data. Moreover, Pandas' integration with other data science libraries like NumPy and Matplotlib allows for seamless data analysis and visualization, streamlining the machine learning workflow and facilitating insightful data exploration.

In machine learning systems, Pandas is indispensable for preparing data before model training and evaluation. It enables data scientists and machine learning practitioners to perform exploratory data analysis, handle missing data, and engineer features effectively, improving the overall quality of the dataset. With its intuitive and efficient functions, Pandas has become a fundamental tool for data wrangling and analysis, contributing significantly to the success of machine learning projects by providing a solid foundation for data handling and manipulation.

Figure 6: Pandas logo

3.2.5 Seaborn

Seaborn is a Python data visualization library that enhances the capabilities of Matplotlib, offering a higher-level interface for creating visually appealing and informative statistical plots. Its extensive collection of plot types and color palettes allows users to create sophisticated visualizations with ease. Seaborn excels at visualizing complex datasets, making it an invaluable tool for data exploration, pattern discovery, and insight generation. Its seamless integration with Pandas Data Frames streamlines data visualization workflows and simplifies the process of plotting relationships between variables. Whether it's scatter plots, bar plots, or kernel density plots, Seaborn's default settings and aesthetically pleasing design choices produce compelling visualizations, that can be further customized to match specific data analysis requirements.

In the realm of machine learning, Seaborn proves to be a valuable asset for data scientists and researchers. It aids in understanding data distributions, detecting outliers, and identifying correlations among features. Seaborn's ability to visualize model performance metrics and evaluation results enables data scientists to assess and fine-tune machine learning models effectively. Moreover, the visual insights provided by Seaborn contribute to more informed decision-making in model selection and parameter tuning, thereby enhancing the overall quality and performance of machine learning systems. With its user-friendly interface and powerful visualization capabilities, Seaborn has become a staple in the data science toolbox, facilitating clearer communication of complex data patterns and driving impactful discoveries in various machine learning applications.

Figure 7: Seaborn logo

3.2.6 Scikit learn

Scikit-learn is a powerful and widely used machine learning library in Python, providing a rich set of tools for various tasks in data science and machine learning. Its intuitive and consistent API allows users to easily implement classification, regression, clustering, and other machine learning algorithms. With a comprehensive collection of preprocessing and evaluation functions, Scikit-learn streamlines the data analysis workflow, making it easier for researchers and practitioners to preprocess datasets, train models, and assess their performance accurately.

The library's versatility and extensive documentation have contributed to its popularity in the data science community. Its user-friendly interface and efficient implementation of algorithms make it a valuable resource for both beginners and experienced machine learning professionals. Scikit-learn's widespread adoption and active development community continue to drive its growth, ensuring that it remains a go-to choice for machine learning tasks and research projects.

Figure 8: Scikit learn logo

3.2.7 Pandas profiling

Pandas Profiling is an open-source Python library that provides an automated and comprehensive exploratory data analysis report for a given dataset. It offers a quick and efficient way to understand the structure and characteristics of the data, making it a valuable tool for data exploration and initial data quality assessment. By generating interactive HTML reports, Pandas Profiling presents a wide range of statistics and visualizations, including data types, missing values, correlation matrices, histograms, and more. This detailed overview allows data scientists and analysts to identify patterns, anomalies, and potential issues in the data, aiding them in making informed decisions on data preprocessing and modeling steps.

With Pandas Profiling, users can rapidly gain insights into large datasets without the need to write extensive code or manually create individual plots. The library is highly customizable, allowing users to configure the level of detail and the specific visualizations they want in the report. Its ability to handle both numerical and categorical data makes it suitable for a wide variety of datasets. Overall, Pandas Profiling is a valuable addition to the data science toolkit, as it expedites the initial data exploration process and empowers data scientists with valuable information to start their analysis efficiently.

Figure 9: Pandas profiling logo

3.3 Data Pre-processing

3.3.1 Data Filtration

The data set used is a synthetic dataset modeled after an existing milling machine and consists of 10,000 data points stored as rows with 10 features in columns by Prof. Stephen Matzka of the Berlin University of Applied Science. Each data column of the data set was checked using Python to ensure if there were any missing data points, and there were no missing data points. The boxplots were created using the Seaborn library to manually check the distribution and ranges of the predictor variables. The boxplots in Figure 10 showed that all the predictors are not in the same range, and therefore outliers should be removed and normalized.

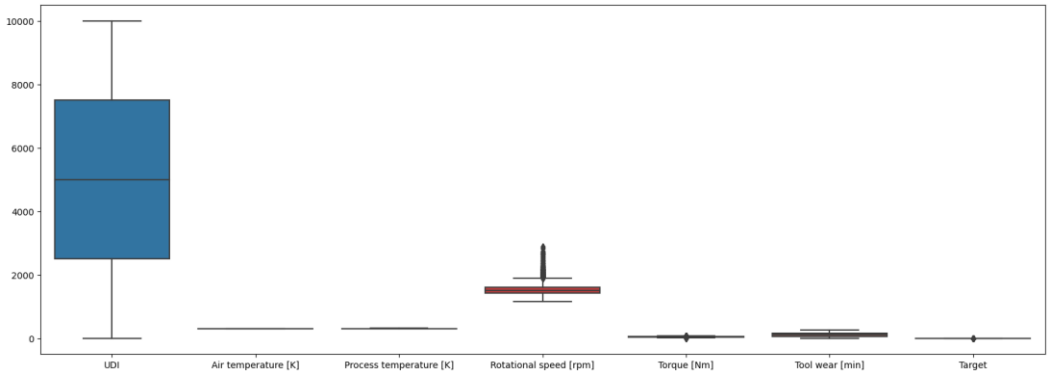


Figure 10: Boxplots generated for each feature

Histograms are graphical representations that illustrate the distribution of a variable's values. They divide the range of values into discrete intervals, known as bins, and display the frequency or count of observations falling into each bin. Histograms provide insights into the shape, central tendency, spread, and presence of outliers within the data. Histogram plots can aid in identifying outliers by visually examining the distribution of data. Outliers often appear as data points that lie outside the expected range of values, leading to deviations from the main body of the histogram. By observing the tails, or extreme ends, of the histogram, potential outliers can be identified for further investigation. When using histogram plots for outlier detection, several factors should be considered. The width of bins can impact the visibility of outliers, so choosing an appropriate bin width is essential. Skewness (asymmetry) and kurtosis (Peakness) of the histogram can indicate the presence of outliers, especially in skewed or heavy-tailed distributions. Additionally, multimodal distributions with multiple peaks might suggest the existence of outliers or distinct groups within the data.

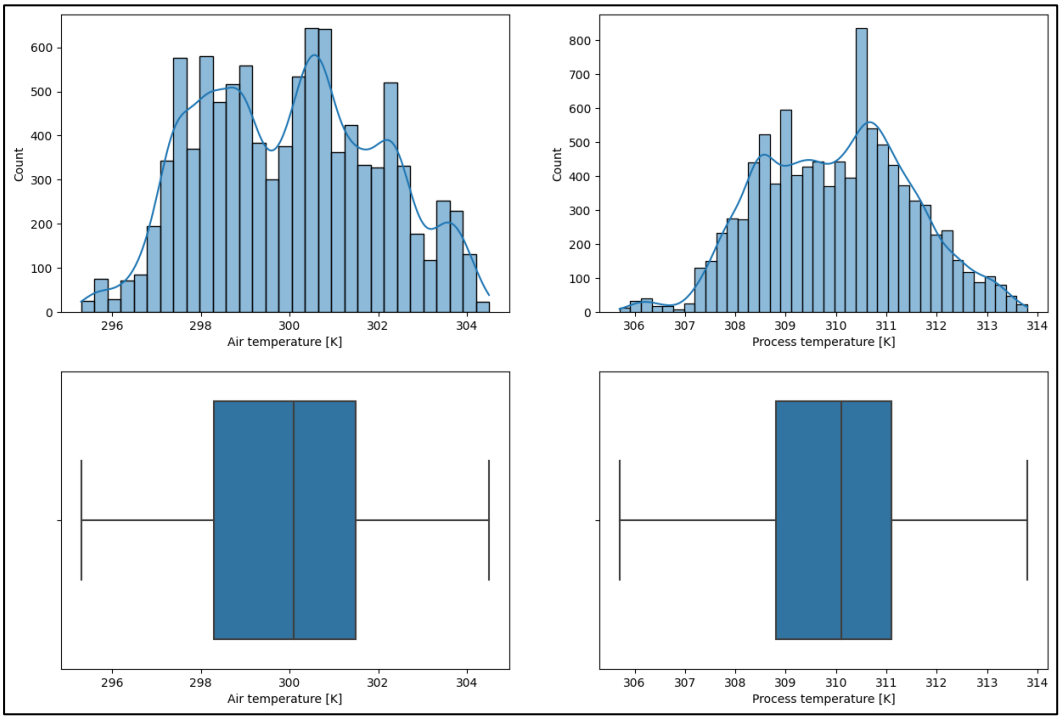


Figure 11: Histograms generated for each feature 1

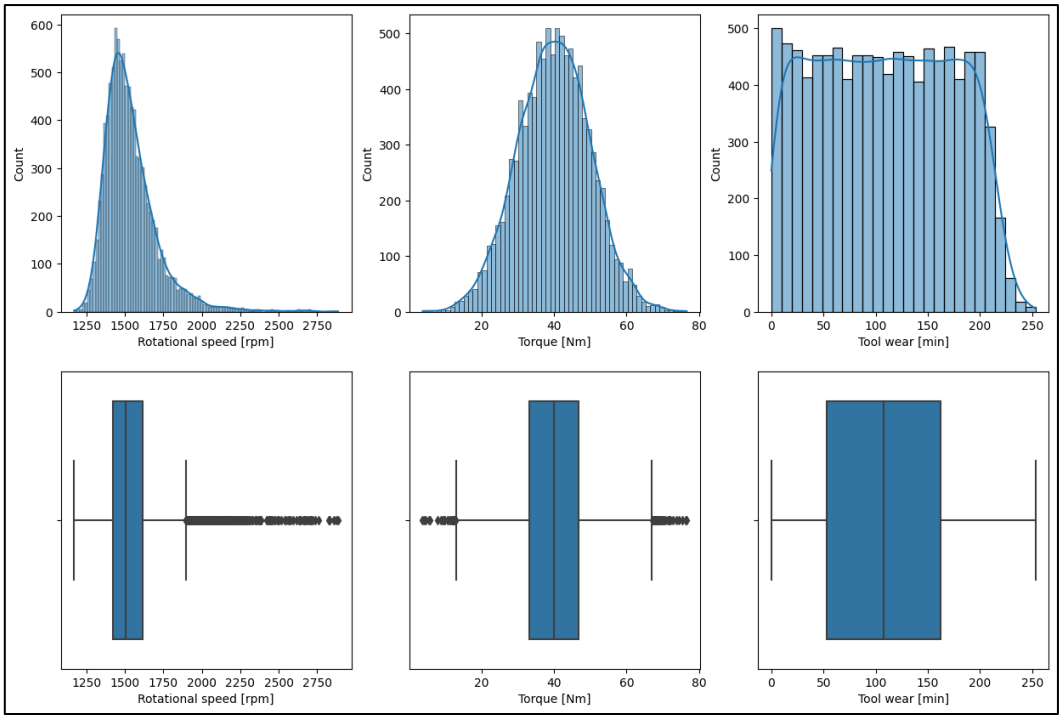


Figure 12: Histograms generated for each feature 2

3.3.2 Correlation

Correlation refers to the statistical association between two variables. It quantifies how changes in one variable relate to changes in another. Correlation values range between -1 and 1, with -1 indicating a strong negative correlation, 1 indicating a strong positive correlation, and 0 indicating no correlation. Understanding correlations in a data set is vital for several reasons. Firstly, it helps in feature selection by identifying highly correlated features and choosing a subset that provides unique and relevant information for model training. Secondly, correlation analysis reduces redundancy by eliminating highly correlated features, simplifying the model, and enhancing interpretability. Thirdly, it assists in checking the assumptions of linear models, ensuring the validity of underlying statistical assumptions. Lastly, correlation analysis aids in data exploration by providing insights into the relationships between variables, facilitating hypothesis generation.

Correlation coefficients, such as Pearson's correlation coefficient and Spearman's rank correlation coefficient, are commonly used to measure correlation. Pearson's correlation coefficient is suitable for linear relationships, while Spearman's correlation coefficient captures monotonic relationships. Interpreting correlation coefficients involves considering their magnitude and sign. The absolute value indicates the strength of the correlation, with values closer to 1 (positive or negative) representing stronger relationships. The sign (+/-) indicates the direction of the correlation, with positive values indicating a positive linear relationship and negative values indicating a negative linear relationship.

Pairplots are graphical representations that showcase the pairwise relationships between variables in a data set. They provide a comprehensive overview of correlations and can help identify patterns and dependencies visually.

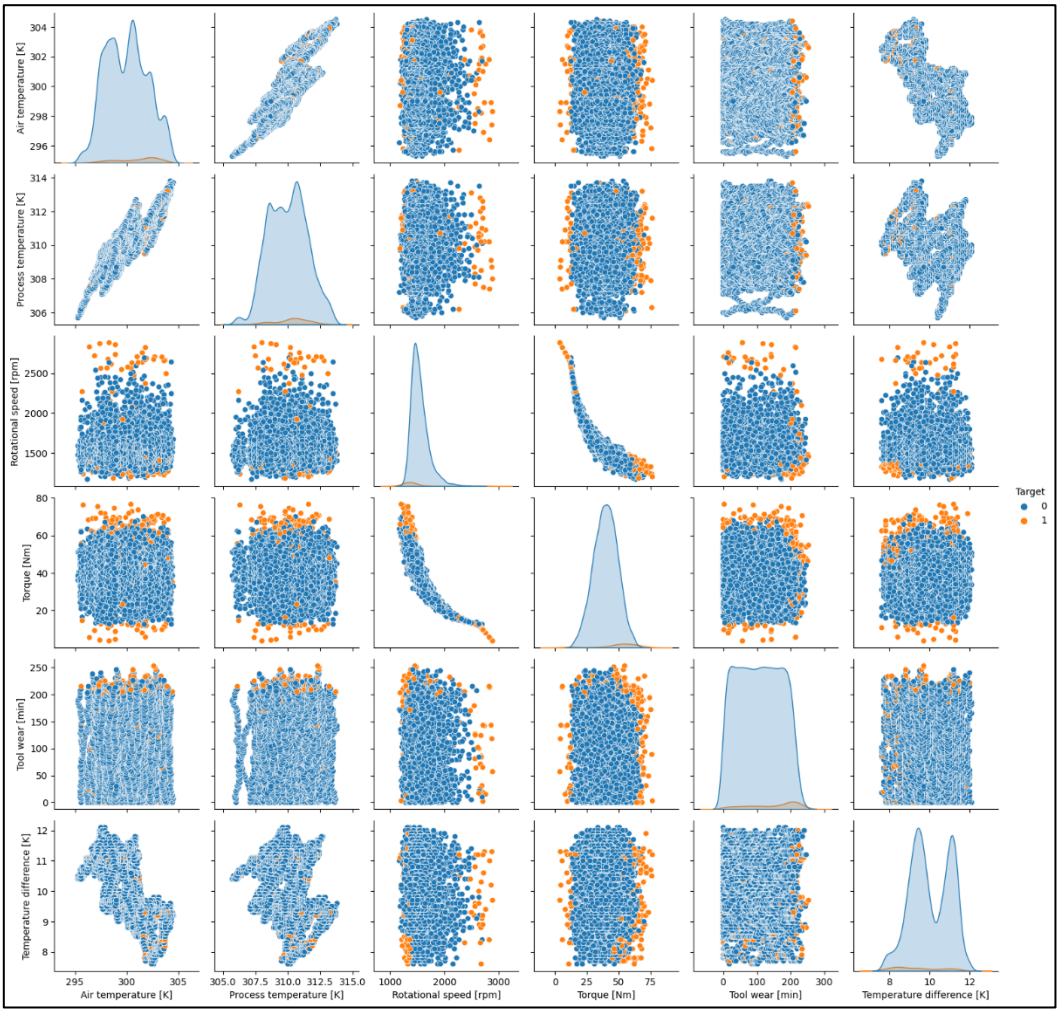


Figure 13: Pairplots generated for each feature

To identify correlated features, correlation matrices, or correlation coefficients are computed for the variables in a data set. High correlation coefficients (above a predefined threshold) suggest strong interdependencies.

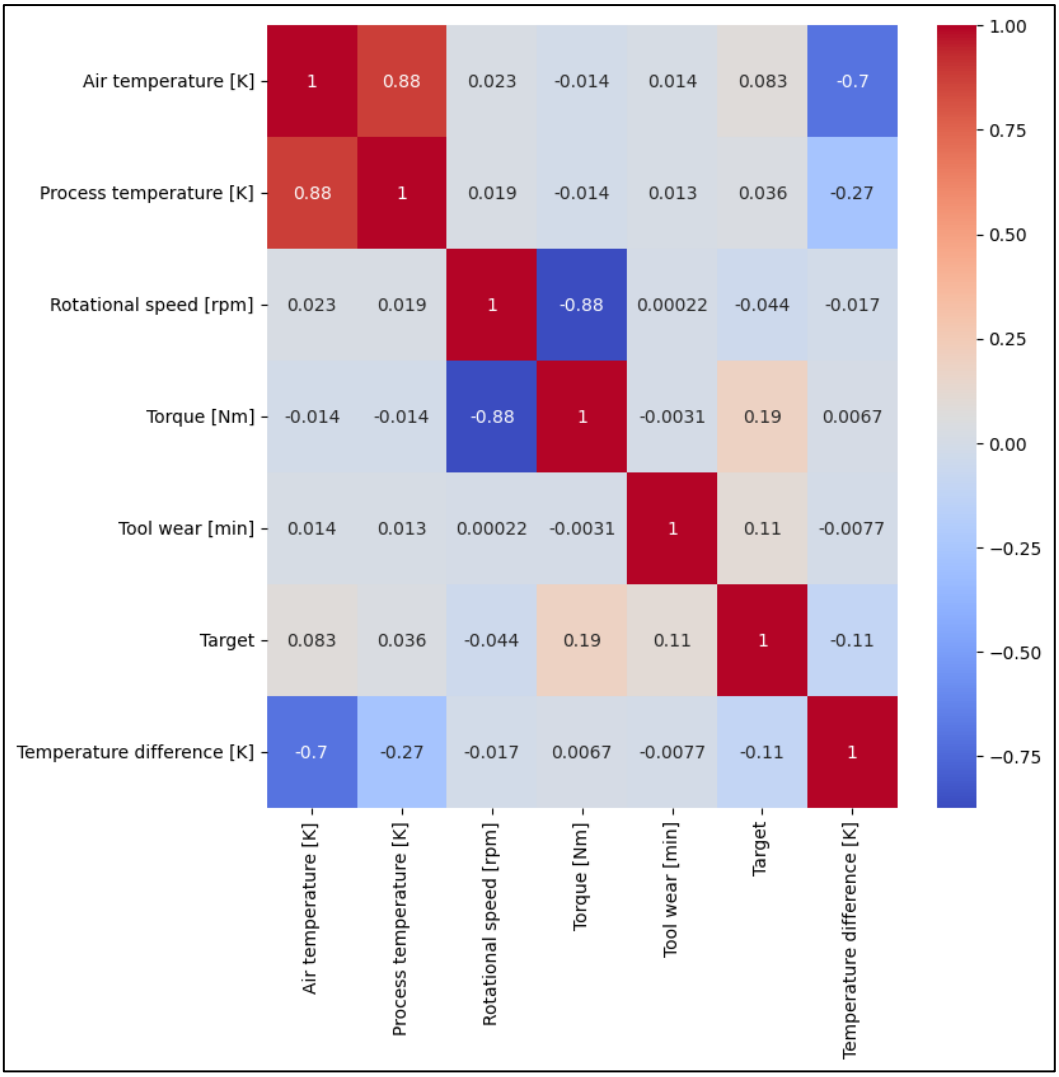


Figure 14: Heatmap generated for each feature

Dealing with correlated features can be approached in various ways. Highly correlated features can be removed to eliminate redundancy and enhance model performance. Alternatively, correlated features can be combined or transformed into a single representative feature, reducing dimensionality while maintaining relevant information. Regularization techniques such as L1 or L2 regularization can also be used to handle correlated features and prevent overfitting. While correlation analysis provides valuable insights, it is important to consider its practical implications. Correlation does not imply causation, so caution should be exercised when making causal claims based solely on correlation analysis. Non-linear relationships may exist, and exploring non-linear correlations using appropriate techniques is recommended. Outliers can influence correlation coefficients, so outlier detection and treatment should be performed beforehand.

3.3.3 Normalization

Normalization involves transforming numerical data to a common scale. By adjusting the values of features, normalization brings them within a specific range or distribution. The goal is to prevent certain features from dominating the learning process and ensure fair comparisons among them, particularly when features have varying scales or units. Normalization can accelerate the convergence of gradient-based optimization algorithms, enabling faster training of machine learning models. Additionally, it can improve the performance of models that are sensitive to feature scales, such as distance-based algorithms or regularization techniques.

There are several common normalization techniques used in machine learning. Min-Max Scaling is a technique that scales the features to a specific range, typically between 0 and 1, by subtracting the minimum value and dividing by the range. Z-Score Normalization, also known as standardization, transforms features to have a mean of 0 and a standard deviation of 1 by subtracting the mean and dividing by the standard deviation. When applying normalization to a data set, several practical considerations should be taken into account. Outliers in the data can have a significant impact on the effectiveness of normalization techniques. Therefore, it is important to handle outliers prior to normalization. Additionally, missing data should be addressed before normalization, as missing values can affect the calculation of normalization parameters.

The impact of normalization on model performance can vary depending on the algorithm and the characteristics of the data set. In general, normalization tends to improve the performance of models that rely on distance-based metrics or have sensitive feature scaling requirements. Algorithms such as k-nearest neighbors or support vector machines often benefit from normalization. However, for certain algorithms like tree-based models (e.g., decision trees, random forests), normalization may have little to no effect on performance since these models are not affected by feature scales.

3.3.4 Examination of data set

The filtered and normalized data set was manually inspected for categorical features, and the results were negative. To analyze the behavior of the parameters, a statistical overview was performed using programming. The data was displayed in order to provide a more accurate depiction of the data set, and there is a relationship between each parameter and correlation. Multivariate plots and histogram plots were employed in the visualization process.

3.4 Training of the Machine Learning Models

The training of the machine learning models involved several crucial steps to ensure the accuracy and reliability of the models. First, we meticulously checked for any missing values in the feature columns of the synthetic dataset. Fortunately, due to the thorough data generation process, we confirmed the absence of any missing values, ensuring the completeness of the dataset.

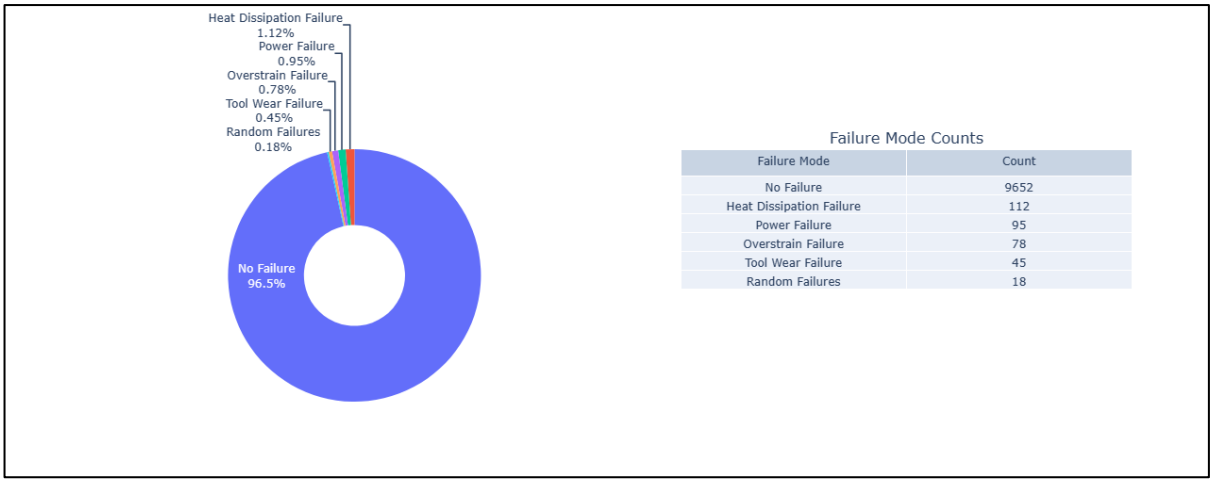
Next, we delved into understanding the distribution and ranges of predictor variables. By conducting a thorough analysis, we were able to identify the characteristics of the data and appropriately scale down the predictor variables. This step was essential in ensuring that the machine learning models could efficiently handle the data while retaining the meaningful information.

Additionally, we took into consideration the temperature difference between the process temperature and air temperature for finding correlations. This approach allowed us to uncover meaningful relationships between variables. Notably, we observed high correlations between process temperature and air temperature, as well as between rotational speed and torque. These insights further informed our feature selection process, ensuring that the models could capture the most relevant patterns and relationships in the data.

3.4.1 Data Analysis

During the analysis of the scatterplots, a crucial observation was made: a significant number of failures occurred when rotational speed or torque values were high. This finding indicated that these two factors play a critical role in influencing failure occurrences. This valuable insight guided our feature selection process and emphasized the need to incorporate rotational speed and torque as essential predictors in the machine learning models

Moreover, the scatterplots unveiled another crucial pattern related to product quality (type). It was evident that the type of product had a profound impact on the occurrence of failures. The clear distinction between failure occurrences for different product types emphasized the strong effect of product quality on the reliability of the milling machine. This finding led to the incorporation of product type as a significant feature in the machine learning models, recognizing its role in predicting failures accurately.



Failure Mode	Count
No Failure	9652
Heat Dissipation Failure	112
Power Failure	95
Overstrain Failure	78
Tool Wear Failure	45
Random Failures	18

Figure 15: Failure mode count in dataset

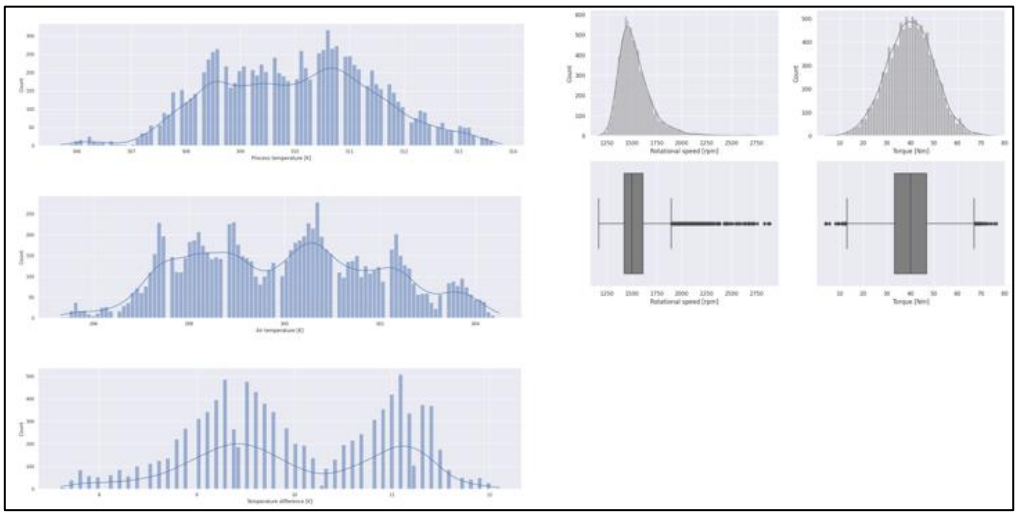


Figure 16: Temperature distribution, Rotational speed & torque distribution

By incorporating these insights and utilizing the carefully prepared dataset, our machine learning models were trained to capture the complex relationships between rotational speed, torque, product quality, and failure occurrences. These models are now well-equipped to predict potential failures and make informed decisions to ensure the smooth and reliable functioning of the milling machine in various operating conditions.

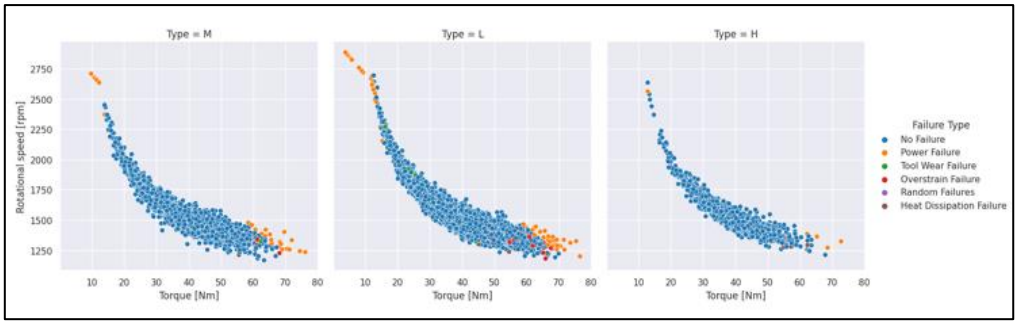


Figure 17: Failure mode variation with respect to Product quality, Torque & rotational speed

3.4.2 Machine Learning Models

In the development of the machine learning system, a diverse set of powerful and widely used models to tackle the predictive tasks effectively was employed. Among the models utilized were Support Vector Machine, Random Forest, Decision Tree, CatBoost, XGBoost, K-Nearest Neighbor, Gradient Boosting, and Deep Neural Network.

By combining the strengths of diverse machine learning models, a comprehensive and robust predictive system capable of handling a wide range of data types and complexity levels, ensuring accurate predictions and valuable insights for various applications, was created.

3.5 Building of a Neural Network

Despite the fact that neural networks are well known for their application to classification difficulties. Examples include optical character recognition, speech recognition, and face recognition. In the project, we used neural networks to solve a regression problem involving cost prediction.

In the first stage, we trained a simple neural network to assess the neural network's capacity to solve regression problems. The initial neural network has two hidden layers with 64 nodes each.

3.6 Hyperparameter Tuning

Hyperparameter tuning, also known as hyperparameter optimization or model selection, is a critical step in the machine learning workflow that involves finding the best set of hyperparameters for a given machine learning model. Hyperparameters are parameters that cannot be learned from the data during the training process and need to be set before training the model.

In machine learning algorithms, hyperparameters determine the behavior and performance of the model. They include variables like learning rate, number of hidden layers and nodes in a neural network, regularization strength, and kernel type in Support Vector Machines. Selecting appropriate hyperparameters is essential to achieve optimal model performance and prevent issues like overfitting or underfitting.

Hyperparameter tuning aims to find the combination of hyperparameters that results in the best model performance on a validation set or cross-validation process. This process typically involves trying out various hyperparameter combinations, training the model with each combination, and evaluating its performance on the validation data. Techniques like grid search, random search, and Bayesian optimization are commonly used to efficiently explore the hyperparameter space and find the best configuration for the model. Hyperparameter tuning is a crucial aspect of building accurate and robust machine learning models, as it significantly impacts the model's ability to generalize well to new, unseen data.

Feature	Model parameters	Hyperparameters
Definition	The values that are learned by the model during training.	The values that are set before training begins.
Purpose	Used to make predictions on new data.	Control the learning process of the model.
How they are set	Learned from the data.	Set by the user.
Examples	Coefficients in a linear regression model.	Number of features to use, learning rate, number of epochs.

Figure 18: Feature summery

3.6.1 Common Hyperparameter

Hyperparameter tuning is a critical process in machine learning that involves fine-tuning various hyperparameters to optimize the performance of a model. These hyperparameters are settings that cannot be learned from the data during training and directly impact the model's behavior and performance. Some commonly tuned hyperparameters include the learning rate, which determines the step size during gradient descent, and the number of hidden units and layers in a neural network, affecting the network's complexity and capacity to capture intricate patterns.

For ensemble methods like Random Forest, hyperparameters such as the number of trees and the maximum tree depth play a crucial role in balancing model complexity and overfitting. Clustering algorithms, on the other hand, rely on the number of clusters as a key hyperparameter, influencing the granularity of data grouping. Additionally, activation functions are essential hyperparameters for neural networks, as they govern the non-linearity and information flow between network layers.

Hyperparameter tuning is typically performed using techniques like grid search or random search, exploring various combinations of hyperparameter values and evaluating the model's performance on a validation set. By finding the optimal hyperparameter configuration, the model can achieve better generalization and higher accuracy, enhancing its effectiveness in real-world applications. Careful and systematic hyperparameter tuning is an integral part of

model optimization, leading to the development of accurate and robust machine learning systems.

3.6.2 Hyperparameter Tuning Techniques

Hyperparameter tuning is a crucial process in machine learning that aims to identify the optimal combination of hyperparameters for a given model. Hyperparameters are parameters that cannot be learned during model training and significantly impact the model's performance. Several techniques are employed for hyperparameter tuning, each with its unique approach.

Random exploring involves randomly sampling hyperparameter values within predefined ranges to explore different configurations and assess their impact on model performance. GridSearchCV systematically evaluates all possible combinations of hyperparameters specified in a grid, making it a comprehensive but computationally expensive method.

RandomizedSearchCV combines random exploration with the advantages of grid search, efficiently sampling a subset of hyperparameter combinations to find promising configurations. Bayesian optimization uses probabilistic models to guide the search process, efficiently balancing exploration and exploitation to identify optimal hyperparameters.

Hyperparameter tuning frameworks like optuna provide efficient and automated optimization processes, leveraging techniques like tree structured Parzen estimators and successive halving to efficiently explore the hyperparameter space.

Hyperparameter tuning is an essential step in model optimization, as selecting the right hyperparameters enhances a model's ability to generalize well and achieve peak performance in real-world scenarios. By employing these tuning techniques, data scientists can ensure the development of accurate and robust machine learning models, tailored to the unique characteristics of the data and the desired task at hand.

Model	Model Accuracy Score	After Hyperparameter Tuning
Support Vector Machine	96.45%	97.60%
Random Forest	98.35%	98.25%
Decision Tree	96.90%	97.85%
CatBoost	98.35%	98.35%
XgBoost	98.40%	98.25%
KNN	96.85%	96.83%
Gradient Boosting	98.00%	98.10%
Logistic Regression	96.80%	97.40%
Deep Neural Network	96.75%	96.75%

Figure 19: Model accuracy after Hyperparameter tuning

3.6.3 Machine Learning Model Evaluation

For a data set with multiple classes (multi-class classification), the choice of the best performance metrics depends on various factors such as class distribution, class imbalance, and the specific objectives of the machine learning model. Following are the performance metrics used for our machine learning models evaluation process.

1. Accuracy: Accuracy measures the overall proportion of correctly classified instances across all classes. It is a straightforward and commonly used metric, but it may not be suitable for imbalanced datasets.

2. Precision (Per Class): Precision measures the proportion of true positive predictions for a specific class among all instances predicted as that class. It is useful when the cost of false positives is high.

3. Recall (Per Class): Recall (also known as sensitivity) measures the proportion of true positive predictions for a specific class among all instances that belong to that class. It is useful when the cost of false negatives is high.

4. F1-score (Per Class): The F1-score is the harmonic mean of precision and recall for a specific class. It provides a balanced measure that considers both false positives and false negatives.

5. Macro-Averaged Precision, Recall, and F1-score: Macro-averaging calculates the precision, recall, and F1-score for each class individually and then takes their average. It treats all classes equally, which is suitable when each class is of equal importance.

6. Micro-Averaged Precision, Recall, and F1-score: Micro-averaging calculates the overall precision, recall, and F1-score by considering the total number of true positives, false positives, and false negatives across all classes. It gives more weight to larger classes.

7. Weighted Precision, Recall, and F1-score: Weighted averaging is similar to macro-averaging but takes into account class imbalance by weighting each class's metric by its proportion in the dataset.

	Cross Validated				Hyper Parameter Tuning	
	Accuracy	Recall	Precision	F1-Score	Total time taken for Hyperparameter tuning / (seconds)	Total Time Taken for Training and Prediction of best parameter / (Seconds)
Support Vector Machine	99.55%	99.55%	99.27%	99.40%	571.9	9.89
Random Forest	98.20%	98.20%	97.35%	97.73%	113.85	0.91
Decision Tree	98.00%	98.00%	97.11%	97.52%	3.3	0.03
CatBoost	98.55%	98.55%	97.71%	98.11%	425.3042	3.17
XgBoost	_	_	_	_	853.2675	0.95
KNN	96.95%	96.95%	95.40%	95.86%	2.5094	0.0125
Gradient Boosting	98.10%	98.10%	97.34%	97.71%	1585.6182	15.65
Logistic Regression	97.50%	97.50%	95.80%	96.57%	91.33	1.46
Deep Neural Network	96.75%	96.75%	95.43%	95.75%	90.25	0.90

Figure 20: Model evaluation after cross validation & Time for Hyperparameter tuning

3.6.4 Comparison with existing models

	Accuracy	Comparison with existing models
Support Vector Machine	99.55%	96.05%
Random Forest	98.20%	99.55%
Decision Tree	98.00%	99.30%
CatBoost	98.55%	No models
XgBoost	_	_
KNN	96.95%	98.56%
Gradient Boosting	98.10%	99.96%
Logistic Regression	97.50%	97.80%
Deep Neural Network	96.75%	No models

Figure 21: Comparison between existing models

Chapter 4: Conclusions

The project's purpose was to assess the feasibility of applying artificial intelligence technologies to real-world mechanical engineering applications. Based on the results of this experiment, it is clear that artificial intelligence approaches are capable of dealing with such unpredictable and complicated situations and anticipating the behavior of such systems.

We may deduce from the project that data preprocessing is an important aspect of the machine learning workflow. Using data preparation techniques such as parameter reduction, data filtration, and data standardization can have a significant impact on the modals. Data examination and visualization also aid in comprehending the behavior of each parameter as well as the relationship between parameters.

Techniques from machine learning can be utilized in the cost prediction process, and error and performance metrics can also be used to understand how effectively the trained model performs and thus help to enhance the trained model by optimizing various parameters that affect prediction quality.

Although neural networks can be employed in regression problems, they are used in this project for classification issues. With the right methodologies and parameters, it is clear that applying neural networks to the task at hand produces superior outcomes than machine learning tools.

The use of the neural network revealed that the use of optimization techniques such as hyperparameter tweaking results in the best neural network architecture. The use of such a streamlined architecture significantly improves prediction quality.

References

1. Weeratunge, H., Aditya, G., Dunstall, S., de Hoog, J., Narsilio, G., Halgamuge, S. (2021). Feasibility and performance analysis of hybrid ground source heat pump systems in fourteen cities. Energy, 234, 121254.
2. https://doi.org/10.1016/j.energy.2021.121254

3. Goodfellow, I., Bengio, Y., Courville, A. Deep learning.

4. Brunton, S., Kutz, J. (2019). Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control.
5. Cambridge: Cambridge University Press. doi:10.1017/9781108380690

6. 4. Ge`Iron, A. (2019). Hands-on machine learning with Scikit-Learn, Keras and TensorFlow: concepts, tools, and techniques to build intelligent systems (2nd ed.). O'Reilly.

6. Francois Chollet. 2017. Deep Learning with Python (1st. ed.). Manning Publications Co., USA.

7. Brownlee, J. (2021). How to Grid Search Hyperparameters for Deep Learning Models in Python With Keras. Machine Learning Mastery. Retrieved 20 December 2021, from https://machinelearningmastery.com/grid-searchhyperparameters-deep-learning-models-python-keras/.

8. Jordan, J. (2021). Hyperparameter tuning for machine learning models.. Jeremy Jordan. Retrieved 20 December 2021, from https://www.jeremyjordan.me/hyperparameter-tuning/.

9. Supervised learning: predicting an output variable from high-dimensional observations. scikit-learn. (2021). Retrieved 20 December 2021, from https://scikit-learn.org/stable/tutorial/statisticalinference/supervisedlearning.html

10. Deep Neural Networks for Regression Problems. Medium. (2021). Retrieved 20 December 2021, from https://towardsdatascience.com/deep-neuralnetworks-for-regression-problems-81321897ca33.

11. SMITH, A., MASON, A. (1997). COST ESTIMATION PREDICTIVE MODELING: REGRESSION VERSUS NEURAL NETWORK. The Engineering Economist, 42(2), 137-161.
7. https://doi.org/10.1080/00137919708903174

12. IBM Machine Learning Professional Certificate | Coursera

13. A Comprehensive Guide on Hyperparameter Tuning and its Techniques
8. A Comprehensive Guide on Hyperparameter Tuning and its Techniques (analyticsvidhya.com)

14. Hyperparameters in Machine Learning - Javatpoint

Google Collaboratory link of the Project:

https://colab.research.google.com/drive/1zzvaDgelW2BnnIXL2k7PSDdrsNBQlwIn?usp=sharing

Appendix

Project_Test_7_1_2.ipynb - Colaboratory.pdf

Google Collaboratory link of the Project:

https://colab.research.google.com/drive/1zzvaDgelW2BnnIXL2k7PSDdrsNBQlwIn?usp=sharing

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import RandomOverSampler
import seaborn as sns

raw_data = pd.read_csv("/content/predictive_maintenance.csv")
df = raw_data

import ipywidgets as widgets
from IPython.display import display

# Function to update the interactive table based on the selected columns
def update_table(selected_columns):
    display(df[selected_columns])

# Get the list of columns from the DataFrame
all_columns = df.columns.tolist()

# Display the initial table with all columns
update_table(all_columns)
```

	UDI	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Target	Failure Type
0	1	M14860	M	298.1	308.6	1551	42.8	0	0	No Failure
1	2	L47181	L	298.2	308.7	1408	46.3	3	0	No Failure
2	3	L47182	L	298.1	308.5	1498	49.4	5	0	No Failure
3	4	L47183	L	298.2	308.6	1433	39.5	7	0	No Failure
4	5	L47184	L	298.2	308.7	1408	40.0	9	0	No Failure
...	...	...	...	...	...	...	...	...	...	...
9995	9996	M24855	M	298.8	308.4	1604	29.5	14	0	No Failure
9996	9997	H39410	H	298.9	308.4	1632	31.8	17	0	No Failure
9997	9998	M24857	M	299.0	308.6	1645	33.4	22	0	No Failure
9998	9999	H39412	H	299.0	308.7	1408	48.5	25	0	No Failure
9999	10000	M24859	M	299.0	308.7	1500	40.2	30	0	No Failure

```
df.info()  # Displaying information about the DataFrame
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 10 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   UDI                      10000 non-null  int64  
 1   Product ID               10000 non-null  object 
 2   Type                     10000 non-null  object 
 3   Air temperature [K]      10000 non-null  float64
 4   Process temperature [K]  10000 non-null  float64
 5   Rotational speed [rpm]   10000 non-null  int64  
 6   Torque [Nm]              10000 non-null  float64
 7   Tool wear [min]          10000 non-null  int64  
 8   Target                   10000 non-null  int64  
 9   Failure Type             10000 non-null  object 
dtypes: float64(3), int64(4), object(3)
memory usage: 781.4+ KB
```

Difference between Process temperature and air temperature might be a indication of abnormal behaviour of CNC machine. Therefore a new column "Temperature difference [K]" was added.

```
df["Temperature difference [K]"] = df["Process temperature [K]"] - df["Air temperature [K]"]
df.sample(4)
```

	UDI	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Target	Failure Type	Temperature difference [K]
7660	7661	M22520	M	300.5	311.6	1246	52.9	165	0	No Failure	11.1
9341	9342	M24201	M	298.3	308.9	1363	46.1	5	0	No Failure	10.6
6949	6950	M21809	M	300.7	311.2	1769	26.5	63	0	No Failure	10.5

```
df.describe().style.background_gradient(cmap="turbo")   # Generating a styled summary of the dataset
```

	UDI	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Target	Temperature difference [K]
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000
mean	5000.500000	300.004930	310.005560	1538.776100	39.986910	107.951000	0.033900	10.000630
std	2886.895680	2.000259	1.483734	179.284096	9.968934	63.654147	0.180981	1.001094
min	1.000000	295.300000	305.700000	1168.000000	3.800000	0.000000	0.000000	7.600000
25%	2500.750000	298.300000	308.800000	1423.000000	33.200000	53.000000	0.000000	9.300000
50%	5000.500000	300.100000	310.100000	1503.000000	40.100000	108.000000	0.000000	9.800000
75%	7500.250000	301.500000	311.100000	1612.000000	46.800000	162.000000	0.000000	11.000000

Check for missing values

```
missing_data = pd.DataFrame({'total_missing': df.isnull().sum(), 'perc_missing': (df.isnull().sum()/82790)*100})
missing_data
```

	total_missing	perc_missing
UDI	0	0.0
Product ID	0	0.0
Type	0	0.0
Air temperature [K]	0	0.0
Process temperature [K]	0	0.0
Rotational speed [rpm]	0	0.0
Torque [Nm]	0	0.0
Tool wear [min]	0	0.0
Target	0	0.0
Failure Type	0	0.0
Temperature difference [K]	0	0.0

Since this is a synthetic dataset there are no missing values.

Dropping 'UDI', 'Product ID'

UDI and Product ID not informative for our ML algoriths. Therefore those two columns were removed.

```
df = raw_data.drop(["UDI","Product ID"],axis=1)
all_columns = df.columns.tolist()
update_table(all_columns)
```

	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Target	Failure Type	Temperature difference [K]
0	M	298.1	308.6	1551	42.8	0	0	No Failure	10.5
1	L	298.2	308.7	1408	46.3	3	0	No Failure	10.5
2	L	298.1	308.5	1498	49.4	5	0	No Failure	10.4
3	L	298.2	308.6	1433	39.5	7	0	No Failure	10.4
4	L	298.2	308.7	1408	40.0	9	0	No Failure	10.5
...	...	...	...	...	...	...	...	...	...

```
pip install pandas_profiling
```

```
Collecting htmlmin==0.1.12 (from ydata-profiling->pandas_profiling)
  Downloading htmlmin-0.1.12.tar.gz (19 kB)
  Preparing metadata (setup.py) ... done
Collecting phik<0.13,>=0.11.1 (from ydata-profiling->pandas_profiling)
  Downloading phik-0.12.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (679 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 679.5/679.5 kB 39.8 MB/s eta 0:00:00
Requirement already satisfied: requests<3,>=2.24.0 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling
Requirement already satisfied: tqdm<5,>=4.48.2 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling) (4
Requirement already satisfied: seaborn<0.13,>=0.10.1 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profili
Collecting multimethod<2,>=1.4 (from ydata-profiling->pandas_profiling)
  Downloading multimethod-1.9.1-py3-none-any.whl (10 kB)
Requirement already satisfied: statsmodels<1,>=0.13.2 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profil
Collecting typeguard<3,>=2.13.2 (from ydata-profiling->pandas_profiling)
  Downloading typeguard-2.13.3-py3-none-any.whl (17 kB)
Collecting imagehash==4.3.1 (from ydata-profiling->pandas_profiling)
  Downloading ImageHash-4.3.1-py2.py3-none-any.whl (296 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 296.5/296.5 kB 26.4 MB/s eta 0:00:00
Collecting wordcloud>=1.9.1 (from ydata-profiling->pandas_profiling)
  Downloading wordcloud-1.9.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (455 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 455.4/455.4 kB 39.6 MB/s eta 0:00:00
Collecting dacite>=1.8 (from ydata-profiling->pandas_profiling)
  Downloading dacite-1.8.1-py3-none-any.whl (14 kB)
Requirement already satisfied: PyWavelets in /usr/local/lib/python3.10/dist-packages (from imagehash==4.3.1->ydata-profiling->pandas_
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages (from imagehash==4.3.1->ydata-profiling->pandas_prof
Requirement already satisfied: attrs>=19.3.0 in /usr/local/lib/python3.10/dist-packages (from visions[type_image_path]==0.7.5->ydata-
Requirement already satisfied: networkx>=2.4 in /usr/local/lib/python3.10/dist-packages (from visions[type_image_path]==0.7.5->ydata-
Collecting tangled-up-in-unicode>=0.0.4 (from visions[type_image_path]==0.7.5->ydata-profiling->pandas_profiling)
  Downloading tangled_up_in_unicode-0.2.0-py3-none-any.whl (4.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.7/4.7 MB 37.4 MB/s eta 0:00:00
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2<3.2,>=2.11.1->ydata-profiling-
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib<4,>=3.2->ydata-profiling-
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib<4,>=3.2->ydata-profiling->pan
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib<4,>=3.2->ydata-profiling
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib<4,>=3.2->ydata-profiling
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib<4,>=3.2->ydata-profiling->
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib<4,>=3.2->ydata-profiling-
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib<4,>=3.2->ydata-profil
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas!=1.4.0,<2.1,>1.1->ydata-profiling
Requirement already satisfied: joblib>=0.14.1 in /usr/local/lib/python3.10/dist-packages (from phik<0.13,>=0.11.1->ydata-profiling->p
Requirement already satisfied: typing-extensions>=4.2.0 in /usr/local/lib/python3.10/dist-packages (from pydantic<2,>=1.8.1->ydata-pr
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.24.0->ydata-prof
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.24.0->ydata-profili
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.24.0->ydata-
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.24.0->ydata-profiling->pa
Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.10/dist-packages (from statsmodels<1,>=0.13.2->ydata-profiling-
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.2->statsmodels<1,>=0.13.2->ydata-profi
Building wheels for collected packages: htmlmin
  Building wheel for htmlmin (setup.py) ... done
  Created wheel for htmlmin: filename=htmlmin-0.1.12-py3-none-any.whl size=27079 sha256=b3b114c19f100ff96238d9ed81ead4ad98ced5452b662
  Stored in directory: /root/.cache/pip/wheels/dd/91/29/a79cecb328d01739e64017b6fb9a1ab9d8cb1853098ec5966d
Successfully built htmlmin
Installing collected packages: htmlmin, typeguard, tangled-up-in-unicode, multimethod, dacite, imagehash, wordcloud, visions, phik, y
  Attempting uninstall: wordcloud
    Found existing installation: wordcloud 1.8.2.2
    Uninstalling wordcloud-1.8.2.2:
      Successfully uninstalled wordcloud-1.8.2.2
Successfully installed dacite-1.8.1 htmlmin-0.1.12 imagehash-4.3.1 multimethod-1.9.1 pandas_profiling-3.6.6 phik-0.12.3 tangled-up-in
```

```
import pandas_profiling

# Generate a Profile Report for the DataFrame
profile = pandas_profiling.ProfileReport(df)
profile
```

```
<ipython-input-15-431ccb7993e7>:1: DeprecationWarning: `import pandas_profiling` is going to be deprecated by April 1st. Please use `imp
  import pandas_profiling
```

Summarize dataset: 100% 54/54 [00:26<00:00, 1.63it/s, Completed]

Generate report structure: 100% 1/1 [00:15<00:00, 15.91s/it]

Render HTML: 100% 1/1 [00:03<00:00, 3.75s/it]

Overview

Dataset statistics

Number of variables	9
Number of observations	10000
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	703.2 KiB
Average record size in memory	72.0 B

Variable types

Categorical	3
Numeric	6

Alerts

`Air temperature [K]` is highly overall correlated with `Process temperature [K]` and 1 other fields (Process temperature [K], Temperature difference [K])	High correlation
`Process temperature [K]` is highly overall correlated with `Air temperature [K]`	High correlation
`Rotational speed [rpm]` is highly overall correlated with `Torque [Nm]`	High correlation
`Torque [Nm]` is highly overall correlated with `Rotational speed [rpm]`	High correlation
`Temperature difference [K]` is highly overall correlated with `Air temperature [K]`	High correlation
`Target` is highly overall correlated with `Failure Type`	High correlation

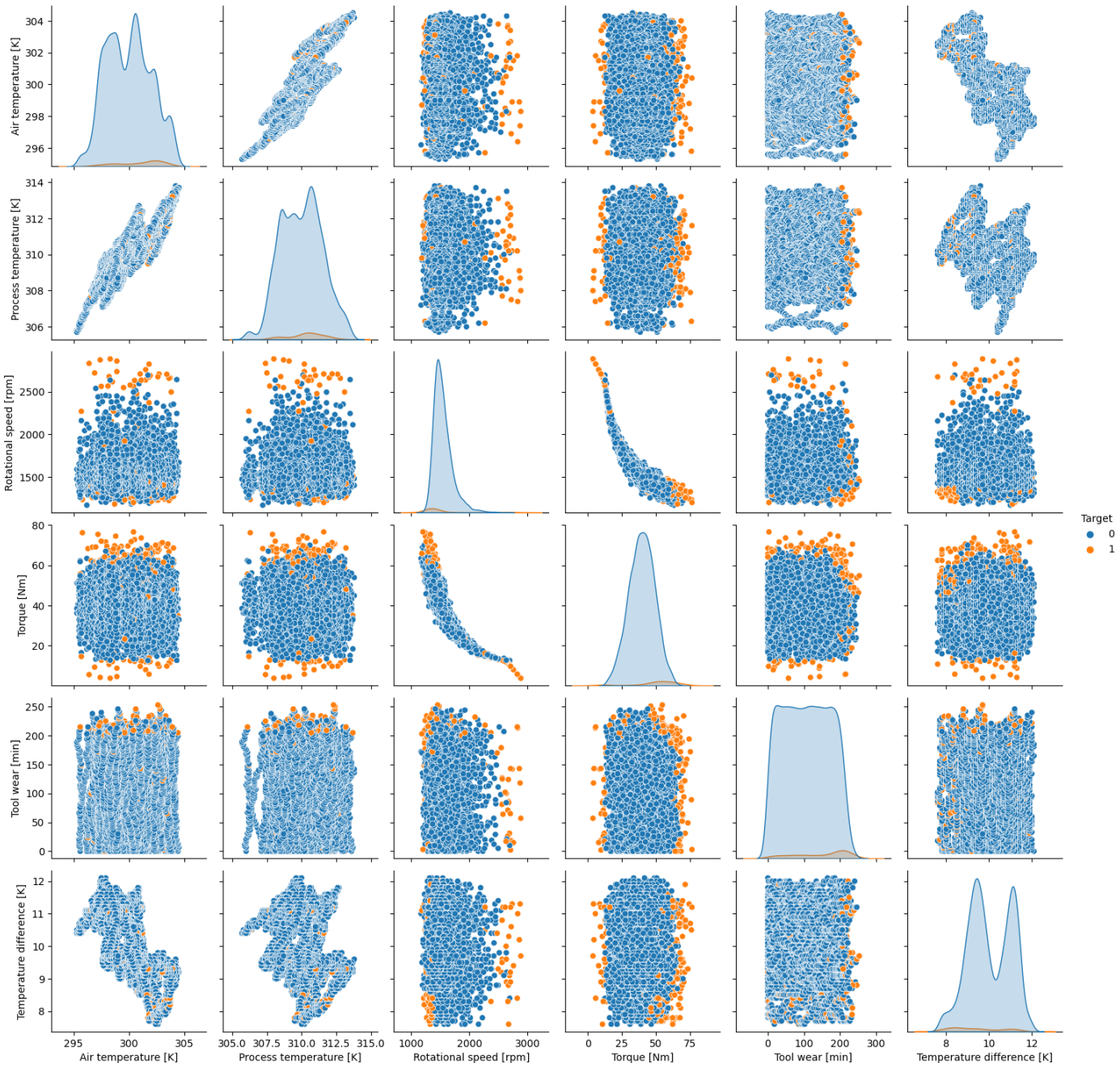
```
# checking distribution and ranges of predictor variables
plt.figure(figsize=(20,7))
sns.boxplot(data = df)
```

<Axes: >

The above diagram shows that all the predictors are not in the same range.So, we should normalize the data.

Correlations

```
sns.pairplot(df, hue = 'Target')
plt.show()
```



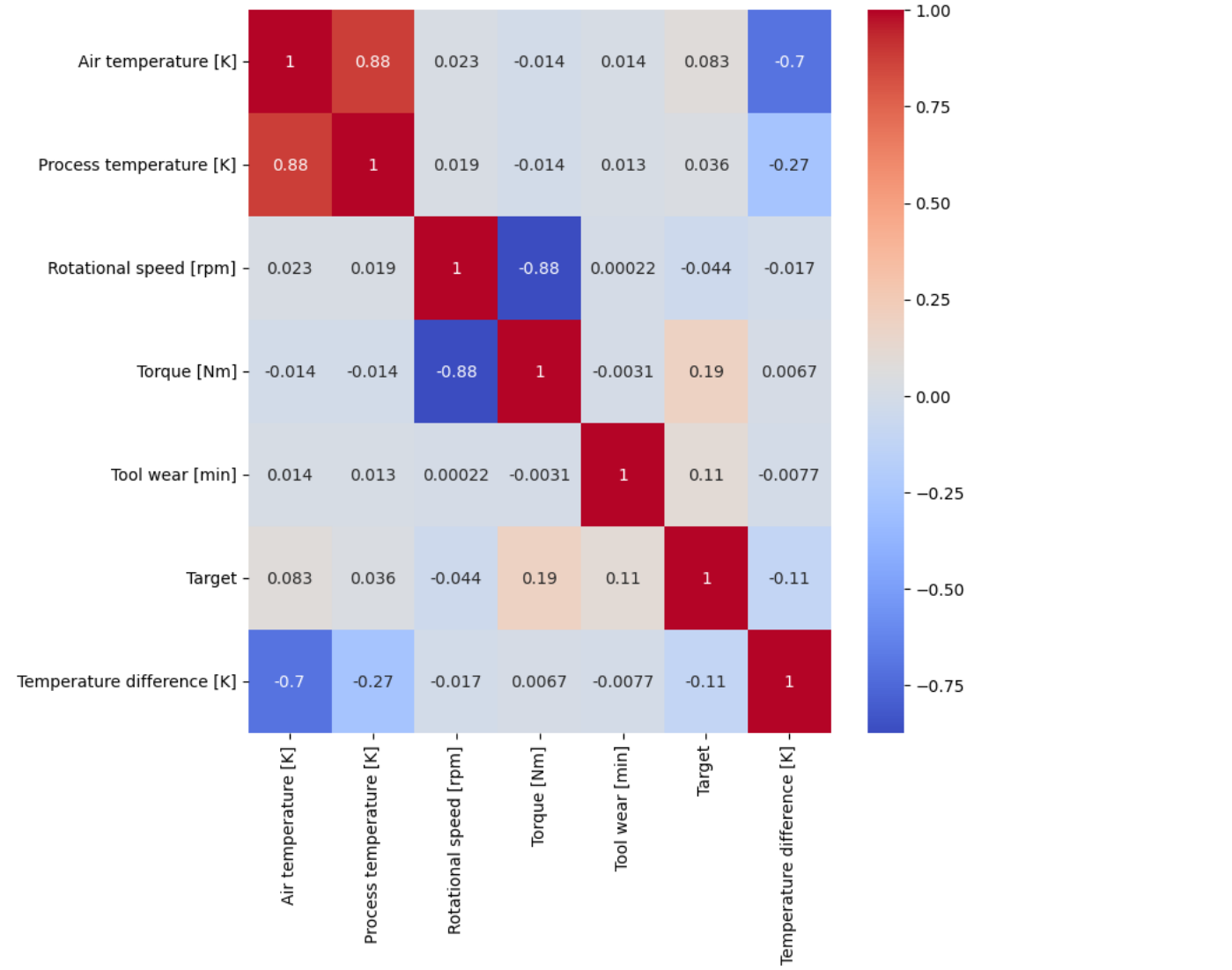
We can extract usefull information using these graph

- Torque vs Rotational speed graph - High torque or High rotational speed tends to CNC machine to fail
- High tool wear tennds machine to fail

Heatmap

```
plt.figure(figsize=(8, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.show()
```

```
<ipython-input-29-182bbc67185e>:2: FutureWarning:

The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid colu
```

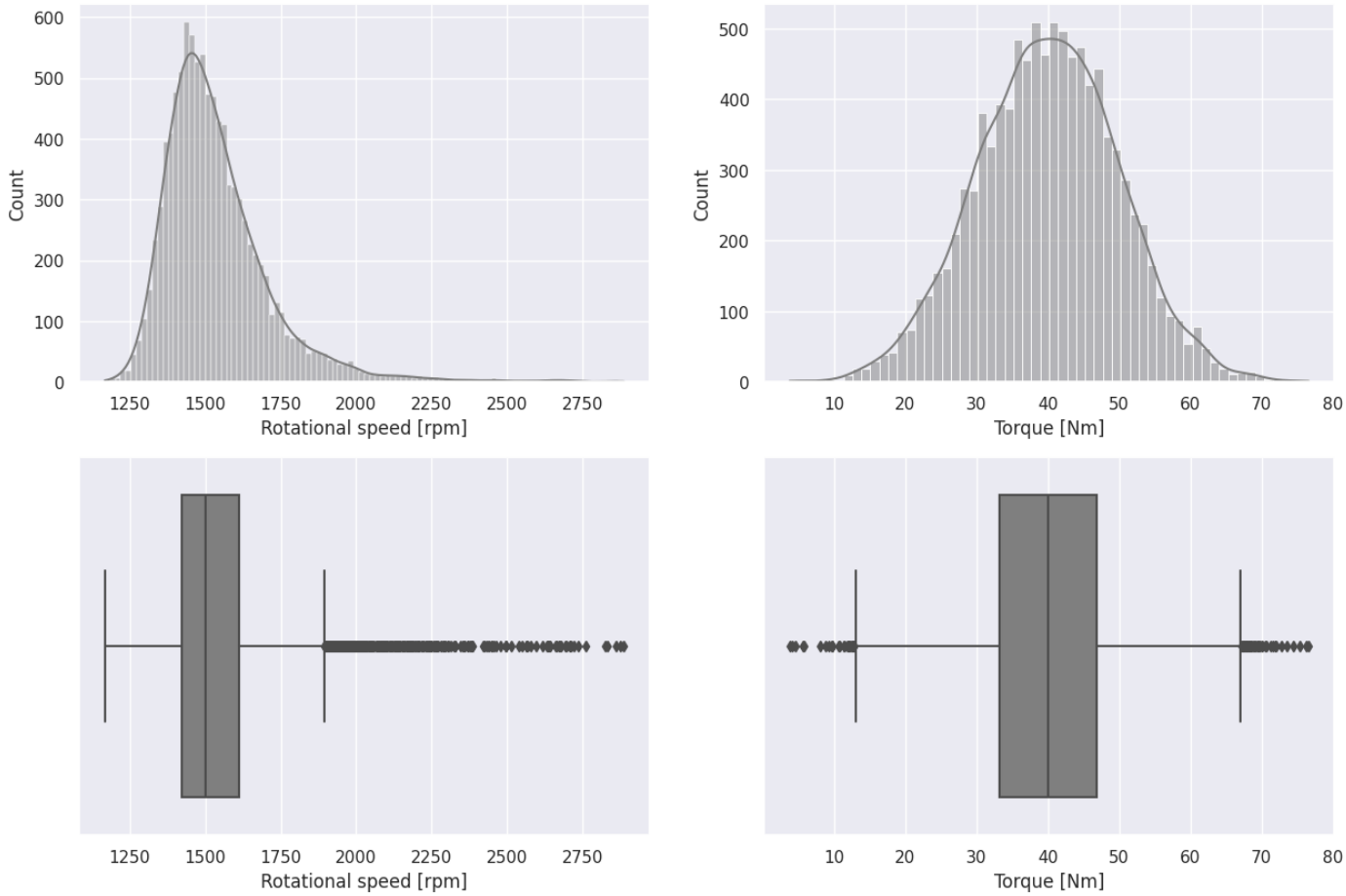


There is high correlation between process temperature and air temperature, and between rotational speed and torque.

Data Analysis

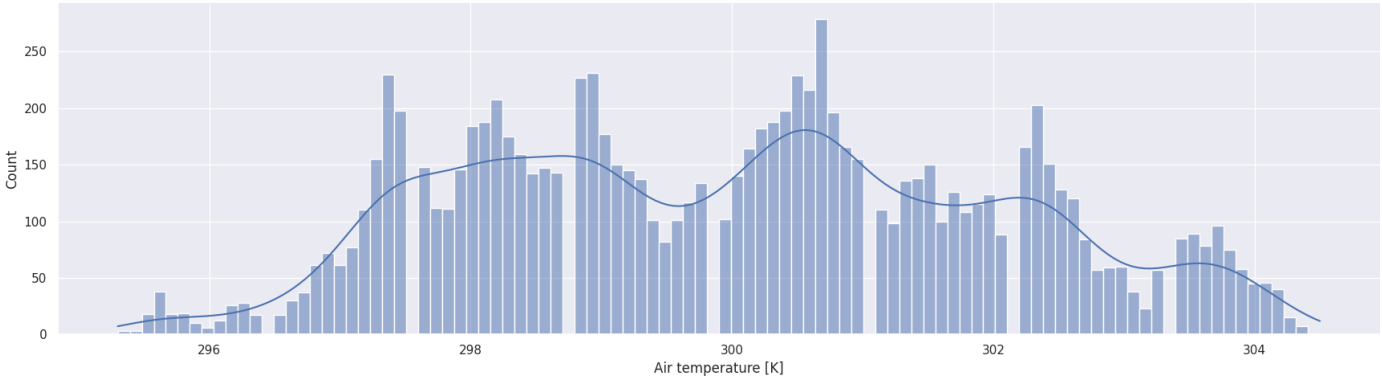
```
sns.set_theme(style="darkgrid")
fig1, ax1 = plt.subplots(2, 2, figsize=(15,10))
sns.histplot(data=df, x='Rotational speed [rpm]', kde=True, ax=ax1[0,0],color='#7f7f7f')
sns.histplot(data=df, x='Torque [Nm]', kde=True, ax=ax1[0,1],color='#7f7f7f')
```

```
sns.boxplot(data=df, x='Rotational speed [rpm]', ax=ax1[1,0],color='#7f7f7f')
sns.boxplot(data=df, x='Torque [Nm]', ax=ax1[1,1],color='#7f7f7f')
plt.show()
```



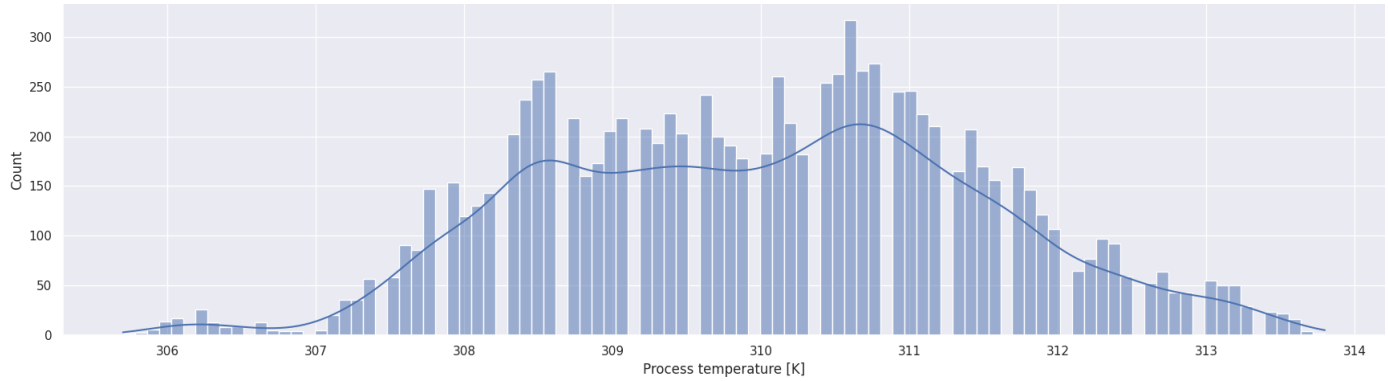

```
sns.displot(data=df, x="Air temperature [K]", kde=True, bins=100, palette="magma", height=5, aspect=3.5);
```

```
<ipython-input-9-bf9973e93372>:1: UserWarning: Ignoring `palette` because no `hue` variable has been assigned.
  sns.displot(data=df, x="Air temperature [K]", kde=True, bins=100, palette="magma", height=5, aspect=3.5);
```



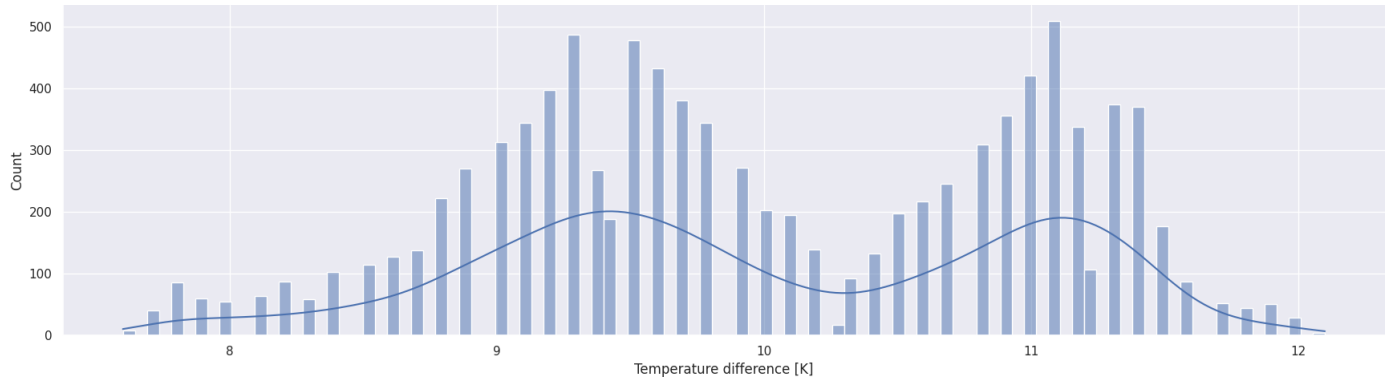

```
sns.displot(data=df, x="Process temperature [K]", kde=True, bins=100, palette="magma", height=5, aspect=3.5);
```

```
<ipython-input-10-9058a109c6a1>:1: UserWarning: Ignoring `palette` because no `hue` variable has been assigned.
  sns.displot(data=df, x="Process temperature [K]", kde=True, bins=100, palette="magma", height=5, aspect=3.5);
```



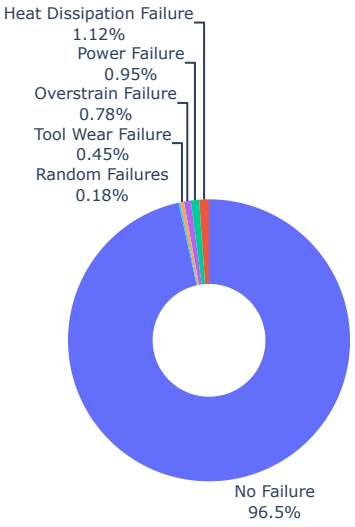

```
sns.displot(data=df, x="Temperature difference [K]", kde=True, bins=100, palette="magma", height=5, aspect=3.5);
```

```
<ipython-input-11-2145a0b0ce6d>:1: UserWarning: Ignoring `palette` because no `hue` variable has been assigned.
  sns.displot(data=df, x="Temperature difference [K]", kde=True, bins=100, palette="magma", height=5, aspect=3.5);
```




```
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Assuming 'Failure Type' is a column in the DataFrame 'df'
failure_counts = df['Failure Type'].value_counts()

# Donut chart
fig_donut = go.Figure(go.Pie(
    labels=failure_counts.index,
    values=failure_counts.values,
    hole=0.4,
    textinfo='label+percent',
    insidetextorientation='radial'
))

fig_donut.update_layout(title='Failure Mode Distribution', showlegend=False)

# Table
fig_table = go.Figure(data=[go.Table(
    header=dict(values=["Failure Mode", "Count"]),
    cells=dict(values=[failure_counts.index, failure_counts.values])
)])

# Create subplots with shared y-axis
specs = [[{'type':'domain'}, {'type':'table'}]]
fig_combined = make_subplots(rows=1, cols=2, specs=specs, subplot_titles=["", "Failure Mode Counts"])

# Add the donut chart and table to the combined figure
fig_combined.add_trace(fig_donut.data[0], row=1, col=1)
fig_combined.add_trace(fig_table.data[0], row=1, col=2)

# Update the layout of the combined figure
```

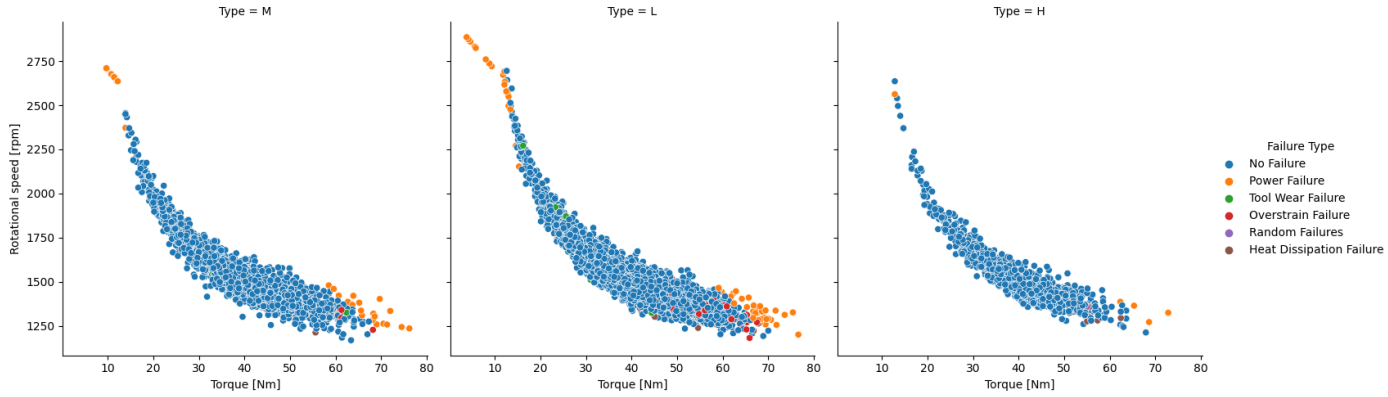
```
fig_combined.update_layout(height=500, showlegend=False)

fig_combined.show()
```



Failure Mode Counts

Failure Mode	Count
No Failure	9652
Heat Dissipation Failure	112
Power Failure	95
Overstrain Failure	78
Tool Wear Failure	45
Random	18

```
sns.relplot(data=df, x="Torque [Nm]", y="Rotational speed [rpm]", hue="Failure Type",col="Type",palette='tab10');
```




```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

It was transformed the categorical labels in the 'Failure Type' column & "Type" column of the DataFrame df into numerical values. Label encoding is a technique used to represent categorical data as integers, which is useful for various machine learning algorithms that require numerical inputs.

```
from sklearn.preprocessing import LabelEncoder
df['Type'] = LabelEncoder().fit_transform(df['Type'])

df.head()
```

	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Target	Failure Type	Temperature difference [K]
0	2	298.1	308.6	1551	42.8	0	0	No Failure	10.5

```
df['Failure Type'] = LabelEncoder().fit_transform(df['Failure Type'])
```

```
df
```

	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Target	Failure Type	Temperature difference [K]
0	2	298.1	308.6	1551	42.8	0	0	1	10.5
1	1	298.2	308.7	1408	46.3	3	0	1	10.5
2	1	298.1	308.5	1498	49.4	5	0	1	10.4
3	1	298.2	308.6	1433	39.5	7	0	1	10.4
4	1	298.2	308.7	1408	40.0	9	0	1	10.5
...	...	...	...	...	...	...	...	...	...
9995	2	298.8	308.4	1604	29.5	14	0	1	9.6
9996	0	298.9	308.4	1632	31.8	17	0	1	9.5
9997	2	299.0	308.6	1645	33.4	22	0	1	9.6
9998	0	299.0	308.7	1408	48.5	25	0	1	9.7
9999	2	299.0	308.7	1500	40.2	30	0	1	9.7

10000 rows × 9 columns

Train, Validation & Test the data set

```
X = df.drop(columns=["Failure Type", "Target"], axis=1)
y = df["Failure Type"]
```

After executing this line of code, the feature matrix 'X' will be standardized, and each column of X will have zero mean and unit variance. Standardizing the features is often a good practice, especially when using algorithms like support vector machines (SVM), k-nearest neighbors (KNN), or neural networks, as it can improve the performance and stability of the models

```
#X = StandardScaler().fit_transform(X)
#X
```

```
array([[ 1.33388944, -0.95238944, -0.94735989, ..., -1.69598374,
        -0.18732201,  0.49884932],
       [-0.33222278, -0.90239341, -0.879959  , ..., -1.6488517 ,
        -0.18732201,  0.49884932],
       [-0.33222278, -0.95238944, -1.01476077, ..., -1.61743034,
        -0.18732201,  0.39895359],
       ...,
       [ 1.33388944, -0.50242514, -0.94735989, ..., -1.35034876,
        -0.18732201, -0.40021228],
       [-1.998335  , -0.50242514, -0.879959  , ..., -1.30321671,
        -0.18732201, -0.30031654],
       [ 1.33388944, -0.50242514, -0.879959  , ..., -1.22466331,
        -0.18732201, -0.30031654]])
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=11)
```

```
#def scale_dataset(dataframe, oversample=False):
  #x = dataframe[dataframe.columns[:-1]].values
  #y = dataframe[dataframe.columns[-1]].values

  #scaler = StandardScaler()
  #x = scaler.fit_transform(x)

  #if oversample:
```

```
    #ros = RandomOverSampler()
    #x, y = ros.fit_resample(x, y)

  #data = np.hstack((x, np.reshape(y, (-1, 1))))

  #return data, x, y


#train, x_train, y_train = scale_dataset(train, oversample=True)
#valid, x_valid, y_valid = scale_dataset(valid, oversample=False)
#test, x_test, y_test = scale_dataset(test, oversample=False)
```

```
X_train
```

	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Temperature difference [K]
4199	2	302.5	311.1	1638	32.0	198	8.6
757	1	296.9	308.1	1555	33.7	229	11.2
4001	0	302.1	311.2	1598	37.9	148	9.1
8163	1	299.7	311.1	1574	36.4	135	11.4
4250	1	302.6	311.0	1660	38.0	99	8.4
...	...	...	...	...	...	...	...
1293	0	298.7	309.8	1582	33.4	121	11.1
4023	1	302.1	311.0	1364	50.0	205	8.9
7259	2	300.4	310.5	1500	43.0	9	10.1
5200	1	303.8	312.8	1416	58.7	156	9.0
3775	0	302.5	311.1	1850	23.3	203	8.6

8000 rows × 7 columns

Machine Learning Algorithms

Machine learning algorithms are computational techniques that allow computers to learn patterns and relationships from data without being explicitly programmed for them. These algorithms can be broadly categorized into three main types based on the learning paradigm: supervised learning, unsupervised learning, and reinforcement learning.In our case we mainly used supervised learning algorithms.

- Support Vector Machine
- Random Forest
- Decision Tree
- CatBoost
- XgBoost
- K-Nearest Neighbor(KNN)
- Gradient Boosting Classifier
- Logistic Regression
- Deep Neural Network

Hyperparameter tuning

Hyperparameters are parameters that are not learned directly from the training data during the training process of a machine learning model. Instead, they are set before the training begins and control the learning process. These parameters influence how the model is trained and how it makes predictions but are not updated through the model's learning process.

The process of training a machine learning model involves finding the best values for the model's hyperparameters, which can significantly impact the model's performance and generalization ability. The selection of appropriate hyperparameters is essential to ensure that the model performs well on new, unseen data and does not overfit or underfit the training data.

Some common examples of hyperparameters in various machine learning algorithms

- Learning Rate (Gradient Descent-based algorithms): The step size at which the model updates its parameters during training
- Number of Hidden Units (Neural Networks): The number of neurons in each layer of a neural network.
- Number of Trees (Random Forest): The number of decision trees in a random forest ensemble.
- Maximum Tree Depth (Decision Trees): The maximum depth allowed for a decision tree.
- Kernel Type and Kernel Parameters (Support Vector Machines): The type of kernel function and its associated parameters used to transform data into a higher-dimensional space.
- Number of Clusters (Clustering Algorithms): The number of clusters to be formed by the clustering algorithm.

GridSearchCV : takes a model and a grid of hyperparameter values as input. It then trains the model with each combination of hyperparameters in the grid and evaluates the model's performance on a held-out set of data. The combination of hyperparameters that produces the best model is then returned.

RandomizedSearchCV : takes a model and a grid of hyperparameter values as input. It then trains the model with a random subset of the hyperparameters in the grid and evaluates the model's performance on a held-out set of data. The combination of hyperparameters that produces the best model is then returned.

Bayesian optimization: It is a sequential model-based optimization algorithm for global optimization of black-box functions. It works by building a probabilistic model of the objective function and then using that model to select the next set of hyperparameters to evaluate. It is a more sophisticated approach to hyperparameter tuning than GridSearchCV or RandomizedSearchCV. It can find the best hyperparameters more quickly and reliably. However, it can also be more computationally expensive

Hyperparameter tuning frameworks

Hyperparameter tuning frameworks are tools that automate the process of finding the optimal values for the hyperparameters of a machine learning model. They can be used to improve the accuracy, performance, and reproducibility of machine learning models. Basically two hyperparameter tuning framworks were used in this projects.

- Scikit-learn
- Optuna

```
import time
```

Support Vector Machine

Normal Model Training

```
 #Support Vector Machines

from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
#from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
svc = SVC()
start_time_train = time.time()
svc.fit(X_train, y_train)
end_time_train = time.time()
y_pred_svc = svc.predict(X_test)

svc_train = round(svc.score(X_train, y_train) * 100, 4)
svc_accuracy = round(accuracy_score(y_pred_svc, y_test) * 100, 4)
time_to_train = end_time_train - start_time_train

print("Training Accuracy    :",svc_train ,"%")
print("Model Accuracy Score :",svc_accuracy ,"%")
print("Time to train the model:", time_to_train, "seconds")

print("\033[1;31m-------------------------------------------------------\033[0m")
print("Classification_Report: \n",classification_report(y_test,y_pred_svc))
print("\033[1;31m-------------------------------------------------------\033[0m")
```

```
#plot_confusion_matrix(svc, X_test, y_test);
```

```
Training Accuracy    : 96.75 %
Model Accuracy Score : 96.45 %
Time to train the model: 0.424206018447876 seconds
-----------------------------------------------------------
Classification_Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        22
           1       0.96      1.00      0.98      1927
           2       0.00      0.00      0.00        16
           3       0.75      0.17      0.27        18
           4       0.00      0.00      0.00         8
           5       0.00      0.00      0.00         9

    accuracy                           0.96      2000
   macro avg       0.29      0.19      0.21      2000
weighted avg       0.94      0.96      0.95      2000

-----------------------------------------------------------
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control
```

Hyperparameter tuning of svm using GridSearchCV

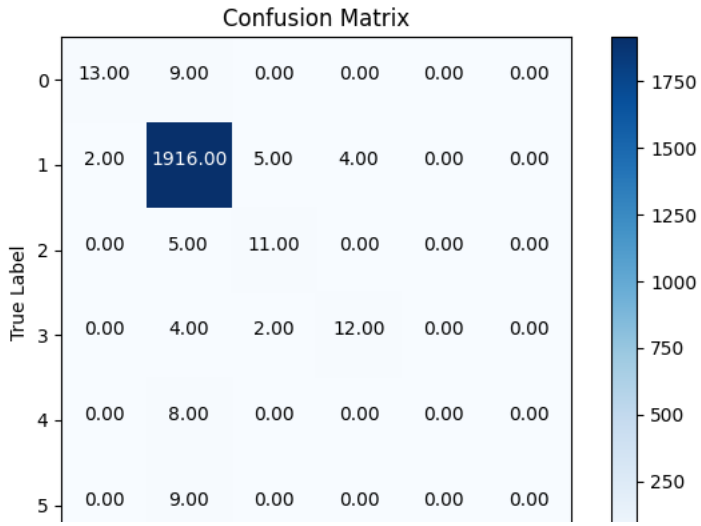
```
import time
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, accuracy_score
from sklearn.svm import SVC

# Step 2: Define hyperparameters and their values to search over
param_grid = {
    'C': [0.1, 1, 10],              # Regularization parameter
    'kernel': ['linear', 'rbf'],    # Kernel type ('linear' or 'rbf')
    'gamma': ['scale', 'auto'],     # Kernel coefficient for 'rbf'
}

# Step 3: Create an instance of GridSearchCV
svc = SVC()
grid_search = GridSearchCV(svc, param_grid, cv=5)

# Step 4: Fit the GridSearchCV instance on the training data and measure time
start_time = time.time()
grid_search.fit(X_train, y_train)
end_time = time.time()
total_time = end_time - start_time

# Step 5: Print the best hyperparameters and corresponding accuracy score
best_params = grid_search.best_params_
best_accuracy = round(grid_search.best_score_ * 100, 2)

print("Best Hyperparameters:", best_params)
print("Best Accuracy Score :", best_accuracy, "%")
print("Total Time Taken    :", round(total_time, 2), "seconds")
print("\033[1;31m-------------------------------------------------------\033[0m")

# Refit the model on the full training data using the best hyperparameters and measure time
best_svc = grid_search.best_estimator_
start_time = time.time()
best_svc.fit(X_train, y_train)

# Evaluate the model on the test data and measure time
y_pred_best = best_svc.predict(X_test)
best_svc_accuracy = round(accuracy_score(y_pred_best, y_test) * 100, 4)
end_time = time.time()
```

```
total_time = end_time - start_time

print("Best Model Training Accuracy:", round(best_svc.score(X_train, y_train) * 100, 4), "%")
print("Best Model Test Accuracy    :", best_svc_accuracy, "%")
print("Total Time Taken for Training and Prediction:", round(total_time, 2), "seconds")
print("\033[1;31m-------------------------------------------------------\033[0m")
print("Best Model Classification Report: \n", classification_report(y_test, y_pred_best))
```

```
Best Hyperparameters: {'C': 1, 'gamma': 'scale', 'kernel': 'linear'}
Best Accuracy Score : 98.17 %
Total Time Taken    : 571.9 seconds
-------------------------------------------------------
Best Model Training Accuracy: 98.3625 %
Best Model Test Accuracy    : 97.6 %
Total Time Taken for Training and Prediction: 18.3 seconds
-------------------------------------------------------
Best Model Classification Report:
               precision    recall  f1-score   support

           0       0.87      0.59      0.70        22
           1       0.98      0.99      0.99      1927
           2       0.61      0.69      0.65        16
           3       0.75      0.67      0.71        18
           4       0.00      0.00      0.00         8
           5       0.00      0.00      0.00         9

    accuracy                           0.98      2000
   macro avg       0.53      0.49      0.51      2000
weighted avg       0.97      0.98      0.97      2000

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control
```

Confusion matrix for best predicted lables from Hyperparameter tuning of svm using GridSearchCV and true lables.

```
from sklearn.metrics import confusion_matrix

# Assuming y_test and y_pred_best are true labels and best predicted labels, respectively
cm = confusion_matrix(y_test, y_pred_best)
# Assuming classes is a list containing the class names in the order of the confusion matrix
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
classes = ["0","1", "2", "3","4","5"]
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)

fmt = '.2f'  # Format to display the numerical values inside the matrix
thresh = cm.max() / 2.
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

plt.ylabel('True Label')
plt.xlabel('Best Predicted Label')
plt.tight_layout()
plt.show()
```



First we directly use GridSearchCV to tune hyperparameters and accurecy increased 96.45 % to 97.60 %.

Cross-Validation

```
from sklearn.model_selection import cross_val_predict, train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix


# Step 2: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 3: Create the SVM model with the best parameters
best_params = {'C': 1, 'gamma': 'scale', 'kernel': 'linear'}
model = SVC(C=best_params['C'], gamma=best_params['gamma'], kernel=best_params['kernel'])

# Step 4: Perform cross-validation and get predictions
num_folds = 5
predicted_labels = cross_val_predict(model, X_train, y_train, cv=num_folds)

# Step 5: Calculate evaluation metrics
accuracy = accuracy_score(y_train, predicted_labels)
precision = precision_score(y_train, predicted_labels, average='weighted')
recall = recall_score(y_train, predicted_labels, average='weighted')
f1 = f1_score(y_train, predicted_labels, average='weighted')
conf_matrix = confusion_matrix(y_train, predicted_labels)

print(f"Cross-validated Accuracy: {accuracy}")
print(f"Cross-validated Precision: {precision}")
print(f"Cross-validated Recall: {recall}")
print(f"Cross-validated F1-score: {f1}")
print("Confusion Matrix:")
print(conf_matrix)

# Step 6: Train the model on the entire training set and test it on the test set
model.fit(X_train, y_train)
test_predictions = model.predict(X_test)

# Step 7: Evaluate the model on the test set
test_accuracy = accuracy_score(y_test, test_predictions)
test_precision = precision_score(y_test, test_predictions, average='weighted')
test_recall = recall_score(y_test, test_predictions, average='weighted')
test_f1 = f1_score(y_test, test_predictions, average='weighted')
test_conf_matrix = confusion_matrix(y_test, test_predictions)

print(f"Test Accuracy: {test_accuracy}")
print(f"Test Precision: {test_precision}")
print(f"Test Recall: {test_recall}")
print(f"Test F1-score: {test_f1}")
print("Test Confusion Matrix:")
print(test_conf_matrix)
```

```
Cross-validated Accuracy: 0.995875
Cross-validated Precision: 0.9945435707906096
Cross-validated Recall: 0.995875
Cross-validated F1-score: 0.995198001402128
Confusion Matrix:
[[  95    0    2    0    0    0]
 [   1 7713    0    0    1    2]
 [   2    0   60    1    0    2]
 [   0    2    2   70    0    1]
 [   0   12    0    0    0    0]
 [   0    1    4    0    0   29]]
Test Accuracy: 0.9955
Test Precision: 0.9926723152920962
Test Recall: 0.9955
Test F1-score: 0.9939960368663596
Test Confusion Matrix:
[[  15    0    0    0    0    0]
 [   1 1934    0    0    0    0]
 [   0    0   13    0    0    0]
 [   0    0    0   20    0    0]
 [   0    6    0    0    0    0]
 [   0    0    2    0    0    9]]
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and be
  _warn_prf(average, modifier, msg_start, len(result))
```

Test accuracy 99.55%.

Random Forest

Normal Model Training

```
from sklearn.ensemble import RandomForestClassifier
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train, y_train)
y_pred_rf = random_forest.predict(X_test)
random_forest.score(X_train, y_train)

random_forest_train = round(random_forest.score(X_train, y_train) * 100, 4)
random_forest_accuracy = round(accuracy_score(y_pred_rf, y_test) * 100, 4)

print("Training Accuracy    :",random_forest_train ,"%")
print("Model Accuracy Score :",random_forest_accuracy ,"%")
print("\033[1m----------------------------------------------------------\033[0m")
print("Classification_Report: \n",classification_report(y_test,y_pred_rf))
print("\033[1m----------------------------------------------------------\033[0m")
```

```
Training Accuracy    : 100.0 %
Model Accuracy Score : 98.35 %
----------------------------------------------------------
Classification_Report:
               precision    recall  f1-score   support

           0       0.94      1.00      0.97        15
           1       0.99      1.00      0.99      1935
           2       0.80      0.62      0.70        13
           3       0.74      0.70      0.72        20
           4       0.00      0.00      0.00         6
           5       0.00      0.00      0.00        11

    accuracy                           0.98      2000
   macro avg       0.58      0.55      0.56      2000
weighted avg       0.97      0.98      0.98      2000

----------------------------------------------------------
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:
```

```
Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control
```

Hyperparameter tuning of Random Forest using GridSearchCV

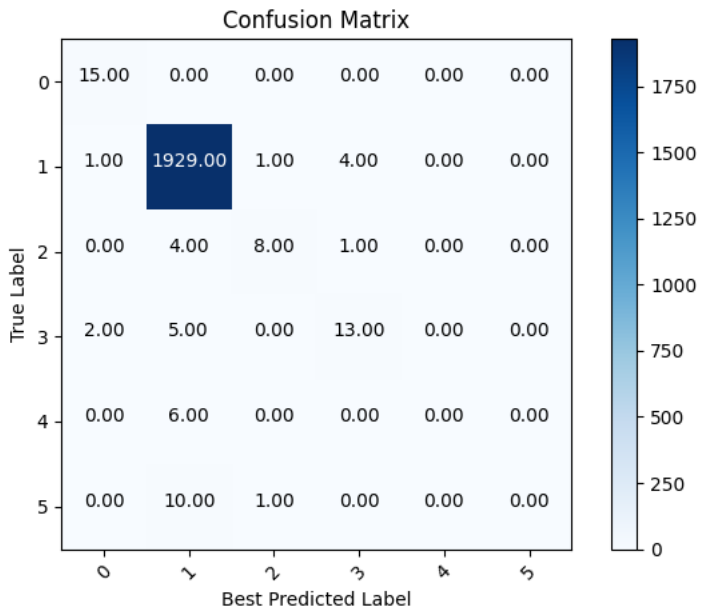
```
import time
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, accuracy_score

# Step 2: Define hyperparameters and their values to search over
param_grid = {
    'n_estimators': [50, 100, 150],    # Number of trees in the forest
    'max_depth': [None, 10, 20],       # Maximum depth of the trees
    'min_samples_split': [2, 5, 10],   # Minimum number of samples required to split an internal node
}

# Step 3: Create an instance of GridSearchCV
rfc = RandomForestClassifier()
grid_search = GridSearchCV(rfc, param_grid, cv=5)

# Step 4: Fit the GridSearchCV instance on the training data and measure time
start_time = time.time()
grid_search.fit(X_train, y_train)
end_time = time.time()
total_time = end_time - start_time

# Step 5: Print the best hyperparameters and corresponding accuracy score
best_params = grid_search.best_params_
best_accuracy = round(grid_search.best_score_ * 100, 2)

print("Best Hyperparameters:", best_params)
print("Best Accuracy Score :", best_accuracy, "%")
print("Total Time Taken    :", round(total_time, 2), "seconds")
print("\033[1m-------------------------------------------------------\033[0m")

# Refit the model on the full training data using the best hyperparameters and measure time
best_rfc = grid_search.best_estimator_
start_time = time.time()
best_rfc.fit(X_train, y_train)


# Evaluate the model on the test data and measure time
y_pred_best = best_rfc.predict(X_test)
best_rfc_accuracy = round(accuracy_score(y_pred_best, y_test) * 100, 4)
end_time = time.time()
total_time = end_time - start_time

print("Best Model Training Accuracy:", round(best_rfc.score(X_train, y_train) * 100, 4), "%")
print("Best Model Test Accuracy    :", best_rfc_accuracy, "%")
print("Total Time Taken for Training and Prediction:", round(total_time, 2), "seconds")
print("\033[1m-------------------------------------------------------\033[0m")
print("Best Model Classification Report: \n", classification_report(y_test, y_pred_best))
```

```
Best Hyperparameters: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 50}
Best Accuracy Score : 98.4 %
Total Time Taken    : 113.85 seconds
-------------------------------------------------------
Best Model Training Accuracy: 100.0 %
Best Model Test Accuracy    : 98.25 %
Total Time Taken for Training and Prediction: 0.49 seconds
-------------------------------------------------------
Best Model Classification Report:
                precision    recall  f1-score   support

            0       0.83      1.00      0.91        15
            1       0.99      1.00      0.99      1935
            2       0.80      0.62      0.70        13
            3       0.72      0.65      0.68        20
            4       0.00      0.00      0.00         6
            5       0.00      0.00      0.00        11

     accuracy                           0.98      2000
    macro avg       0.56      0.54      0.55      2000
 weighted avg       0.97      0.98      0.98      2000
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control
```

Confusion Matrix

```
from sklearn.metrics import confusion_matrix

# Assuming y_test and y_pred_best are true labels and best predicted labels, respectively
cm = confusion_matrix(y_test, y_pred_best)
# Assuming classes is a list containing the class names in the order of the confusion matrix
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
classes = ["0","1", "2", "3","4","5"]
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)

fmt = '.2f'  # Format to display the numerical values inside the matrix
thresh = cm.max() / 2.
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

plt.ylabel('True Label')
plt.xlabel('Best Predicted Label')
plt.tight_layout()
plt.show()
```



Cross-Validation

```
from sklearn.model_selection import cross_val_predict, train_test_split
from sklearn.ensemble import RandomForestClassifier  # For classification tasks
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 3: Create the Random Forest model with the best parameters
best_params = {'max_depth': None, 'min_samples_split': 10, 'n_estimators': 50}
model = RandomForestClassifier(
    max_depth=best_params['max_depth'],
    min_samples_split=best_params['min_samples_split'],
    n_estimators=best_params['n_estimators'],
    random_state=42)

# Step 4: Perform cross-validation and get predictions
num_folds = 5
predicted_labels = cross_val_predict(model, X_train, y_train, cv=num_folds)

# Step 5: Calculate evaluation metrics
accuracy = accuracy_score(y_train, predicted_labels)
precision = precision_score(y_train, predicted_labels, average='weighted')
recall = recall_score(y_train, predicted_labels, average='weighted')
f1 = f1_score(y_train, predicted_labels, average='weighted')
conf_matrix = confusion_matrix(y_train, predicted_labels)

print("\033[1;31m-------------------------------------------------------\033[0m")
print(f"Cross-validated Accuracy: {accuracy}")
print(f"Cross-validated Precision: {precision}")
print(f"Cross-validated Recall: {recall}")
print(f"Cross-validated F1-score: {f1}")
print("Confusion Matrix:")
print(conf_matrix)

# Step 6: Train the model on the entire training set and test it on the test set
model.fit(X_train, y_train)
test_predictions = model.predict(X_test)

# Step 7: Evaluate the model on the test set
test_accuracy = accuracy_score(y_test, test_predictions)
test_precision = precision_score(y_test, test_predictions, average='weighted')
test_recall = recall_score(y_test, test_predictions, average='weighted')
test_f1 = f1_score(y_test, test_predictions, average='weighted')
test_conf_matrix = confusion_matrix(y_test, test_predictions)

print("\033[1;31m-------------------------------------------------------\033[0m")
print(f"Test Accuracy: {test_accuracy}")
print(f"Test Precision: {test_precision}")
print(f"Test Recall: {test_recall}")
print(f"Test F1-score: {test_f1}")
print("Test Confusion Matrix:")
print(test_conf_matrix)

# Additional classification report
print("\033[1;31m-------------------------------------------------------\033[0m")
print("Test Classification Report:")
print(classification_report(y_test, test_predictions))
print("\033[1;31m-------------------------------------------------------\033[0m")
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this beha

-------------------------------------------------------
Cross-validated Accuracy: 0.982625
Cross-validated Precision: 0.9758846091234584
Cross-validated Recall: 0.982625
Cross-validated F1-score: 0.9785733800997599
Confusion Matrix:
[[  88    8    0    1    0    0]
 [   5 7700    3    8    0    1]
 [   1   37   25    2    0    0]
 [   2   22    3   48    0    0]
 [   0   12    0    0    0    0]
 [   0   33    1    0    0    0]]
-------------------------------------------------------
Test Accuracy: 0.982
Test Precision: 0.9734637671811669
Test Recall: 0.982
Test F1-score: 0.9773342167607133
Test Confusion Matrix:
[[  15    0    0    0    0    0]
 [   2 1929    0    4    0    0]
 [   0    5    7    1    0    0]
```

```
[   2    5    0   13    0    0]
[   0    6    0    0    0    0]
[   0   10    1    0    0    0]]
----------------------------------------------------------
Test Classification Report:
              precision    recall  f1-score   support

           0       0.79      1.00      0.88        15
           1       0.99      1.00      0.99      1935
           2       0.88      0.54      0.67        13
           3       0.72      0.65      0.68        20
           4       0.00      0.00      0.00         6
           5       0.00      0.00      0.00        11

    accuracy                           0.98      2000
   macro avg       0.56      0.53      0.54      2000
weighted avg       0.97      0.98      0.98      2000

----------------------------------------------------------
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this beha

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
```

Decision Tree

Normal Model Training

```
# Decision Tree
from sklearn.tree import DecisionTreeClassifier
decision = DecisionTreeClassifier()
decision.fit(X_train, y_train)
y_pred_dec = decision.predict(X_test)

decision_train = round(decision.score(X_train, y_train) * 100, 4)
decision_accuracy = round(accuracy_score(y_pred_dec, y_test) * 100, 4)

print("Training Accuracy    :",decision_train ,"%")
print("Model Accuracy Score :",decision_accuracy ,"%")
print("\033[1;31m----------------------------------------------------------\033[0m")
print("Classification_Report: \n",classification_report(y_test,y_pred_dec))
print("\033[1;31m----------------------------------------------------------\033[0m")
```

```
Training Accuracy    : 100.0 %
Model Accuracy Score : 96.9 %
----------------------------------------------------------
Classification_Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       0.99      0.98      0.98      1935
           2       0.44      0.62      0.52        13
           3       0.73      0.80      0.76        20
           4       0.00      0.00      0.00         6
           5       0.07      0.09      0.08        11

    accuracy                           0.97      2000
   macro avg       0.54      0.58      0.56      2000
weighted avg       0.97      0.97      0.97      2000

----------------------------------------------------------
```

Hyperparameter tuning of Decision Tree using GridSearchCV

```
import time
from sklearn.model_selection import GridSearchCV
```

```
from sklearn.metrics import classification_report, accuracy_score
from sklearn.tree import DecisionTreeClassifier

# Step 2: Define hyperparameters and their values to search over
param_grid = {
    'criterion': ['gini', 'entropy'],  # Criterion for splitting ('gini' or 'entropy')
    'max_depth': [None, 5, 10, 20],    # Maximum depth of the tree
    'min_samples_split': [2, 5, 10],   # Minimum samples required to split an internal node
}

# Step 3: Create an instance of GridSearchCV
dt_classifier = DecisionTreeClassifier()
grid_search = GridSearchCV(dt_classifier, param_grid, cv=5)

# Step 4: Fit the GridSearchCV instance on the training data and measure time
start_time = time.time()
grid_search.fit(X_train, y_train)
end_time = time.time()
total_time = end_time - start_time

# Step 5: Print the best hyperparameters and corresponding accuracy score
best_params = grid_search.best_params_
best_accuracy = round(grid_search.best_score_ * 100, 2)

print("Best Hyperparameters:", best_params)
print("Best Accuracy Score :", best_accuracy, "%")
print("Total Time Taken    :", round(total_time, 2), "seconds")
print("\033[1m-----------------------------------------------------------\033[0m")

# Refit the model on the full training data using the best hyperparameters and measure time
best_dt_classifier = grid_search.best_estimator_
start_time = time.time()
best_dt_classifier.fit(X_train, y_train)


# Evaluate the model on the test data and measure time
y_pred_best = best_dt_classifier.predict(X_test)
best_dt_accuracy = round(accuracy_score(y_pred_best, y_test) * 100, 4)
end_time = time.time()
total_time = end_time - start_time

print("Best Model Training Accuracy:", round(best_dt_classifier.score(X_train, y_train) * 100, 4), "%")
print("Best Model Test Accuracy    :", best_dt_accuracy, "%")
print("Total Time Taken for Training and Prediction:", round(total_time, 2), "seconds")
print("\033[1m-----------------------------------------------------------\033[0m")
print("Best Model Classification Report: \n", classification_report(y_test, y_pred_best))
```

```
Best Hyperparameters: {'criterion': 'gini', 'max_depth': 10, 'min_samples_split': 2}
Best Accuracy Score : 98.32 %
Total Time Taken    : 3.3 seconds
-----------------------------------------------------------
Best Model Training Accuracy: 99.55 %
Best Model Test Accuracy    : 97.85 %
Total Time Taken for Training and Prediction: 0.03 seconds
-----------------------------------------------------------
Best Model Classification Report:
               precision    recall  f1-score   support

            0       1.00      1.00      1.00        15
            1       0.99      0.99      0.99      1935
            2       0.44      0.62      0.52        13
            3       0.76      0.80      0.78        20
            4       0.00      0.00      0.00         6
            5       0.00      0.00      0.00        11

     accuracy                           0.98      2000
    macro avg       0.53      0.57      0.55      2000
weighted avg       0.97      0.98      0.98      2000
```

Cross-Validation

```
from sklearn.model_selection import cross_val_predict, train_test_split
from sklearn.tree import DecisionTreeClassifier  # For classification tasks
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report
```

```
# Step 2: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 3: Create the Decision Tree model with the best parameters
best_params = {'criterion': 'entropy', 'max_depth': 5, 'min_samples_split': 5}
model = DecisionTreeClassifier(
    criterion=best_params['criterion'],
    max_depth=best_params['max_depth'],
    min_samples_split=best_params['min_samples_split'],
    random_state=42)

# Step 4: Perform cross-validation and get predictions
num_folds = 5
predicted_labels = cross_val_predict(model, X_train, y_train, cv=num_folds)

# Step 5: Calculate evaluation metrics
accuracy = accuracy_score(y_train, predicted_labels)
precision = precision_score(y_train, predicted_labels, average='weighted')
recall = recall_score(y_train, predicted_labels, average='weighted')
f1 = f1_score(y_train, predicted_labels, average='weighted')
conf_matrix = confusion_matrix(y_train, predicted_labels)

print("\033[1;31m-------------------------------------------------------\033[0m")
print(f"Cross-validated Accuracy: {accuracy}")
print(f"Cross-validated Precision: {precision}")
print(f"Cross-validated Recall: {recall}")
print(f"Cross-validated F1-score: {f1}")
print("Confusion Matrix:")
print(conf_matrix)

# Step 6: Train the model on the entire training set and test it on the test set
model.fit(X_train, y_train)
test_predictions = model.predict(X_test)

# Step 7: Evaluate the model on the test set
test_accuracy = accuracy_score(y_test, test_predictions)
test_precision = precision_score(y_test, test_predictions, average='weighted')
test_recall = recall_score(y_test, test_predictions, average='weighted')
test_f1 = f1_score(y_test, test_predictions, average='weighted')
test_conf_matrix = confusion_matrix(y_test, test_predictions)

print("\033[1;31m-------------------------------------------------------\033[0m")
print(f"Test Accuracy: {test_accuracy}")
print(f"Test Precision: {test_precision}")
print(f"Test Recall: {test_recall}")
print(f"Test F1-score: {test_f1}")
print("Test Confusion Matrix:")
print(test_conf_matrix)

# Additional classification report
print("\033[1;31m-------------------------------------------------------\033[0m")
print("Test Classification Report:")
print(classification_report(y_test, test_predictions))
print("\033[1;31m-------------------------------------------------------\033[0m")
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this beha

-------------------------------------------------------
Cross-validated Accuracy: 0.979875
Cross-validated Precision: 0.9728741805085719
Cross-validated Recall: 0.979875
Cross-validated F1-score: 0.9762334461909616
Confusion Matrix:
[[  93    3    1    0    0    0]
 [   6 7678   17   16    0    0]
 [   1   32   27    5    0    0]
 [   1   28    5   41    0    0]
 [   0   12    0    0    0    0]
 [   0   34    0    0    0    0]]
-------------------------------------------------------
Test Accuracy: 0.98
Test Precision: 0.971115096912363
Test Recall: 0.98
Test F1-score: 0.9752067667560977
Test Confusion Matrix:
[[  15    0    0    0    0    0]
```

```
[   4 1926    2    3    0    0]
[   0    8    4    1    0    0]
[   2    2    1   15    0    0]
[   0    6    0    0    0    0]
[   0   10    1    0    0    0]]
--------------------------------------------------------
Test Classification Report:
              precision    recall  f1-score   support

           0       0.71      1.00      0.83        15
           1       0.99      1.00      0.99      1935
           2       0.50      0.31      0.38        13
           3       0.79      0.75      0.77        20
           4       0.00      0.00      0.00         6
           5       0.00      0.00      0.00        11

    accuracy                           0.98      2000
   macro avg       0.50      0.51      0.50      2000
weighted avg       0.97      0.98      0.98      2000

--------------------------------------------------------
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this beha

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:
```

CatBoost

CatBoost is a powerful open-source gradient boosting library designed for machine learning tasks. It was developed by Yandex and offers state-of-the-art performance in various supervised learning tasks, such as classification, regression, and ranking. CatBoost is particularly known for its ability to handle categorical features efficiently, without requiring extensive preprocessing

Normal Model Training

```
#pip install numpy pandas catboost
```

```
pip install catboost
```

```
Collecting catboost
  Downloading catboost-1.2-cp310-cp310-manylinux2014_x86_64.whl (98.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 98.6/98.6 MB 2.8 MB/s eta 0:00:00
Requirement already satisfied: graphviz in /usr/local/lib/python3.10/dist-packages (from catboost) (0.20.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from catboost) (3.7.1)
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.10/dist-packages (from catboost) (1.22.4)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.10/dist-packages (from catboost) (1.5.3)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from catboost) (1.10.1)
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages (from catboost) (5.13.1)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from catboost) (1.16.0)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24->catboost) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24->catboost) (2022.7.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (1.1.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (4.41.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (1.4.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (23.1)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (3.1.0)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from plotly->catboost) (8.2.2)
Installing collected packages: catboost
Successfully installed catboost-1.2
```

```
from catboost import CatBoostClassifier
catboost_model = CatBoostClassifier(iterations=1000, learning_rate=0.1, depth=6, loss_function='MultiClass')
catboost_model.fit(X_train, y_train, verbose=False)
```

```
y_pred_cat = catboost_model.predict(X_test)

catboost_train = round(catboost_model.score(X_train, y_train) * 100, 4)
catboost_accuracy = round(accuracy_score(y_pred_rf, y_test) * 100, 4)

print("Training Accuracy    : {:.4f}%".format(catboost_train))
print("Model Accuracy Score : {:.4f}%".format(catboost_accuracy))
print("\033[1m-----------------------------------------------------------\033[0m")
print("Classification_Report: \n",classification_report(y_test,y_pred_dec))
print("\033[1m-----------------------------------------------------------\033[0m")
```

```
Training Accuracy    : 99.9125%
Model Accuracy Score : 98.3500%
-----------------------------------------------------------
Classification_Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       0.99      0.98      0.98      1935
           2       0.44      0.62      0.52        13
           3       0.73      0.80      0.76        20
           4       0.00      0.00      0.00         6
           5       0.07      0.09      0.08        11

    accuracy                           0.97      2000
   macro avg       0.54      0.58      0.56      2000
weighted avg       0.97      0.97      0.97      2000

-----------------------------------------------------------
```

Hyperparameter tuning of CatBoost using GridSearchCV

```
import time
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, accuracy_score
from catboost import CatBoostClassifier

# Step 2: Define hyperparameters and their values to search over
param_grid = {
    'iterations': [100, 200, 300],      # Number of trees (boosting iterations)
    'learning_rate': [0.01, 0.1, 0.2],  # Step size for gradient descent
    'depth': [4, 6, 8],                 # Depth of the trees
}

# Step 3: Create an instance of GridSearchCV
catboost = CatBoostClassifier()
grid_search = GridSearchCV(catboost, param_grid, cv=5)

# Step 4: Fit the GridSearchCV instance on the training data and measure time
start_time = time.time()
grid_search.fit(X_train, y_train)
end_time = time.time()
total_time = end_time - start_time

# Step 5: Print the best hyperparameters and corresponding accuracy score
best_params = grid_search.best_params_
best_accuracy = round(grid_search.best_score_ * 100, 4)

print("Best Hyperparameters:", best_params)
print("Best Accuracy Score :", best_accuracy, "%")
print("Total Time Taken    :", round(total_time, 4), "seconds")
print("\033[1m-----------------------------------------------------------\033[0m")

# Refit the model on the full training data using the best hyperparameters and measure time
best_catboost = grid_search.best_estimator_
start_time = time.time()
best_catboost.fit(X_train, y_train)


# Evaluate the model on the test data and measure time
y_pred_best = best_catboost.predict(X_test)
best_catboost_accuracy = round(accuracy_score(y_pred_best, y_test) * 100, 4)
end_time = time.time()
total_time = end_time - start_time

print("Best Model Training Accuracy:", round(best_catboost.score(X_train, y_train) * 100, 4), "%")
```

```
print("Best Model Test Accuracy    :", best_catboost_accuracy, "%")
print("Total Time Taken for Training and Prediction:", round(total_time, 2), "seconds")
print("\033[1m-----------------------------------------------------------\033[0m")
print("Best Model Classification Report: \n", classification_report(y_test, y_pred_best))
print("\033[1m-----------------------------------------------------------\033[0m")
```

```
174:    learn: 0.0318049        total: 1.03s    remaining: 148ms
175:    learn: 0.0317669        total: 1.04s    remaining: 142ms
176:    learn: 0.0316828        total: 1.04s    remaining: 136ms
177:    learn: 0.0316132        total: 1.05s    remaining: 130ms
178:    learn: 0.0315533        total: 1.06s    remaining: 124ms
179:    learn: 0.0314375        total: 1.06s    remaining: 118ms
180:    learn: 0.0313695        total: 1.07s    remaining: 113ms
181:    learn: 0.0312930        total: 1.08s    remaining: 107ms
182:    learn: 0.0312034        total: 1.08s    remaining: 101ms
183:    learn: 0.0311287        total: 1.09s    remaining: 94.8ms
184:    learn: 0.0310691        total: 1.09s    remaining: 88.8ms
185:    learn: 0.0309552        total: 1.1s     remaining: 82.9ms
186:    learn: 0.0308345        total: 1.11s    remaining: 76.9ms
187:    learn: 0.0307289        total: 1.12s    remaining: 71.4ms
188:    learn: 0.0306933        total: 1.13s    remaining: 65.5ms
189:    learn: 0.0305093        total: 1.13s    remaining: 59.7ms
190:    learn: 0.0304755        total: 1.14s    remaining: 53.7ms
191:    learn: 0.0304102        total: 1.14s    remaining: 47.7ms
192:    learn: 0.0303325        total: 1.15s    remaining: 41.7ms
193:    learn: 0.0302950        total: 1.16s    remaining: 35.7ms
194:    learn: 0.0302421        total: 1.16s    remaining: 29.8ms
195:    learn: 0.0300427        total: 1.17s    remaining: 23.8ms
196:    learn: 0.0299174        total: 1.17s    remaining: 17.9ms
197:    learn: 0.0298298        total: 1.18s    remaining: 11.9ms
198:    learn: 0.0297921        total: 1.18s    remaining: 5.95ms
199:    learn: 0.0297077        total: 1.19s    remaining: 0us
Best Model Training Accuracy: 99.325 %
Best Model Test Accuracy    : 98.35 %
Total Time Taken for Training and Prediction: 1.3 seconds
-----------------------------------------------------------
Best Model Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.93      0.85        15
           1       0.99      1.00      0.99      1935
           2       0.77      0.77      0.77        13
           3       0.88      0.70      0.78        20
           4       0.00      0.00      0.00         6
           5       0.00      0.00      0.00        11

    accuracy                           0.98      2000
   macro avg       0.57      0.57      0.56      2000
weighted avg       0.98      0.98      0.98      2000

-----------------------------------------------------------
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
```

Cross-Validation

```
from catboost import CatBoostClassifier  # For classification tasks
from sklearn.model_selection import cross_val_predict, train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 3: Create the CatBoost model with the best parameters
best_params = {'depth': 6, 'iterations': 200, 'learning_rate': 0.2}
model = CatBoostClassifier(
    depth=best_params['depth'],
    iterations=best_params['iterations'],
    learning_rate=best_params['learning_rate'],
    random_seed=42,
```

```
    verbose=False)

# Step 4: Perform cross-validation and get predictions
num_folds = 5
predicted_labels = cross_val_predict(model, X_train, y_train, cv=num_folds)

# Step 5: Calculate evaluation metrics
accuracy = accuracy_score(y_train, predicted_labels)
precision = precision_score(y_train, predicted_labels, average='weighted')
recall = recall_score(y_train, predicted_labels, average='weighted')
f1 = f1_score(y_train, predicted_labels, average='weighted')
conf_matrix = confusion_matrix(y_train, predicted_labels)

print("\033[1;31m-------------------------------------------------------\033[0m")
print(f"Cross-validated Accuracy: {accuracy}")
print(f"Cross-validated Precision: {precision}")
print(f"Cross-validated Recall: {recall}")
print(f"Cross-validated F1-score: {f1}")
print("Confusion Matrix:")
print(conf_matrix)

# Step 6: Train the model on the entire training set and test it on the test set
model.fit(X_train, y_train)
test_predictions = model.predict(X_test)

# Step 7: Evaluate the model on the test set
test_accuracy = accuracy_score(y_test, test_predictions)
test_precision = precision_score(y_test, test_predictions, average='weighted')
test_recall = recall_score(y_test, test_predictions, average='weighted')
test_f1 = f1_score(y_test, test_predictions, average='weighted')
test_conf_matrix = confusion_matrix(y_test, test_predictions)

print("\033[1;31m-------------------------------------------------------\033[0m")
print(f"Test Accuracy: {test_accuracy}")
print(f"Test Precision: {test_precision}")
print(f"Test Recall: {test_recall}")
print(f"Test F1-score: {test_f1}")
print("Test Confusion Matrix:")
print(test_conf_matrix)

# Additional classification report
print("\033[1;31m-------------------------------------------------------\033[0m")
print("Test Classification Report:")
print(classification_report(y_test, test_predictions))
print("\033[1;31m-------------------------------------------------------\033[0m")
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this beha

-------------------------------------------------------
Cross-validated Accuracy: 0.983125
Cross-validated Precision: 0.9780832375288391
Cross-validated Recall: 0.983125
Cross-validated F1-score: 0.9802911995100735
Confusion Matrix:
[[  90    5    2    0    0    0]
 [   6 7686    5   16    0    4]
 [   1   14   47    2    0    1]
 [   3   26    5   41    0    0]
 [   0   12    0    0    0    0]
 [   0   32    1    0    0    1]]
-------------------------------------------------------
Test Accuracy: 0.9855
Test Precision: 0.9770584531955501
Test Recall: 0.9855
Test F1-score: 0.9811443270423533
Test Confusion Matrix:
[[  15    0    0    0    0    0]
 [   1 1931    1    2    0    0]
 [   0    4    9    0    0    0]
 [   2    2    0   16    0    0]
 [   0    6    0    0    0    0]
 [   0   10    1    0    0    0]]
-------------------------------------------------------
Test Classification Report:
              precision    recall  f1-score   support

           0       0.83      1.00      0.91        15
           1       0.99      1.00      0.99      1935
```

```
              2       0.82      0.69      0.75        13
              3       0.89      0.80      0.84        20
              4       0.00      0.00      0.00         6
              5       0.00      0.00      0.00        11

       accuracy                           0.99      2000
      macro avg       0.59      0.58      0.58      2000
   weighted avg       0.98      0.99      0.98      2000

   ----------------------------------------------------------
   /usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

   Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this beha

   /usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

   Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont

   /usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

   Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont

   /usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:
```

XgBoost

XGBoost (Extreme Gradient Boosting) is a popular open-source gradient boosting library designed for machine learning tasks. It is known for its high performance, scalability, and flexibility, making it a popular choice for various supervised learning problems, including classification and regression. Here's a brief overview of using XGBoost for classification:

Normal Model Training

```
import xgboost as xgb
import numpy as np
from sklearn.metrics import accuracy_score, classification_report

# Convert the data to NumPy arrays if they are not already
X_train_np = np.array(X_train)
y_train_np = np.array(y_train)

# Create XGBoost Classifier
xgb_classifier = xgb.XGBClassifier()

# Train the model
xgb_classifier.fit(X_train_np, y_train_np)

# Make predictions on the test set
y_pred_xgb = xgb_classifier.predict(X_test)

# Calculate training accuracy
xgb_train_accuracy = round(xgb_classifier.score(X_train_np, y_train_np) * 100, 4)

# Calculate accuracy score on the test set
xgb_accuracy = round(accuracy_score(y_pred_xgb, y_test) * 100, 4)

print("\033[1m----------------------------------------------------------\033[0m")
print("Training Accuracy    :", xgb_train_accuracy, "%")
print("Model Accuracy Score :", xgb_accuracy, "%")
print("\033[1m----------------------------------------------------------\033[0m")
print("Classification_Report: \n", classification_report(y_test, y_pred_xgb))
print("\033[1m----------------------------------------------------------\033[0m")
```

```
   ----------------------------------------------------------
   Training Accuracy    : 100.0 %
   Model Accuracy Score : 98.4 %
   ----------------------------------------------------------
   Classification_Report:
                 precision    recall  f1-score   support

              0       0.83      1.00      0.91        15
              1       0.99      1.00      0.99      1935
              2       0.71      0.77      0.74        13
              3       0.82      0.70      0.76        20
```

```
              4       0.00      0.00      0.00         6
              5       0.00      0.00      0.00        11

       accuracy                           0.98      2000
      macro avg       0.56      0.58      0.57      2000
   weighted avg       0.98      0.98      0.98      2000

   ------------------------------------------------------------
   /usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

   Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control

   /usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

   Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control

   /usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

   Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control
```

```
"""import time
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, accuracy_score
import xgboost as xgb

# Step 2: Define hyperparameters and their values to search over
param_grid = {
    'n_estimators': [100, 200, 300],      # Number of trees (boosting iterations)
    'learning_rate': [0.01, 0.1, 0.2],   # Step size for gradient descent
    'max_depth': [4, 6, 8],               # Depth of the trees
}

# Step 3: Create an instance of GridSearchCV
xgb_model = xgb.XGBClassifier()
grid_search = GridSearchCV(xgb_model, param_grid, cv=5)

# Step 4: Fit the GridSearchCV instance on the training data and measure time
start_time = time.time()
grid_search.fit(X_train, y_train)
end_time = time.time()
total_time = end_time - start_time

# Step 5: Print the best hyperparameters and corresponding accuracy score
best_params = grid_search.best_params_
best_accuracy = round(grid_search.best_score_ * 100, 4)

print("Best Hyperparameters:", best_params)
print("Best Accuracy Score :", best_accuracy, "%")
print("Total Time Taken    :", round(total_time, 4), "seconds")
print("\033[1m------------------------------------------------------------\033[0m")

# Refit the model on the full training data using the best hyperparameters and measure time
best_xgb_model = grid_search.best_estimator_
start_time = time.time()
best_xgb_model.fit(X_train, y_train)
end_time = time.time()
total_time = end_time - start_time

# Evaluate the model on the test data and measure time
y_pred_best = best_xgb_model.predict(X_test)
best_xgb_model_accuracy = round(accuracy_score(y_pred_best, y_test) * 100, 4)

print("Best Model Training Accuracy:", round(best_xgb_model.score(X_train, y_train) * 100, 4), "%")
print("Best Model Test Accuracy    :", best_xgb_model_accuracy, "%")
print("Total Time Taken for Training and Prediction:", round(total_time, 2), "seconds")
print("\033[1m------------------------------------------------------------\033[0m")
print("Best Model Classification Report: \n", classification_report(y_test, y_pred_best))"""
```

Hyperparameter tuning of XgBoost using GridSearchCV

```
import time
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, accuracy_score
```

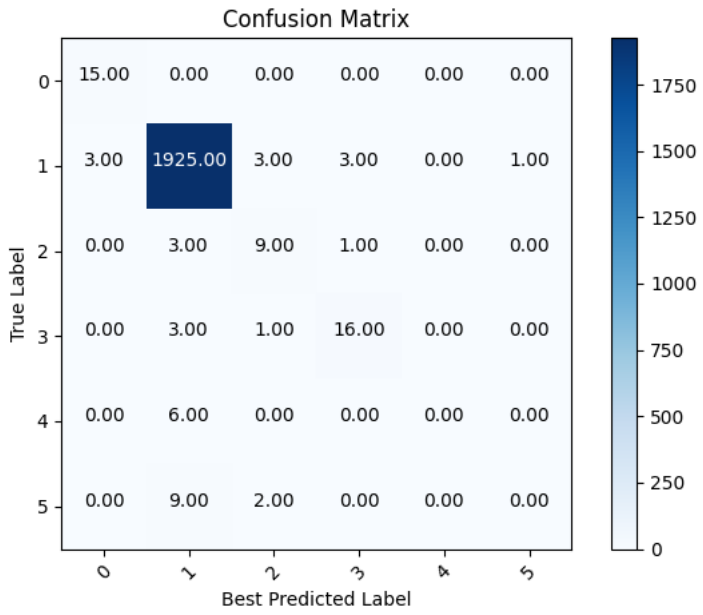
```
import xgboost as xgb

# Assuming the feature names in X_train and X_test are not in the correct format
X_train.columns = [str(col).replace("[", "_").replace("]", "_").replace("<", "_") for col in X_train.columns]
X_test.columns = [str(col).replace("[", "_").replace("]", "_").replace("<", "_") for col in X_test.columns]


# Step 2: Define hyperparameters and their values to search over
param_grid = {
    'learning_rate': [0.01, 0.1, 0.2],      # Learning rate for boosting
    'max_depth': [3, 5, 7],                 # Maximum depth of a tree
    'n_estimators': [50, 100, 200],         # Number of boosting rounds
    'subsample': [0.8, 0.9, 1.0],           # Subsample ratio of the training instances
}

# Step 3: Create an instance of GridSearchCV with XGBoost classifier
xgb_classifier = xgb.XGBClassifier()
grid_search = GridSearchCV(xgb_classifier, param_grid, cv=5)

# Step 4: Fit the GridSearchCV instance on the training data and measure time
start_time = time.time()
grid_search.fit(X_train, y_train)
end_time = time.time()
total_time = end_time - start_time

# Step 5: Print the best hyperparameters and corresponding accuracy score
best_params = grid_search.best_params_
best_accuracy = round(grid_search.best_score_ * 100, 4)

print("Best Hyperparameters:", best_params)
print("Best Accuracy Score :", best_accuracy, "%")
print("Total Time Taken    :", round(total_time, 4), "seconds")
print("\033[1m-------------------------------------------------------\033[0m")

# Refit the model on the full training data using the best hyperparameters and measure time
best_xgb_classifier = grid_search.best_estimator_
start_time = time.time()
best_xgb_classifier.fit(X_train, y_train)


# Evaluate the model on the test data and measure time
y_pred_best = best_xgb_classifier.predict(X_test)
best_xgb_accuracy = round(accuracy_score(y_pred_best, y_test) * 100, 4)
end_time = time.time()
total_time = end_time - start_time

print("Best Model Training Accuracy:", round(best_xgb_classifier.score(X_train, y_train) * 100, 4), "%")
print("Best Model Test Accuracy    :", best_xgb_accuracy, "%")
print("Total Time Taken for Training and Prediction:", round(total_time, 2), "seconds")
print("\033[1m-------------------------------------------------------\033[0m")
print("Best Model Classification Report: \n", classification_report(y_test, y_pred_best))
```

```
Best Hyperparameters: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 50, 'subsample': 1.0}
Best Accuracy Score : 98.775 %
Total Time Taken    : 853.2675 seconds
-------------------------------------------------------
Best Model Training Accuracy: 99.5375 %
Best Model Test Accuracy    : 98.25 %
Total Time Taken for Training and Prediction: 0.95 seconds
-------------------------------------------------------
Best Model Classification Report:
               precision    recall  f1-score   support

           0       0.83      1.00      0.91        15
           1       0.99      0.99      0.99      1935
           2       0.60      0.69      0.64        13
           3       0.80      0.80      0.80        20
           4       0.00      0.00      0.00         6
           5       0.00      0.00      0.00        11

    accuracy                           0.98      2000
   macro avg       0.54      0.58      0.56      2000
weighted avg       0.98      0.98      0.98      2000

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control
```

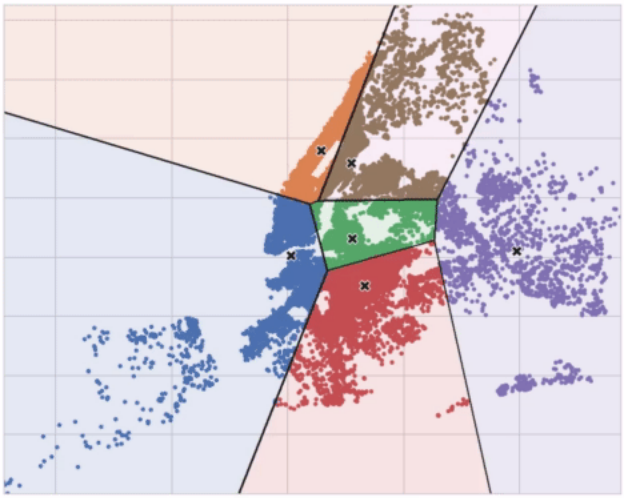
Confusion Matrix

```
from sklearn.metrics import confusion_matrix

# Assuming y_test and y_pred_best are true labels and best predicted labels, respectively
cm = confusion_matrix(y_test, y_pred_best)
# Assuming classes is a list containing the class names in the order of the confusion matrix
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
classes = ["0","1", "2", "3","4","5"]
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)

fmt = '.2f'  # Format to display the numerical values inside the matrix
thresh = cm.max() / 2.
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

plt.ylabel('True Label')
plt.xlabel('Best Predicted Label')
plt.tight_layout()
plt.show()
```



Cross-Validation

K-Nearest Neighbor(KNN)



How KNN works

Normal Model Training

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

# Assuming you have already split your dataset into X_train, X_test, y_train, and y_test

# K-nearest neighbor classifier with k=5 (you can change the value of n_neighbors as needed)
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)

# Calculate training accuracy
knn_train = round(knn.score(X_train, y_train) * 100, 4)

# Calculate model accuracy score
knn_accuracy = round(accuracy_score(y_pred_knn, y_test) * 100, 4)

print("Training Accuracy    :", knn_train, "%")
print("Model Accuracy Score :", knn_accuracy, "%")
print("\033[1m-------------------------------------------------------\033[0m")
print("Classification_Report: \n", classification_report(y_test, y_pred_knn))
print("\033[1m-------------------------------------------------------\033[0m")
```

```
Training Accuracy    : 97.075 %
Model Accuracy Score : 96.85 %
-------------------------------------------------------
Classification_Report:
               precision    recall  f1-score   support

           0       0.00      0.00      0.00        15
           1       0.97      1.00      0.98      1935
           2       0.33      0.08      0.12        13
           3       0.75      0.15      0.25        20
           4       0.00      0.00      0.00         6
           5       0.00      0.00      0.00        11

    accuracy                           0.97      2000
   macro avg       0.34      0.20      0.23      2000
weighted avg       0.95      0.97      0.96      2000

-------------------------------------------------------
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control
```

Hyperparameter tuning of KNN using GridSearchCV

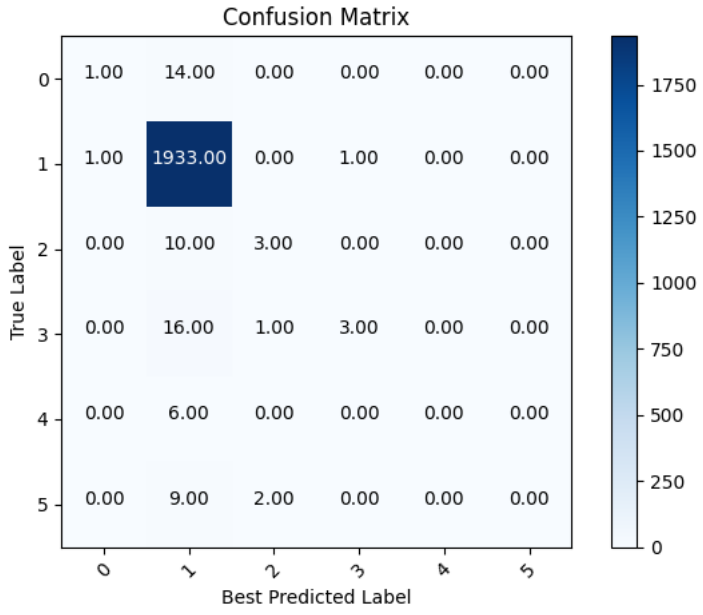
```
import time
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, accuracy_score
from sklearn.neighbors import KNeighborsClassifier

# Step 2: Define hyperparameters and their values to search over
param_grid = {
    'n_neighbors': [3, 5, 7],        # Number of neighbors to consider
    'weights': ['uniform', 'distance'],  # Weight function used in prediction
    'p': [1, 2],                     # Power parameter for the Minkowski distance metric
}

# Step 3: Create an instance of GridSearchCV for KNN
knn = KNeighborsClassifier()
grid_search = GridSearchCV(knn, param_grid, cv=5)

# Step 4: Fit the GridSearchCV instance on the training data and measure time
start_time = time.time()
grid_search.fit(X_train, y_train)
end_time = time.time()
total_time = end_time - start_time

# Step 5: Print the best hyperparameters and corresponding accuracy score
best_params = grid_search.best_params_
best_accuracy = round(grid_search.best_score_ * 100, 4)

print("\033[1m-----------------------------------------------------------\033[0m")
print("Best Hyperparameters:", best_params)
print("Best Accuracy Score :", best_accuracy, "%")
print("Total Time Taken    :", round(total_time, 4), "seconds")
print("\033[1m-----------------------------------------------------------\033[0m")

# Refit the model on the full training data using the best hyperparameters and measure time
best_knn = grid_search.best_estimator_
start_time = time.time()
best_knn.fit(X_train, y_train)

# Evaluate the model on the test data and measure time
y_pred_best = best_knn.predict(X_test)
best_knn_accuracy = round(accuracy_score(y_pred_best, y_test) * 100, 4)
end_time = time.time()
total_time = end_time - start_time

print("Best Model Training Accuracy:", round(best_knn.score(X_train, y_train) * 100, 4), "%")
print("Best Model Test Accuracy    :", best_knn_accuracy, "%")
print("Total Time Taken for Training and Prediction:", round(total_time, 4), "seconds")
print("\033[1m-----------------------------------------------------------\033[0m")
print("Best Model Classification Report: \n", classification_report(y_test, y_pred_best))
print("\033[1m-----------------------------------------------------------\033[0m")
```

```
-----------------------------------------------------------
Best Hyperparameters: {'n_neighbors': 7, 'p': 1, 'weights': 'uniform'}
Best Accuracy Score : 96.825 %
Total Time Taken    : 2.5094 seconds
-----------------------------------------------------------
Best Model Training Accuracy: 97.0375 %
Best Model Test Accuracy    : 97.0 %
Total Time Taken for Training and Prediction: 0.0789 seconds
-----------------------------------------------------------
Best Model Classification Report:
               precision    recall  f1-score   support

           0       0.50      0.07      0.12        15
           1       0.97      1.00      0.99      1935
           2       0.50      0.23      0.32        13
           3       0.75      0.15      0.25        20
           4       0.00      0.00      0.00         6
           5       0.00      0.00      0.00        11

    accuracy                           0.97      2000
   macro avg       0.45      0.24      0.28      2000
weighted avg       0.96      0.97      0.96      2000
```

```
------------------------------------------------------------
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control
```

Confusion Matrix

```
from sklearn.metrics import confusion_matrix

# Assuming y_test and y_pred_best are true labels and best predicted labels, respectively
cm = confusion_matrix(y_test, y_pred_best)
# Assuming classes is a list containing the class names in the order of the confusion matrix
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
classes = ["0","1", "2", "3","4","5"]
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)

fmt = '.2f'  # Format to display the numerical values inside the matrix
thresh = cm.max() / 2.
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

plt.ylabel('True Label')
plt.xlabel('Best Predicted Label')
plt.tight_layout()
plt.show()
```



Cross-Validation

```
#  Create the KNN model with the best parameters
best_params = {'n_neighbors': 7, 'p': 1, 'weights': 'distance'}
model = KNeighborsClassifier(n_neighbors=best_params['n_neighbors'],
```

```
                            p=best_params['p'],
                            weights=best_params['weights'])

# Perform cross-validation and get predictions
num_folds = 5
predicted_labels = cross_val_predict(model, X_train, y_train, cv=num_folds)

# Calculate evaluation metrics
accuracy = accuracy_score(y_train, predicted_labels)
precision = precision_score(y_train, predicted_labels, average='weighted')
recall = recall_score(y_train, predicted_labels, average='weighted')
f1 = f1_score(y_train, predicted_labels, average='weighted')
conf_matrix = confusion_matrix(y_train, predicted_labels)

print("\033[1m-----------------------------------------------------------\033[0m")
print(f"Cross-validated Accuracy: {accuracy}")
print(f"Cross-validated Precision: {precision}")
print(f"Cross-validated Recall: {recall}")
print(f"Cross-validated F1-score: {f1}")
print("\033[1m-----------------------------------------------------------\033[0m")
print("Confusion Matrix:")
print(conf_matrix)

# Train the model on the entire training set and test it on the test set
model.fit(X_train, y_train)
test_predictions = model.predict(X_test)

# Evaluate the model on the test set
test_accuracy = accuracy_score(y_test, test_predictions)
test_precision = precision_score(y_test, test_predictions, average='weighted')
test_recall = recall_score(y_test, test_predictions, average='weighted')
test_f1 = f1_score(y_test, test_predictions, average='weighted')
test_conf_matrix = confusion_matrix(y_test, test_predictions)

print("\033[1m-----------------------------------------------------------\033[0m")
print(f"Test Accuracy: {test_accuracy}")
print(f"Test Precision: {test_precision}")
print(f"Test Recall: {test_recall}")
print(f"Test F1-score: {test_f1}")
print("\033[1m-----------------------------------------------------------\033[0m")
print("Test Confusion Matrix:")
print(test_conf_matrix)
print("\033[1m-----------------------------------------------------------\033[0m")
```

```
-----------------------------------------------------------
Cross-validated Accuracy: 0.96825
Cross-validated Precision: 0.9533522367296019
Cross-validated Recall: 0.96825
Cross-validated F1-score: 0.9575651304644048
-----------------------------------------------------------
Confusion Matrix:
[[   2   92    3    0    0    0]
 [   4 7699    8    5    0    1]
 [   0   42   23    0    0    0]
 [   0   48    5   22    0    0]
 [   0   12    0    0    0    0]
 [   0   34    0    0    0    0]]
-----------------------------------------------------------
Test Accuracy: 0.9695
Test Precision: 0.9539696779063915
Test Recall: 0.9695
Test F1-score: 0.958578213764907
-----------------------------------------------------------
Test Confusion Matrix:
[[   1   14    0    0    0    0]
 [   2 1932    0    1    0    0]
 [   0   10    3    0    0    0]
 [   0   16    1    3    0    0]
 [   0    6    0    0    0    0]
 [   0    9    2    0    0    0]]
-----------------------------------------------------------
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavic

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavic
```

Gradient Boosting

Normal Model Training

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, classification_report

# Assuming you have already defined X_train, y_train, X_test, and y_test

# Create Gradient Boosting Classifier
gb_classifier = GradientBoostingClassifier()

# Train the model
gb_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred_gb = gb_classifier.predict(X_test)

# Calculate training accuracy
gb_train_accuracy = round(gb_classifier.score(X_train, y_train) * 100, 4)

# Calculate accuracy score on the test set
gb_accuracy = round(accuracy_score(y_pred_gb, y_test) * 100, 4)

print("Training Accuracy    : {:.4f}%".format(gb_train_accuracy))
print("Model Accuracy Score : {:.4f}%".format(gb_accuracy))
print("\033[1m-------------------------------------------------------\033[0m")
print("Classification_Report: \n", classification_report(y_test, y_pred_gb))
print("\033[1m-------------------------------------------------------\033[0m")
```

```
Training Accuracy    : 99.7750%
Model Accuracy Score : 98.0000%
-------------------------------------------------------
Classification_Report:
               precision    recall  f1-score   support

           0       0.83      1.00      0.91        15
           1       0.99      0.99      0.99      1935
           2       0.64      0.69      0.67        13
           3       0.72      0.65      0.68        20
           4       0.00      0.00      0.00         6
           5       0.00      0.00      0.00        11

    accuracy                           0.98      2000
   macro avg       0.53      0.56      0.54      2000
weighted avg       0.97      0.98      0.98      2000

-------------------------------------------------------
```

Hyperparameter tuning of Gradient Boosting using GridSearchCV

```
import time
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, accuracy_score
from sklearn.ensemble import GradientBoostingClassifier

# Step 2: Define hyperparameters and their values to search over
param_grid = {
    'n_estimators': [50, 100, 200],     # Number of boosting stages to be run
    'learning_rate': [0.1, 0.01, 0.001], # Step size at each iteration
    'max_depth': [3, 5, 7]               # Maximum depth of the individual trees
}

# Step 3: Create an instance of GridSearchCV
gbc = GradientBoostingClassifier()
grid_search = GridSearchCV(gbc, param_grid, cv=5)

# Step 4: Fit the GridSearchCV instance on the training data and measure time
start_time = time.time()
grid_search.fit(X_train, y_train)
end_time = time.time()
```

```
total_time = end_time - start_time

# Step 5: Print the best hyperparameters and corresponding accuracy score
best_params = grid_search.best_params_
best_accuracy = round(grid_search.best_score_ * 100, 4)

print("Best Hyperparameters:", best_params)
print("Best Accuracy Score :", best_accuracy, "%")
print("Total Time Taken    :", round(total_time, 4), "seconds")
print("\033[1m-----------------------------------------------------------\033[0m")

# Refit the model on the full training data using the best hyperparameters and measure time
best_gbc = grid_search.best_estimator_
start_time = time.time()
best_gbc.fit(X_train, y_train)


# Evaluate the model on the test data and measure time
y_pred_best = best_gbc.predict(X_test)
best_gbc_accuracy = round(accuracy_score(y_pred_best, y_test) * 100, 4)
end_time = time.time()
total_time = end_time - start_time

print("Best Model Training Accuracy:", round(best_gbc.score(X_train, y_train) * 100, 4), "%")
print("Best Model Test Accuracy    :", best_gbc_accuracy, "%")
print("Total Time Taken for Training and Prediction:", round(total_time, 2), "seconds")
print("\033[1m-----------------------------------------------------------\033[0m")
print("Best Model Classification Report: \n", classification_report(y_test, y_pred_best))
print("\033[1m-----------------------------------------------------------\033[0m")
```

```
Best Hyperparameters: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 100}
Best Accuracy Score : 98.4875 %
Total Time Taken    : 1585.6182 seconds
-----------------------------------------------------------
Best Model Training Accuracy: 99.8375 %
Best Model Test Accuracy    : 98.1 %
Total Time Taken for Training and Prediction: 8.12 seconds
-----------------------------------------------------------
Best Model Classification Report:
               precision    recall  f1-score   support

           0       0.83      1.00      0.91        15
           1       0.99      0.99      0.99      1935
           2       0.64      0.69      0.67        13
           3       0.68      0.65      0.67        20
           4       0.00      0.00      0.00         6
           5       0.00      0.00      0.00        11

    accuracy                           0.98      2000
   macro avg       0.52      0.56      0.54      2000
weighted avg       0.97      0.98      0.98      2000

-----------------------------------------------------------
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control
```

Cross-Validation

```
from sklearn.model_selection import cross_val_predict, train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

# Create the Gradient Boosting model with the best parameters
best_params = {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 200}
model = GradientBoostingClassifier(learning_rate=best_params['learning_rate'],
                                   max_depth=best_params['max_depth'],
```

```
                                n_estimators=best_params['n_estimators'])

# Perform cross-validation and get predictions
num_folds = 5
predicted_labels = cross_val_predict(model, X_train, y_train, cv=num_folds)

# Calculate evaluation metrics
accuracy = accuracy_score(y_train, predicted_labels)
precision = precision_score(y_train, predicted_labels, average='weighted')
recall = recall_score(y_train, predicted_labels, average='weighted')
f1 = f1_score(y_train, predicted_labels, average='weighted')
conf_matrix = confusion_matrix(y_train, predicted_labels)

print("\033[1m-------------------------------------------------------\033[0m")
print(f"Cross-validated Accuracy: {accuracy}")
print(f"Cross-validated Precision: {precision}")
print(f"Cross-validated Recall: {recall}")
print(f"Cross-validated F1-score: {f1}")
print("Confusion Matrix:")
print(conf_matrix)

# Train the model on the entire training set and test it on the test set
model.fit(X_train, y_train)
test_predictions = model.predict(X_test)

# Evaluate the model on the test set
test_accuracy = accuracy_score(y_test, test_predictions)
test_precision = precision_score(y_test, test_predictions, average='weighted')
test_recall = recall_score(y_test, test_predictions, average='weighted')
test_f1 = f1_score(y_test, test_predictions, average='weighted')
test_conf_matrix = confusion_matrix(y_test, test_predictions)

print("\033[1m-------------------------------------------------------\033[0m")
print(f"Test Accuracy: {test_accuracy}")
print(f"Test Precision: {test_precision}")
print(f"Test Recall: {test_recall}")
print(f"Test F1-score: {test_f1}")
print("Test Confusion Matrix:")
print(test_conf_matrix)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavic

-------------------------------------------------------
Cross-validated Accuracy: 0.98275
Cross-validated Precision: 0.9770383556532828
Cross-validated Recall: 0.98275
Cross-validated F1-score: 0.9794714773331553
Confusion Matrix:
[[  88    7    1    1    0    0]
 [   3 7696    7    4    0    7]
 [   1   29   32    2    0    1]
 [   1   24    4   46    0    0]
 [   0   12    0    0    0    0]
 [   0   34    0    0    0    0]]
-------------------------------------------------------
Test Accuracy: 0.981
Test Precision: 0.9733978632478631
Test Recall: 0.981
Test F1-score: 0.977102140363193
Test Confusion Matrix:
[[  15    0    0    0    0    0]
 [   1 1926    2    4    0    2]
 [   0    4    8    1    0    0]
 [   2    4    1   13    0    0]
 [   0    6    0    0    0    0]
 [   0   10    1    0    0    0]]
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavic
```

Anomaly Detection

```
from sklearn.ensemble import IsolationForest

# Assuming you have already defined X_train and X_test for anomaly detection

# Create Isolation Forest model
isolation_forest = IsolationForest()

# Fit the model on the training data
isolation_forest.fit(X_train)

# Make predictions on the test set (-1: anomaly, 1: normal)
y_pred_anomaly = isolation_forest.predict(X_test)

# Convert predictions to binary format (0: anomaly, 1: normal)
y_pred_anomaly_binary = [1 if pred == 1 else 0 for pred in y_pred_anomaly]

# Calculate accuracy score for anomaly detection
anomaly_accuracy = round(accuracy_score(y_pred_anomaly_binary, y_test) * 100, 4)

print("Anomaly Detection Accuracy: {}%".format(anomaly_accuracy))
```

```
Anomaly Detection Accuracy: 80.75%
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# Assuming you have already defined X_train, y_train, X_test, and y_test

# Create Logistic Regression Classifier
log_reg_classifier = LogisticRegression()

# Train the model
log_reg_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred_log_reg = log_reg_classifier.predict(X_test)

# Calculate training accuracy
log_reg_train_accuracy = round(log_reg_classifier.score(X_train, y_train) * 100, 4)

# Calculate accuracy score on the test set
log_reg_accuracy = round(accuracy_score(y_pred_log_reg, y_test) * 100, 4)

print("Training Accuracy    :", log_reg_train_accuracy, "%")
print("Model Accuracy Score :", log_reg_accuracy, "%")
print("\033[1m-------------------------------------------------------\033[0m")
print("Classification_Report: \n", classification_report(y_test, y_pred_log_reg))
print("\033[1m-------------------------------------------------------\033[0m")
```

```
Training Accuracy    : 96.6625 %
Model Accuracy Score : 96.8 %
-------------------------------------------------------
Classification_Report:
               precision    recall  f1-score   support

           0       0.00      0.00      0.00        15
           1       0.97      1.00      0.98      1935
           2       0.33      0.15      0.21        13
           3       0.75      0.15      0.25        20
           4       0.00      0.00      0.00         6
           5       0.00      0.00      0.00        11

    accuracy                           0.97      2000
   macro avg       0.34      0.22      0.24      2000
weighted avg       0.95      0.97      0.96      2000

-------------------------------------------------------
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control
```

GridsearchCV

```
from sklearn.linear_model import LogisticRegression

# Define hyperparameters and their values to search over
param_grid = {
    'C': [0.1, 1, 10],              # Regularization parameter
    'penalty': ['l1', 'l2'],        # Regularization type ('l1' or 'l2')
    'solver': ['liblinear']         # Optimization algorithm
}

# Create an instance of GridSearchCV
logreg = LogisticRegression()
grid_search = GridSearchCV(logreg, param_grid, cv=5)

# Fit the GridSearchCV instance on the training data and measure time
start_time = time.time()
grid_search.fit(X_train, y_train)
end_time = time.time()
total_time = end_time - start_time

# Print the best hyperparameters and corresponding accuracy score
best_params = grid_search.best_params_
best_accuracy = round(grid_search.best_score_ * 100, 2)

print("Best Hyperparameters:", best_params)
print("Best Accuracy Score :", best_accuracy, "%")
print("Total Time Taken    :", round(total_time, 2), "seconds")
print("\033[1m--------------------------------------------------------\033[0m")

# Refit the model on the full training data using the best hyperparameters and measure time
best_logreg = grid_search.best_estimator_
start_time = time.time()
best_logreg.fit(X_train, y_train)


# Evaluate the model on the test data and measure time
y_pred_best = best_logreg.predict(X_test)
best_logreg_accuracy = round(accuracy_score(y_pred_best, y_test) * 100, 4)
end_time = time.time()
total_time = end_time - start_time

print("\033[1m--------------------------------------------------------\033[0m")
print("Best Model Training Accuracy:", round(best_logreg.score(X_train, y_train) * 100, 4), "%")
print("Best Model Test Accuracy    :", best_logreg_accuracy, "%")
print("Total Time Taken for Training and Prediction:", round(total_time, 2), "seconds")
print("\033[1m--------------------------------------------------------\033[0m")
print("Best Model Classification Report: \n", classification_report(y_test, y_pred_best))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning:

Liblinear failed to converge, increase the number of iterations.

/usr/local/lib/python3.10/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning:

Liblinear failed to converge, increase the number of iterations.

/usr/local/lib/python3.10/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning:
```

```
Liblinear failed to converge, increase the number of iterations.
/usr/local/lib/python3.10/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning:
Liblinear failed to converge, increase the number of iterations.
/usr/local/lib/python3.10/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning:
Liblinear failed to converge, increase the number of iterations.
/usr/local/lib/python3.10/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning:
Liblinear failed to converge, increase the number of iterations.
/usr/local/lib/python3.10/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning:
Liblinear failed to converge, increase the number of iterations.
/usr/local/lib/python3.10/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning:
Liblinear failed to converge, increase the number of iterations.
/usr/local/lib/python3.10/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning:
Liblinear failed to converge, increase the number of iterations.
/usr/local/lib/python3.10/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning:
Liblinear failed to converge, increase the number of iterations.
/usr/local/lib/python3.10/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning:
Liblinear failed to converge, increase the number of iterations.
/usr/local/lib/python3.10/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning:
Liblinear failed to converge, increase the number of iterations.
/usr/local/lib/python3.10/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning:
Liblinear failed to converge, increase the number of iterations.
/usr/local/lib/python3.10/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning:
Liblinear failed to converge, increase the number of iterations.
/usr/local/lib/python3.10/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning:
```

Cross-Validation

```
# Create the Logistic Regression model with the best parameters
best_params = {'C': 0.1, 'penalty': 'l1', 'solver': 'liblinear'}
model = LogisticRegression(C=best_params['C'], penalty=best_params['penalty'], solver=best_params['solver'], multi_class='auto')

#Perform cross-validation and get predictions
num_folds = 5
predicted_labels = cross_val_predict(model, X_train, y_train, cv=num_folds)

# Calculate evaluation metrics
accuracy = accuracy_score(y_train, predicted_labels)
precision = precision_score(y_train, predicted_labels, average='weighted')
recall = recall_score(y_train, predicted_labels, average='weighted')
f1 = f1_score(y_train, predicted_labels, average='weighted')
conf_matrix = confusion_matrix(y_train, predicted_labels)

print("\033[1m--------------------------------------------------------\033[0m")
print(f"Cross-validated Accuracy: {accuracy}")
print(f"Cross-validated Precision: {precision}")
print(f"Cross-validated Recall: {recall}")
print(f"Cross-validated F1-score: {f1}")
print("\033[1m--------------------------------------------------------\033[0m")
print("Confusion Matrix:")
print(conf_matrix)

# Train the model on the entire training set and test it on the test set
model.fit(X_train, y_train)
test_predictions = model.predict(X_test)

# Evaluate the model on the test set
```

```
test_accuracy = accuracy_score(y_test, test_predictions)
test_precision = precision_score(y_test, test_predictions, average='weighted')
test_recall = recall_score(y_test, test_predictions, average='weighted')
test_f1 = f1_score(y_test, test_predictions, average='weighted')
test_conf_matrix = confusion_matrix(y_test, test_predictions)

print("\033[1m-------------------------------------------------------\033[0m")
print(f"Test Accuracy: {test_accuracy}")
print(f"Test Precision: {test_precision}")
print(f"Test Recall: {test_recall}")
print(f"Test F1-score: {test_f1}")
print("\033[1m-------------------------------------------------------\033[0m")
print("Test Confusion Matrix:")
print(test_conf_matrix)
print("\033[1m-------------------------------------------------------\033[0m")
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning:

Liblinear failed to converge, increase the number of iterations.

/usr/local/lib/python3.10/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning:

Liblinear failed to converge, increase the number of iterations.

/usr/local/lib/python3.10/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning:

Liblinear failed to converge, increase the number of iterations.

/usr/local/lib/python3.10/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning:

Liblinear failed to converge, increase the number of iterations.

/usr/local/lib/python3.10/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning:

Liblinear failed to converge, increase the number of iterations.

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this beha

-------------------------------------------------------
Cross-validated Accuracy: 0.97275
Cross-validated Precision: 0.9627570498229642
Cross-validated Recall: 0.97275
Cross-validated F1-score: 0.9639994406675094
-------------------------------------------------------
Confusion Matrix:
[[  12   82    3    0    0    0]
 [   3 7709    1    4    0    0]
 [   0   50   15    0    0    0]
 [   1   22    6   46    0    0]
 [   0   12    0    0    0    0]
 [   0   33    1    0    0    0]]
-------------------------------------------------------
Test Accuracy: 0.975
Test Precision: 0.9580835859861884
Test Recall: 0.975
Test F1-score: 0.965658691274159
-------------------------------------------------------
Test Confusion Matrix:
[[   0   15    0    0    0    0]
 [   0 1934    0    1    0    0]
 [   0   11    2    0    0    0]
 [   2    3    1   14    0    0]
 [   0    6    0    0    0    0]
 [   0   10    1    0    0    0]]
-------------------------------------------------------
/usr/local/lib/python3.10/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning:

Liblinear failed to converge, increase the number of iterations.

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:
```

Deep Neural Networks

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import accuracy_score, classification_report

# Assuming you have your data loaded and preprocessed as X_train, y_train, X_test, y_test

# Create the deep learning model
deep_learning_model = Sequential()
deep_learning_model.add(Dense(128, input_dim=X_train.shape[1], activation='relu'))
deep_learning_model.add(Dropout(0.2))
deep_learning_model.add(Dense(64, activation='relu'))
deep_learning_model.add(Dropout(0.2))
deep_learning_model.add(Dense(1, activation='sigmoid'))  # For binary classification

# Compile the model
deep_learning_model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
deep_learning_model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.2)

# Evaluate the model on the test set
y_pred_probs = deep_learning_model.predict(X_test)
y_pred_deep = (y_pred_probs > 0.5).astype(int)  # Round probabilities to get binary predictions
deep_learning_accuracy = round(accuracy_score(y_pred_deep, y_test) * 100, 4)

print("\033[1m-------------------------------------------------------\033[0m")
print("Deep Learning Model Accuracy Score :", deep_learning_accuracy, "%")
print("\033[1m-------------------------------------------------------\033[0m")
print("Classification_Report: \n", classification_report(y_test, y_pred_deep))
```

```
Epoch 1/50
200/200 [==============================] - 2s 6ms/step - loss: -275.3885 - accuracy: 0.9597 - val_loss: -1202.3534 - val_accuracy: 0.
Epoch 2/50
200/200 [==============================] - 1s 5ms/step - loss: -3685.8389 - accuracy: 0.9659 - val_loss: -12144.6172 - val_accuracy:
Epoch 3/50
200/200 [==============================] - 1s 4ms/step - loss: -17570.3984 - accuracy: 0.9659 - val_loss: -41309.7109 - val_accuracy:
Epoch 4/50
200/200 [==============================] - 1s 3ms/step - loss: -47569.1836 - accuracy: 0.9659 - val_loss: -96256.0000 - val_accuracy:
Epoch 5/50
200/200 [==============================] - 1s 3ms/step - loss: -97638.7266 - accuracy: 0.9659 - val_loss: -185212.8438 - val_accuracy
Epoch 6/50
200/200 [==============================] - 1s 3ms/step - loss: -169301.3594 - accuracy: 0.9659 - val_loss: -298615.6250 - val_accurac
Epoch 7/50
200/200 [==============================] - 1s 4ms/step - loss: -267205.1875 - accuracy: 0.9659 - val_loss: -462135.9062 - val_accurac
Epoch 8/50
200/200 [==============================] - 1s 3ms/step - loss: -398569.5625 - accuracy: 0.9659 - val_loss: -659883.7500 - val_accurac
Epoch 9/50
200/200 [==============================] - 1s 3ms/step - loss: -550954.5000 - accuracy: 0.9659 - val_loss: -904563.0625 - val_accurac
Epoch 10/50
200/200 [==============================] - 1s 3ms/step - loss: -750786.8125 - accuracy: 0.9659 - val_loss: -1196824.5000 - val_accura
Epoch 11/50
200/200 [==============================] - 1s 3ms/step - loss: -958913.1875 - accuracy: 0.9659 - val_loss: -1537337.0000 - val_accura
Epoch 12/50
200/200 [==============================] - 1s 3ms/step - loss: -1263566.1250 - accuracy: 0.9659 - val_loss: -1913795.3750 - val_accur
Epoch 13/50
200/200 [==============================] - 1s 3ms/step - loss: -1528464.0000 - accuracy: 0.9659 - val_loss: -2346667.0000 - val_accur
Epoch 14/50
200/200 [==============================] - 1s 3ms/step - loss: -1840410.5000 - accuracy: 0.9659 - val_loss: -2834119.7500 - val_accur
Epoch 15/50
200/200 [==============================] - 1s 3ms/step - loss: -2235456.0000 - accuracy: 0.9659 - val_loss: -3393285.0000 - val_accur
Epoch 16/50
200/200 [==============================] - 1s 3ms/step - loss: -2624466.0000 - accuracy: 0.9659 - val_loss: -4003441.2500 - val_accur
Epoch 17/50
200/200 [==============================] - 1s 3ms/step - loss: -3105222.5000 - accuracy: 0.9659 - val_loss: -4651995.0000 - val_accur
Epoch 18/50
200/200 [==============================] - 1s 3ms/step - loss: -3612704.7500 - accuracy: 0.9659 - val_loss: -5383709.0000 - val_accur
Epoch 19/50
200/200 [==============================] - 1s 4ms/step - loss: -4042292.2500 - accuracy: 0.9659 - val_loss: -6132049.5000 - val_accur
Epoch 20/50
200/200 [==============================] - 1s 5ms/step - loss: -4621231.5000 - accuracy: 0.9659 - val_loss: -6921214.5000 - val_accur
Epoch 21/50
200/200 [==============================] - 1s 5ms/step - loss: -5194642.0000 - accuracy: 0.9659 - val_loss: -7840853.0000 - val_accur
Epoch 22/50
200/200 [==============================] - 1s 4ms/step - loss: -5870599.5000 - accuracy: 0.9659 - val_loss: -8811231.0000 - val_accur
Epoch 23/50
200/200 [==============================] - 1s 3ms/step - loss: -6791923.0000 - accuracy: 0.9659 - val_loss: -9807318.0000 - val_accur
Epoch 24/50
200/200 [==============================] - 1s 3ms/step - loss: -7287859.0000 - accuracy: 0.9659 - val_loss: -10958721.0000 - val_accu
Epoch 25/50
200/200 [==============================] - 1s 3ms/step - loss: -8286449.5000 - accuracy: 0.9659 - val_loss: -12109709.0000 - val_accu
```

```
Epoch 26/50
200/200 [==============================] - 1s 3ms/step - loss: -9173921.0000 - accuracy: 0.9659 - val_loss: -13440883.0000 - val_accu
Epoch 27/50
200/200 [==============================] - 1s 4ms/step - loss: -9938760.0000 - accuracy: 0.9659 - val_loss: -14679103.0000 - val_accu
Epoch 28/50
200/200 [==============================] - 1s 3ms/step - loss: -10840567.0000 - accuracy: 0.9659 - val_loss: -16060266.0000 - val_acc
```

```
import numpy as np
from sklearn.model_selection import GridSearchCV
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier

# Define a function to create the Keras model
def create_model(learning_rate=0.001, dense_units_1=128, dense_units_2=64, dropout_rate=0.2):
    deep_learning_model = Sequential()
    deep_learning_model.add(Dense(dense_units_1, input_dim=X_train.shape[1], activation='relu'))
    deep_learning_model.add(Dropout(dropout_rate))
    deep_learning_model.add(Dense(dense_units_2, activation='relu'))
    deep_learning_model.add(Dropout(dropout_rate))
    deep_learning_model.add(Dense(1, activation='sigmoid'))

    deep_learning_model.compile(optimizer=Adam(learning_rate=learning_rate),
                                loss='binary_crossentropy',
                                metrics=['accuracy'])
    return deep_learning_model

start_time = time.time()

# Create KerasClassifier
keras_classifier = KerasClassifier(build_fn=create_model, verbose=0)

# Define hyperparameters to search
param_grid = {
    'learning_rate': [0.001, 0.01],
    'dense_units_1': [64, 128],
    'dense_units_2': [32, 64],
    'dropout_rate': [0.2, 0.3]
}

# Create GridSearchCV
grid_search = GridSearchCV(estimator=keras_classifier,
                           param_grid=param_grid,
                           cv=3,
                           verbose=1,
                           n_jobs=-1)

# Perform the grid search on X_train and y_train
grid_result = grid_search.fit(X_train, y_train)

# Get the best parameters and model
best_params = grid_result.best_params_
best_model = grid_result.best_estimator_

# Evaluate the best model on the test set
start_time1 = time.time()
y_pred_probs = best_model.predict(X_test)
y_pred_deep = (y_pred_probs > 0.5).astype(int)
deep_learning_accuracy = round(accuracy_score(y_pred_deep, y_test) * 100, 4)
end_time = time.time()
total_time = end_time - start_time
total_time_pre = end_time - start_time1
print("Total Time Taken:", round(total_time, 2), "seconds")

print("\033[1m-----------------------------------------------------------\033[0m")
print("Total Time Taken for Training and Prediction:", round(total_time_pre, 2), "seconds")
print("Best Hyperparameters:", best_params)
print("Deep Learning Model Accuracy Score:", deep_learning_accuracy, "%")
print("\033[1m-----------------------------------------------------------\033[0m")
print("Classification Report:\n", classification_report(y_test, y_pred_deep))
```

```
Fitting 3 folds for each of 16 candidates, totalling 48 fits
<ipython-input-72-b18205600e9d>:22: DeprecationWarning:

KerasClassifier is deprecated, use Sci-Keras (https://github.com/adriangb/scikeras) instead. See https://www.adriangb.com/scikeras/stabl

63/63 [==============================] - 0s 1ms/step
Total Time Taken: 90.27 seconds
```

```
----------------------------------------------------------
Total Time Taken for Training and Prediction: 0.23 seconds
Best Hyperparameters: {'dense_units_1': 64, 'dense_units_2': 32, 'dropout_rate': 0.2, 'learning_rate': 0.001}
Deep Learning Model Accuracy Score: 96.75 %
----------------------------------------------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        15
           1       0.97      1.00      0.98      1935
           2       0.00      0.00      0.00        13
           3       0.00      0.00      0.00        20
           4       0.00      0.00      0.00         6
           5       0.00      0.00      0.00        11

    accuracy                           0.97      2000
   macro avg       0.16      0.17      0.16      2000
weighted avg       0.94      0.97      0.95      2000

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import Adam

# Define the neural network architecture and parameters
dense_units_1 = 64
dense_units_2 = 32
dropout_rate = 0.2
learning_rate = 0.001
num_classes = len(np.unique(y))  # Number of unique classes in the target variable

# Split the data into training and testing sets
#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create the neural network model
model = Sequential()
model.add(Dense(dense_units_1, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dropout(dropout_rate))
model.add(Dense(dense_units_2, activation='relu'))
model.add(Dropout(dropout_rate))
model.add(Dense(num_classes, activation='softmax'))

# Compile the model with the given learning rate
optimizer = Adam(learning_rate=learning_rate)
model.compile(loss='sparse_categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])

# Fit the model with cross-validation
num_folds = 5
history = model.fit(X_train, y_train, batch_size=32, epochs=50, validation_split=1/num_folds, verbose=0)

# Evaluate the model on the training set
predicted_labels = np.argmax(model.predict(X_train), axis=-1)
accuracy = accuracy_score(y_train, predicted_labels)
precision = precision_score(y_train, predicted_labels, average='weighted')
recall = recall_score(y_train, predicted_labels, average='weighted')
f1 = f1_score(y_train, predicted_labels, average='weighted')
conf_matrix = confusion_matrix(y_train, predicted_labels)

print(f"Cross-validated Accuracy: {accuracy}")
print(f"Cross-validated Precision: {precision}")
print(f"Cross-validated Recall: {recall}")
print(f"Cross-validated F1-score: {f1}")
print("Confusion Matrix:")
print(conf_matrix)
```

```
# Evaluate the model on the test set
test_predictions = np.argmax(model.predict(X_test), axis=-1)
test_accuracy = accuracy_score(y_test, test_predictions)
test_precision = precision_score(y_test, test_predictions, average='weighted')
test_recall = recall_score(y_test, test_predictions, average='weighted')
test_f1 = f1_score(y_test, test_predictions, average='weighted')
test_conf_matrix = confusion_matrix(y_test, test_predictions)

print(f"Test Accuracy: {test_accuracy}")
print(f"Test Precision: {test_precision}")
print(f"Test Recall: {test_recall}")
print(f"Test F1-score: {test_f1}")
print("Test Confusion Matrix:")
print(test_conf_matrix)
```

```
250/250 [==============================] - 1s 2ms/step
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavic

Cross-validated Accuracy: 0.964625
Cross-validated Precision: 0.930501390625
Cross-validated Recall: 0.964625
Cross-validated F1-score: 0.9472559807851372
Confusion Matrix:
[[   0   97    0    0    0    0]
 [   0 7717    0    0    0    0]
 [   0   65    0    0    0    0]
 [   0   75    0    0    0    0]
 [   0   12    0    0    0    0]
 [   0   34    0    0    0    0]]
63/63 [==============================] - 0s 2ms/step
Test Accuracy: 0.9675
Test Precision: 0.9360562499999999
Test Recall: 0.9675
Test F1-score: 0.9515184243964423
Test Confusion Matrix:
[[   0   15    0    0    0    0]
 [   0 1935    0    0    0    0]
 [   0   13    0    0    0    0]
 [   0   20    0    0    0    0]
 [   0    6    0    0    0    0]
 [   0   11    0    0    0    0]]
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavic
```