Development of a Condition Assessment Rating System and Prediction Model for Railway Tracks

Bharath Rajendir Rajendran

A Thesis

In the Department of

Building, Civil, and Environmental Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of

Master of Applied Science (Civil Engineering)

at Concordia University

Montréal, Québec, Canada

October 2024

© Bharath Rajendir Rajendran, 2024

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By: **Bharath Rajendir Rajendran**

Entitled: **Development of a Condition Assessment Rating System and Prediction Model for Railway Tracks**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Civil Engineering)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
Dr. Jong Won Ma

_____ Examiner
Dr. Osama Moselhi

_____ Examiner
Dr. Jong Won Ma

_____ Supervisor
Dr. Rebecca Dziedzic

Approved by _____
Dr. Po Han Chen, Graduate Program Director

October 15, 2024

_____
Dr. Mourad Debbabi, Dean of Gina Cody School of Engineering and Computer Science

Abstract

Development of a Condition Assessment Rating System and Prediction Model for Railway Tracks

Bharath Rajendir Rajendran

Canada has an extensive rail network spanning 45,000 kilometres. The railway system plays a crucial role in serving almost every sector of the Canadian economy. Primarily, it transports freight to and from the U.S. and global markets through coastal ports. However, failures in the railway infrastructure can have severe safety and financial consequences. In 2023, 43.13% of main-track derailments were attributed to track defects, according to the Transportation Safety Board of Canada. These defects, including issues with track geometry and component failures, underline the need for better track condition monitoring and maintenance to prevent derailments. This research aims to address this need by developing a comprehensive rating system for evaluating the condition of ties and rail fastening components and machine learning models to predict future track conditions. While traditional condition assessment ratings have relied on subjective evaluations and considered components separately, this study proposes a Tie and Rail Fastening system that evaluates the condition of ties, tie plates, and spikes. Domain expertise was incorporated through the Analytic Hierarchy Process (AHP) to prioritize the importance of various defects. The resulting weighting system provides a more detailed and integrated approach compared to existing rating methods, which primarily focus on crack size. Machine learning models, including Random Forest, XGBoost, and Cat Boost, were employed to predict future conditions, such as defect tags, amplitude, and length. These models achieved a 95% accuracy for detecting defect tags and a 75% accuracy when predicting defect tags based on predicted amplitude. On the one hand, the proposed tie and rail fastening rating system can improve the prioritization of future rail maintenance works. On the other hand, the proposed machine learning models can improve the planning of future maintenance by offering better tools for monitoring and predicting track conditions.

Acknowledgements

I express my deepest gratitude to Professor Rebecca Dziedzic, my kind and dedicated supervisor, for her unwavering support and guidance throughout my master's studies. Completing this research and writing this thesis would not have been possible without her expertise and insight at every stage of the process. Her extensive knowledge, patience, and invaluable advice have guided me through this challenging yet adventurous journey. I would also like to acknowledge Concordia University for funding the project and providing technical support during the research. My sincere thanks go to Richard Fox-Ivey from Pavemetrics Inc. for providing the dataset for this study and offering valuable support. I also extend my gratitude to my colleagues Daniel Blais and Matthew Krech from Transport Canada for their support during my internship and to all the railroad professionals across North America and other countries who actively participated in the survey and contributed valuable insights to developing the condition assessment framework. Lastly, I must thank my parents for their unconditional encouragement and support throughout this journey and express my gratitude to God for their blessings in overcoming the hardships. This thesis is dedicated to my Parents and my mentor, Dr. Rebecca Dziedzic.

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

1.1. Background

Canada has an extensive rail network spanning 45,000 kilometres of track (Transport Canada 2023). The railway system plays a crucial role in serving almost every sector of the Canadian economy. Primarily, it transports freight to and from the U.S. and global markets through coastal ports. Additionally, there are numerous passenger lines operating across Canada. Thus, failures in these networks can have serious consequences for human safety, as well as high costs. According to the Transportation Safety Board of Canada (TSB 2023), a significant portion of main-track derailments in 2023, specifically 43.13% (22 out of 51 derailments), were attributed to track defects. These track defects encompassed issues like track geometry, broken rails, and other track components, and this emphasizes the importance of addressing and maintaining track infrastructure to ensure railway safety and prevent derailments. According to the American Railroads Association, the US freight rail network transports one-third of all exports from the United States and around 40% of all long-distance freight (Black 2022). Track defects are one of the main reasons for train accidents in the US. For example, in 2012, 33.03% of 1747 train accidents recorded by the Federal Railroad Administration (FRA) were due to track defects, causing $102.9 million in total reportable damage (Peng, Ouyang, and Somani 2013). The FRA safety compliance classifies defects into red and yellow tags. Red tag defects should be fixed immediately since they violate the FRA standards, and yellow tag defects should be fixed before turning red (RAS 2015). Defects can be identified with track geometry vehicles using visual inspection and technologies like induction and ultrasonic devices (Cannon et al. 2003). Railway tracks can be impacted by a range of defects, which can significantly affect the safety and efficiency of train operations. These defects may encompass cracks, wear and tear, misalignments, and other structural issues that could compromise the integrity of the track system. Regular inspection and maintenance are crucial in identifying and addressing these issues before they escalate into substantial hazards. Additionally, technological advancements have introduced various methods for detecting and mitigating track defects, such as automated track inspections using lasers, Lidar, and drones. In track maintenance, rails may experience breakage or wear, while ties can split, crack, or become severed. Fastenings might be missed, spikes could break, become

loose, or go missing, anchors may fail to hold, and ballast can become fouled or provide poor drainage. Track geometry issues can lead to poor gauge holding or misalignment. Since railway tracks consist of various components, defects may appear differently.

Most current condition assessment systems focus mainly on track geometry and ballast, but they often miss important factors like spikes, tie plates, and the exact location of cracks. While track geometry is necessary, leaving out the condition of spikes and tie plates prevents a full understanding of the track's overall health. For example, good spikes and tie plates can help maintain track integrity, even if the geometry is slightly compromised. Additionally, cracks near these components may pose greater risks than those in other areas. This highlights the need for a more comprehensive rating system that incorporates all these elements to fully assess the track's condition. In condition prediction models, the focus is often on identifying defects quickly without paying enough attention to the details of the defects themselves, like their type, size, and length. While it is helpful to predict tag defects, this does not provide enough information to decide how and when to fix the issues. If prediction models could also tell us more about the defects, such as their severity and how fast they are getting worse, it would help make better maintenance decisions. This shows a gap in current models, which need to go beyond just finding defects and offer more detailed insights into the nature of the problems.

1.2 Objective

The objective of this work is to formulate methodologies for the assessment and prediction of railway track conditions. The specific goals to achieve this aim are as follows:

- Develop a comprehensive condition rating system to systematically evaluate the condition of rail ties and fastenings.
- Develop machine learning models to predict the condition of railway tracks, including defect tags, types, length, and amplitude.

1.3 Thesis Structure

This thesis is organized as follows: Chapter 2 presents a literature review covering the various components of railway tracks and their role in supporting train operations. It reviews the factors contributing to the deterioration of key components like rails, sleepers or ties, ballast, and fastening systems. The literature on the degradation of track geometry is explored, highlighting the impact on overall track performance. Additionally, it examines the railway track's geometry parameters, and the current methods used for condition assessment. The review also includes condition prediction models and evaluates their effectiveness. Finally, it discusses the benefits and limitations of previous methodologies found in the literature. Chapter 3 outlines the proposed methodology for the track's condition rating system and prediction model, which integrates the Analytical Hierarchy Process (AHP) to establish a comprehensive rating system and employs machine learning-based approaches for predicting track condition and defect characteristics. This section also introduces two different case studies to evaluate the proposed strategy. Chapter 4 presents result regarding the rating system for evaluating the condition of railroad ties and fastening components in the railway track and the prediction of defect tags, types, length, and amplitude. In Chapter 5, discussions associated with the results are provided. Finally, conclusions are drawn in Chapter 6.

1.4 Contributions

Two main contributions of the work are:

- The development of a comprehensive condition rating system using the Analytical Hierarchy Process (AHP) to enhance the understanding of how multiple factors like rail ties, fastenings, and crack locations impact track performance. This model provides a more detailed and structured approach to evaluating track components, supporting future research on automated condition assessment. Practically, this model will improve maintenance planning and risk mitigation, allowing for targeted interventions based on specific asset conditions.

- The development of machine learning models that predict the condition of railway tracks and provide detailed predictions of defect characteristics such as tag, type, length, and amplitude. In practice, they will enhance the accuracy of maintenance planning and safety management by forecasting specific defect behaviours, enabling proactive and data-driven decision-making.

Chapter 2. Literature Review

Assessing and predicting track condition is crucial for effective railway maintenance and operational efficiency. While traditional assessment methods can be costly due to manual inspections, advancements in automatic track inspection technologies have made the process more cost-efficient. These technologies allow real-time monitoring and data collection, enabling timely maintenance decisions. To predict track deterioration, various models, including mechanical, statistical, and artificial intelligence approaches, are used. Previous studies (Falamarzi, Moridpour, and Nazem 2019) demonstrate the effectiveness of these models in forecasting maintenance needs. Predictive modelling helps reduce costs, optimize maintenance schedules, and enhance safety by addressing potential issues proactively. A comprehensive approach integrating condition assessment and predictive modelling is essential for effective railway maintenance.

2.1 Railway Track Components

The main components of railway tracks are rails, sleepers or railroad ties, ballasts and fastening systems, as indicated in Figure 1. Rails are the track components arranged in two parallel lines to give trains a stable, continuous, and level surface (Chandra et al. 2013). The flat-bottom rail is the most widely used rail profile worldwide, with a flat bottom. The nonstandard rail type differs from the flat-bottom rail because it has a thicker web to accommodate expansion devices, switches and crossing components. Grooved rail is used in enclosed track systems like roads and yards (Esveld 2001a). Steel rails are used in North American railroads. Based on their mechanical properties, like tensile strength and hardness, there are two types of carbon steel rails and low alloy steel rails (AREMA Manual for Railway Engineering 2022a). Rectangular support for the rails on railroad tracks is known as a railroad tie or sleeper. Ties, typically set perpendicular to the rails, hold the rails upright and maintain the proper gauge while transferring loads to the ballast and subgrade of the track. The individual crosstie receives the load from the rail and transfers it to the ballast. In North American railroads, concrete, timber, engineered composite, and steel ties are used(AREMA Manual for Railway Engineering 2022b).

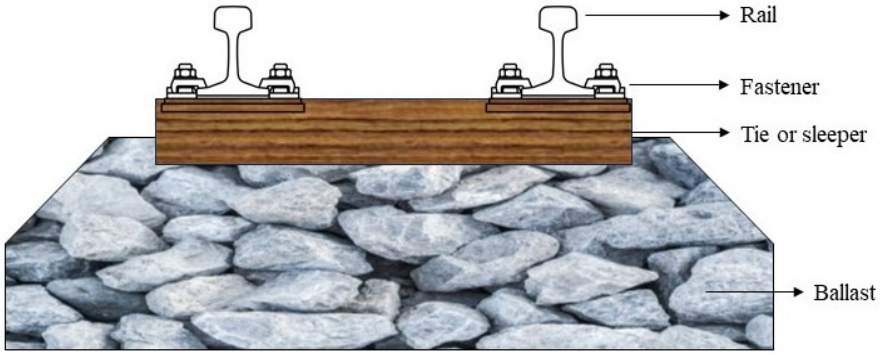


Figure 1: The main components of the railway track.

Rails are structural component of the railway, as they directly encounter the wheel surfaces of the rail vehicles(Zerbst et al. 2009). Rail damage occurs mainly due to the interaction between wheels and rails, which is caused by higher axle loads and train speeds. Over time, the rails tend to wear out due to increased loading cycles. This complex process involves various modes of material degradation and changes in the contact surface. It may result in material removal or displacement, plastic deformation, and phase transformation within or between the contact surface (Enblom 2009). Rail deterioration can be caused by environmental factors such as extreme cold, high temperature, high humidity, rain, and snow. According to (Ma et al. 2018) Rail rollers exposed to low temperatures tend to wear out faster, become harder, have a higher adhesion coefficient, and experience a shift in wear mechanisms from abrasive to adhesive wear with surface cracking. However, extremely low temperatures may somewhat reduce the adhesive wear effect. These findings are crucial in understanding how rail systems and materials perform in cold weather conditions, which can impact maintenance and safety. Therefore, rail is a critical component of railway tracks. It is essential to closely monitor its condition and perform appropriate maintenance to prevent derailments and ensure the overall safety of the track systems.

Railway ties, known as sleepers, and support rails, maintain track geometry and ensure safe and efficient train operations(Yu and Jeong 2012). Tie failure occurs due to the forces generated by the wheels and rails; there can be high stress levels when the rail base meets the tie. In some cases, these stresses can be too much for the tie to handle, causing it to deteriorate and eventually leading to a rail rollover and derailment(Marquis, Muhlanger, and Jeong 2011). Tie failure can be caused by environmental factors such as exposure to wet/dry or freeze-thaw cycles. This can result in splits in the ties, which may spread from one end to another. If rain, ice, or ballast enters the split,

it can widen the gap until the tie is unable to hold the spikes or support the load(Palese et al. 1999)Inspecting the condition of ties is essential to prevent failures that could lead to derailment and weaken the track's substructure, including the ballast and subgrade.

A rail fastening system is a technique for attaching rails to railroad ties or sleepers. Rails and base plates are fastened to railroad ties in the track with the help of rail spikes as indicated in Figure 2, which are substantial nails with an offset head. A rail spike has a flat-edged point and is chisel-shaped; it is driven with the edge perpendicular to the grain, which increases resistance to loosening. The primary purpose is to maintain rail gauge (Hay 1982). Several research studies (M. Dersch et al. 2019), (Gao, McHenry, and Kerchof 2018a) Show the impact of spikes as a crucial factor on ties. For instance, if the spikes are broken or missing, the stress from the train is directly transferred to the tie, and the tie deteriorates. Thus, the spike holds the ties with the rail, gets the load from the train and distributes it to the ties (M. S. Dersch, Khachaturian, and Edwards 2021). Derailment of train (The Transportation Safety Board of Canada 2012)It happened due to broken and missing spikes.

Rail tie plates, as indicated in Figure 2, are used to support the rails and fix the entire rail fastening systems. It always works with anchor bolts or spikes by sustaining the load of a rail track and transferring part of the load to the tie sleepers, with a flat, smooth resting surface to guarantee vertical alignment and hold the rail in the correct gauge for a rail line system(Gao, McHenry, and Kerchof 2018b). The damaging effects caused by tie-plates and the ballast, namely plate cutting and ballast abrasion, are accelerated with faster and higher tonnage trains. This causes crossties to age prematurely and results in high crosstie replacement rates. Wood crossties exhibit some inherent disadvantages. Wood is susceptible to mechanical degradation mainly due to splitting, checking, plate cutting, spike killing, and tamp killing. In addition, wood ties are subjected to harsh environmental conditions that can cause rot and decay (Sonti et al. 1995).

The track sub-structure has three layers: Ballast, Sub-ballast, and subgrade, as indicated in Figure 3. Track ballast, which creates the track bed, is used to support the railroad ties' load, make it easier for water to drain, and control vegetation that could obstruct the track's structure. As the trains pass over the track, ballast keeps it in place. As track ballast, several substances have been employed, including crushed stone, washed gravel, slag, chats, coal cinders, sand, and burnt clay(Solomon 2001). The sub-ballast is the layer of soil or aggregate material placed between the subgrade and

the ballast, which helps improve drainage and stability. The subgrade is the natural or prepared surface on which the railway track is built, and it can be made of soil, rock, or other materials.

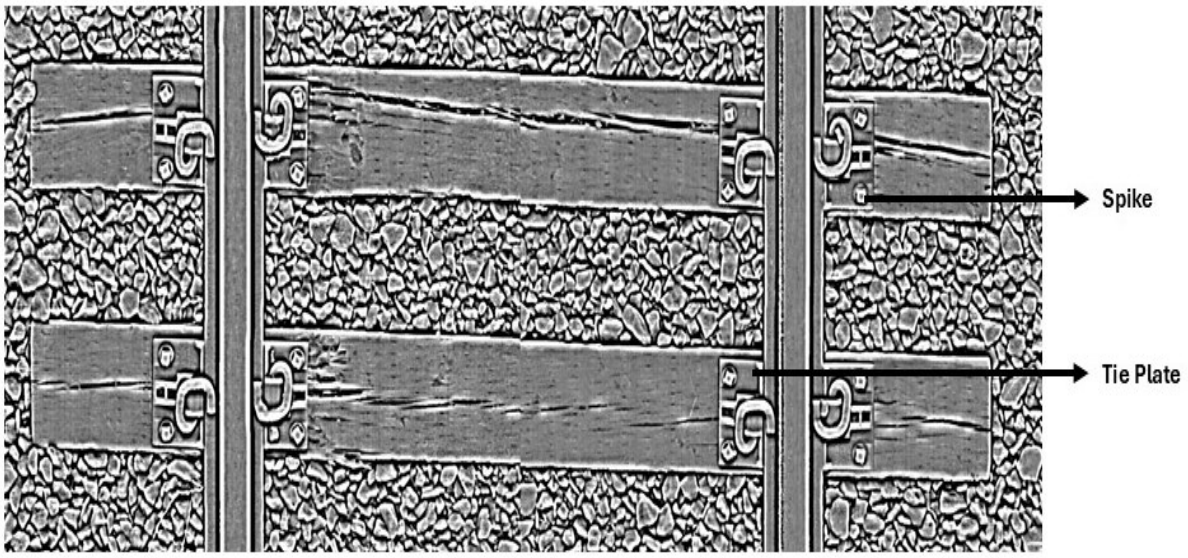


Figure 2: Spike and Tie plate

The ballast bed is critical in functioning ballasted tracks at high speeds. It is subjected to cyclic train loads and contamination intrusion during prolonged operation, which can lead to ballast particle degradation (crushing and abrasion) and bed pollution. These issues can result in track deformation, poor drainage, and reduced bearing capacity(Q. Hu et al. 2023). Railway ballast is typically made up of uniformly graded angular aggregate. As ballast ages, it can become increasingly fouled by various fine materials, which accumulate in the voids of the ballast and decrease shear strength, resiliency, and drainage capability(Indraratna, Su, and Rujikiatkamjorn 2011a). The fouling process can be accelerated when contaminant materials from other sources collect in the intergranular voids. When the ballast becomes fouled, it loses its ability to perform its functions efficiently. If the level of contamination reaches the bottom side of the tie, the track substructure starts to fail (Ionescu 2023). When the ballast is not functioning correctly, the strength of the track structure may be inadequate, compromising track stability. Therefore, it is crucial to monitor the condition of the ballast to ensure safer operation and prevent degradation of the other track components.

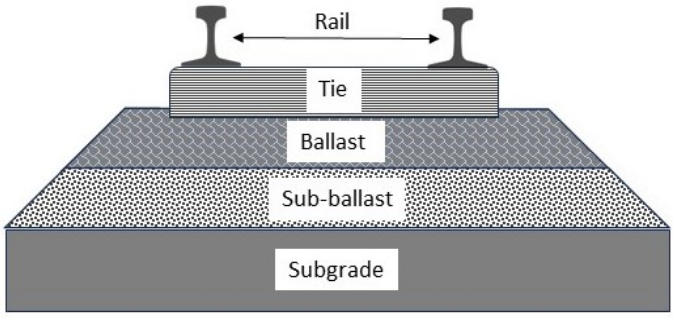


Figure 3: The track sub-structure.

2.2 Track Geometry

Track geometry refers to the precise location of each rail or track center line in space, which includes gauge, twist, longitudinal level, alignment, and cross-level (also known as superelevation or cant), as illustrated in Figure 4. The gauge of a railway track is the distance between the inner sides of the left and right rail heads, measured perpendicular to the track center(Puffert 2000). When the top surfaces of two rails are at different elevations, this is known as a twist(Javad Sadeghi and Askarinejad 2010). Longitudinal level refers to the difference (in millimeters) between a point on the top of the rail in the running plane and the ideal mean line of the longitudinal profile(A. Ramos Andrade and Teixeira 2011). Alignment is the deviation in lateral positions of the left and right rails from a mean trajectory. It is obtained by filtering out wavelengths longer than a given length(Weston et al. 2007). Cross-level is the deviation between the top surfaces of two rails at a specific point along the track(Esveld 2001b).

An ideal railway track should have a correct and uniform gauge. The rails should have perfect cross levels, and in curves, the outer rail should have a proper superelevation to consider the centrifugal force. The alignment should be straight and free of any kinks. In the case of curves, a proper transition should be provided between the straight track and the curve. The gradient should be uniform and as gentle as possible. The change of gradient should be followed by a proper vertical curve to ensure a smooth ride. The track should be resilient and elastic so that it can absorb the shocks and vibrations of running trains. It should also have a good drainage system to maintain its stability and should have good lateral strength to withstand variations in temperature and other factors(Gofran J. Qasim 2019).

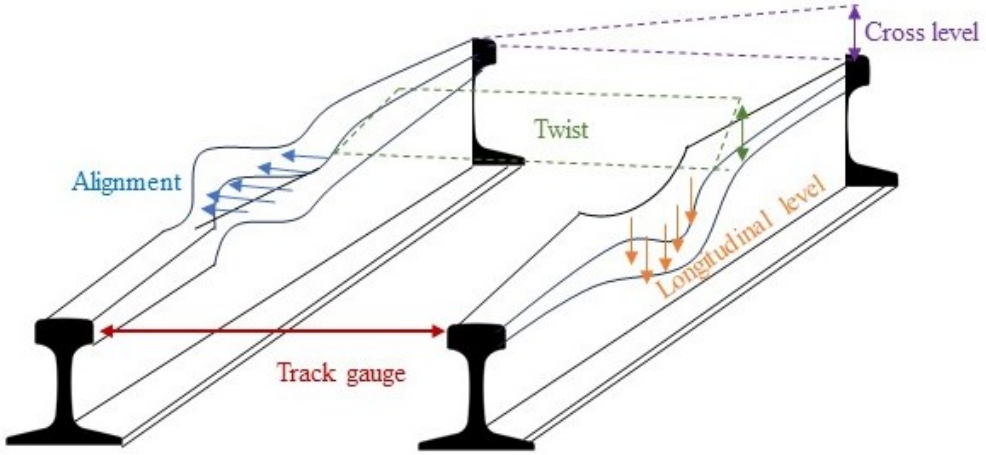


Figure 4: Track geometry parameters.

Railway tracks are designed with curves to navigate around obstacles, create more efficient slopes, and connect important locations. Horizontal curves alter the direction of the track, while vertical curves are placed where two slopes meet, or the slope meets level ground. To ensure a smooth ride on a horizontal curve, the outer rail is raised above the inner rail, a technique called superelevation. It is important to have superelevation in the track to distribute the load evenly on both rails and reduce wear and tear on curves(Chandra and Agarwal 2013).

The geometry of a railway track is a crucial part of any railway system. It directly impacts the performance of the track itself and the behavior of the vehicles that use it (Powell and Gräbe 2017). As the track ages and is used, its geometry can degrade, negatively affecting safety and performance. If the track geometry becomes unacceptable, it can result in derailment, which can have significant consequences such as high costs of operation, economic loss, damage to the railway asset and environment, and even loss of human life. Rectifying poor track geometry is the most expensive part of maintenance(Gustavsson 2015). Table 1 displays the factors that contribute to the degradation of track geometry. Ensuring railway safety requires careful analysis of track geometry defects. Preventive maintenance can be scheduled by identifying when repairs are necessary to reduce the risk of track failures. Regular inspection of track geometry is essential, and maintenance actions should be planned accordingly to maintain an acceptable level of safety.

Table 1 Factors contributing to the railway track geometry degradation

Reference	Factors/Events contributing to the railway track geometry degradation	Geometry defects
(Bing and Gross 1983)	Annual tonnage, Axle load, Train speed, and Ballast type	Alignment, gauge and cross level
(Puzavac, Popović, and Lazarević 2012)	Track stiffness	Alignment
(Guler 2014a)	Traffic loads, Speed, curvature, Gradient, Cross level, Sleeper type (concrete or wooden), Rail type (49.430 or 49.050 kg/m), Rail length, Falling rock, Landslide, Snow and Flood	Twist, Gauge, alignment, cross-level
(Zarembski et al. 2015)	Missing Ballast	Cross-level, gauge, dip, alignment, surface and warp
(C. Hu and Liu 2016)	Train load, Train speed, Track layout, Track class, Time intervals of inspection, Defect length and Amplitude	Surface, cross-level and dip
(Cárdenas-Gallo et al. 2017)	Tonnage, Defect amplitude, Track type, Track class, Speed	Cross-level, Dip and surface
(D. Li 2018)	Axle loads, Train speed, Traffic density and Track subgrade	Alignment, gauge and cross level

2.3 Condition Rating System

A systematic railway maintenance process is crucial for ensuring the safety and reliability of the railway system. However, railway maintenance can be quite costly and typically accounts for a significant portion of the budget. For example, in 2013, the Canadian National Railway Company spent $2.74 billion on primary track maintenance (Scanlan, Hendry, and Martin 2016). According to (Transport Canada 2022) Operating rules, regular inspections, and adherence to track standards are essential for safe and efficient rail transportation. As autonomous track inspection programs advance and railways collect more data, this data can provide valuable insights. According to the (Railway Association of Canada 2022), 42,631 kilometers of freight track are operated. As rail traffic volumes and the associated annual tonnage continue to rise, maintaining consistent and effective track maintenance becomes more challenging. This increase in traffic and axle loading places greater demand on the track infrastructure and its components, making regular safety inspections even more critical. However, emerging technologies present an opportunity for the rail industry to enhance safety, optimize maintenance strategies, and create a more reliable and efficient network (Marquis, Muhlanger, and Jeong 2011), Compounded by diminishing track access times and constrained maintenance budgets, the conventional practice of scheduling large, consolidated zones for maintenance and rehabilitation is now being re-evaluated. According to the (Railway Tie Association 2024), 20 to 22 million rail ties are replaced annually in Canada and the USA. In its place, there is a growing call for a more precise and meticulously referenced analysis of the in-situ condition of rail ties and fastening systems. This shift in approach aims to optimize operational efficiency and prioritize safety, recognizing the imperative of mitigating the risk of accidents and ensuring the overall safety of rail operations.

Condition assessment involves thoroughly evaluating the physical state of an asset to determine its current condition, identifying any existing issues, and prioritizing maintenance or repair needs(Marlow and Burn 2008). In the context of railways, condition assessment involves evaluating the current state of railway infrastructure, including tracks with rail, ties, fastening systems, and ballast, to determine maintenance needs and prioritize interventions. Traditionally, maintenance teams would conduct periodic inspections to identify potential issues before they become major problems. However, this method had limitations: inspections were time-consuming, sometimes missed critical issues, and could not always keep up with the rapid pace of degradation.

Several authors (Xu et al. 2011) (Lasisi and Attoh-Okine 2018) have developed condition rating systems for railway tracks, to assess geometry parameters like gauge, cross-level, left and right surface, and left and right alignment. The Track Quality Index (TQI) evaluates the overall state of a railway track based on factors like geometric defects such as gauge, cross-level, left/right surface, and alignment. (J. M. Sadeghi and Askarinejad 2011)developed a track quality index to assess the track's condition based on human visual inspections. This index is useful for maintenance planning and ensuring safety. However, the condition rating does not consider factors such as the condition of wooden ties, the location of cracks in the ties, and fastening systems. Additionally, since the rating is based on visual inspections by humans, there is a possibility of human error and subjectivity in the assessment of the track's condition.

(Madejski, Janusz 2015) used manual equipment to collect geometry track measurements and developed a condition assessment that included a five-parameter defectiveness. This parameter assesses the geometrical condition of the track by aggregating five parameters, each representing a specific geometrical defect. Each parameter is a ratio of the length when the acceptable limits for the defects exceed the total length of the section. The Indian Railway has developed a method known as the Track Geometry Index (TGI) to assess the geometrical condition of tracks. This method relies on the standard deviation of geometrical defects (Mundrey, J. S 2009). The Swedish National Railway has created a quality index to assess the condition of railway tracks. This index uses the standard deviation of left and right profiles and geometry defects to determine the track's condition. By comparing the current standard deviation with the allowable standard deviation based on track categories, the index provides a standardized method for evaluating track geometry conditions. The index ranges from 50 to 150, with acceptable values falling between 70 and 90 (Andersson, M. 2002). The Federal Railroad Administration (FRA) has developed a series of objective Track Quality Indexes (TQIs) to complement the Federal Track Safety Standards (FTSS), utilizing track geometry data. These indexes use a space curve length to quantify track quality, with each TQI computed over nominal 161-meter track segments. The method involves calculating TQIs for profile, alignment, cross-level, and gauge, providing a track condition assessment by federal safety standards (J. Sadeghi 2010a).

(Yan and Corman 2020) reviewed the Canadian track quality index, calculated from the average of six different quality indices: gauge, cross-level, left (right) surface, and left (right) alignment.

TQI evaluates the general state of a railway track based on several elements, including surface defects, alignment, and track geometry. This study highlights the potential of on-board monitoring (OBM) techniques to reduce inspection costs and increase data collection without disrupting traffic. It reveals that Track Quality Indices (TQIs) are often developed based on national regulations, underscoring the need to consider multiple TQIs for effective maintenance decisions. Through case studies with hypothetical data, the study finds that high sensitivity and accuracy indices are effective in defect detection but may lead to false positives. The study calls for continuous research and development of TQIs, particularly for components such as ties and subgrades, to adapt to technological advancements and changing railway conditions. However, the study's limitations include reliance on hypothetical data due to the unavailability of real data. The track quality index assesses only geometric parameters, not structural conditions like rail, ties, and fastening systems.

(Bai et al. 2015) described the Track Quality Index (TQI) within the Chinese railroad system, encompassing vertical and horizontal alignment, gauge, cant, and twist parameters. A unique feature of this TQI is the calculation of standard deviations for each parameter, which are then aggregated to derive the overall TQI value. This approach differs from Sadeghi's TQI model used in Iran, where different parameters may be weighted differently. In the Chinese model, each parameter is given equal importance, reflecting a balanced approach to track quality assessment. This method aims to provide a fair evaluation of all critical aspects of track geometry, ensuring no single parameter disproportionately influences the overall quality index.

(El-Sibaie and Zhang 2004) Further developed this method by analyzing extensive track geometry data collected by modern inspection vehicles to establish objective, quantitative indicators that describe track conditions. The TQIs are derived from key track geometry parameters important for track performance and safety. Despite these advancements, there are some limitations. The study primarily focuses on Classes 3 to 5 tracks due to data availability, resulting in less reliable TQI thresholds for Class 2 tracks. Additionally, there is an overlap in TQI values between different track classes, which may reduce the precision of class differentiation.

(Q. Li et al. 2019) developed a comprehensive model for evaluating the health of railway tracks by dividing a continuous track line into adjacent segments, referred to as track grids. The model

employs a condition-evaluation index system, which considers multiple perspectives: Track Quality Index (TQI), Rate of change of TQI, Average failure rate, Rate of change of failure, Concentration rate of failure, and Hazard rate. Deep autoencoder networks (DANs) are used to reduce the dimensions of the data on these condition measures. At the same time, the hybrid hierarchical k-means clustering (HHKMC) method identifies track grid health features. The tree-augmented naïve Bayes (TAN) algorithm then calculates the track grid health index (TGHI), which provides a comprehensive assessment of track health on a smaller spatial scale. This model was validated using measurement data from the Lanxin Railway in China, showing superior performance compared to conventional methods. However, despite technological advancements, more is still needed, especially in selecting and weighing condition indexes. The current approach may introduce subjectivity and requires a structured, systematic method.

The Track Quality Index (TQI) has become an essential tool for assessing railway tracks' condition and maintenance needs. TQI methods, such as the UK SD Index (Setiawan and Sri Atmaja 2016), Netherlands Q Index (R.-K. Liu et al. 2015), USA TRI (Lasisi and Attoh-Okine 2018), and various others, provide quantitative evaluations based on specific track parameters like gauge, superelevation, and alignment. These indices ensure railway operations' safety, reliability, and efficient resource allocation. However, existing methodologies primarily focus on isolated aspects of track quality, often relying on limited parameters that may not capture the comprehensive nature of track degradation. For instance, while these indices offer valuable insights into certain aspects like track geometry, they frequently overlook other critical factors, such as the condition of sub-structural components and traffic influences, leading to potentially incomplete assessments of track health.

In contrast to traditional TQIs, (Hui Li and Xiao 2014) proposed a Generalized Energy Index (GEI), addressing the limitations of conventional TQIs in accounting for the influence of different-wavelength components of track irregularity. The GEI focuses on the effects of varying wheel-to-rail wavelength vibrations, emphasizing the importance of considering longer vibration-based wavelengths, particularly at higher speeds. This approach significantly departs from traditional indices by integrating the dynamic interactions between the wheel and rail, which are often overlooked in standard TQIs. While traditional TQIs, such as those mentioned by (Bogdan Sowinski 2013), usually evaluate individual parameters in isolation, the GEI provides a more

holistic assessment by capturing the energy dynamics of track irregularities. Additionally, comparisons between TQIs like the Chinese and Swedish models reveal differing methodologies: the Chinese Index assigns equal weight to each parameter, while Sweden's TQI places greater emphasis on cant error, highlighting the variability in TQI formulations and the need for comprehensive evaluations to determine their effectiveness.

(Haifeng Li and Xu 2009) has developed a railway track Integral Maintenance Index (IMI) that represents a significant advancement in railway maintenance. The IMI provides a comprehensive metric for evaluating track geometry and determining maintenance needs, aiming to address the increasing complexity of track maintenance due to large-scale speed-ups in China's railway network. By integrating multiple track geometry parameters, such as profile, alignment, cross-level, and historical maintenance data, the IMI allows for a more holistic assessment of track conditions. This comprehensive approach can be used to develop more accurate and effective maintenance plans. Unlike traditional methods that often focus on individual track quality indices (TQIs), the IMI considers the cumulative impact of various factors, making it a more reliable indicator of overall track health. The application of IMI on the Shanghai-Nanjing railway line has demonstrated its potential in aiding the efficient allocation of resources and ensuring the safety and reliability of railway operations. However, the IMI has limitations, including the complexity of its calculation and the need for accurate track geometry data. Continuous refinement and adaptation are necessary to address these inherent complexities and limitations in railway maintenance.

One major limitation of current TQI approaches is their narrow focus on geometric parameters, which limits their ability to provide a comprehensive assessment of track health. TQIs primarily evaluate factors such as gauge, alignment, cross-level, and twist. Most methods are constrained by their emphasis on specific elements of the track system, such as geometry, without adequately integrating data on sub-structural or traffic-related parameters. This fragmented approach can hinder the ability to fully understand the interplay between various factors contributing to track deterioration, thereby limiting the effectiveness of maintenance strategies. Additionally, TQIs that rely on standard deviation calculations for these geometric measurements can misrepresent track conditions, especially in curves where natural variations in parameters like gauge and twist occur. This focus on geometry alone can lead to skewed assessments, where the true health of the track,

particularly regarding the condition of its foundational components, is underrepresented (Offenbacher et al. 2020). Moreover, the application of these indices can vary significantly across different railway systems, as seen in the case of Indonesian Railways (Setiawan and Sri Atmaja 2016), where TQI application differs in maintenance regulations and accident investigations.

(S Kaewunruen, AM Remennikov 2005) have developed an innovative approach to evaluate the structural health of railway tracks by combining field measurements with track simulations. Their integrated method uses experimental modal analysis and finite element modeling to assess the dynamic parameters of in-situ railway track components. The study focused on a coal line in Central Queensland, Australia, where rail assemblies were tested using an instrumented hammer impact technique. The recorded frequency response functions (FRFs) were analyzed to determine the track components' dynamic stiffness and damping constants. The methodology involves conducting field dynamic testing by applying excitations to the track using an instrumented hammer, with the resulting vibrations captured by accelerometers. The data is processed using Fast Fourier Transform (FFT) and Mode Superposition (MS) methods to extract dynamic properties such as stiffness and damping coefficients. While the approach offers significant benefits in identifying the structural health of railway tracks, it only assesses specific components of the track, such as rail pads and ballast, potentially overlooking other critical elements. Additionally, the variability in damping coefficients and the specificity of the test site in Central Queensland also limit the generalizability of the results to other railway environments.

The U.S. Army (Uzarski et al. 1993) developed the RAILER system to evaluate the condition of low-volume railroad tracks by using several indices that measure key components such as ties, rails, joints, ballast, and subgrade. These indices help managers prioritize maintenance and repairs to ensure the safe operation of trains. For example, the Tie Condition Index (TCI) assesses the condition of the ties supporting the rails by checking for defects such as cracking, rot, missing ties, and improper positioning. Proper tie condition is essential to maintaining the stability of the rails and evenly distributing the weight of passing trains. The Rail and Joints Condition Index (RJCI) evaluates the state of the rails and the joints that connect them. It measures wear, cracks, and joint stability, which is critical for preventing track failures and derailments. Rail defects can pose significant risks if not detected and repaired promptly. The Ballast and Subgrade Condition Index (BSCI) focuses on the integrity of the ballast and the subgrade, which provide structural support

for the track. The BSCI evaluates factors such as ballast fouling, drainage problems, and subgrade compaction, which affect track alignment and stability. Poor ballast or subgrade conditions can lead to uneven settling, impacting train operations' smoothness and safety. These assessments are combined in the Track Structure Condition Index (TSCI), which gives an overall rating of the track's structural condition. This rating helps decision-makers plan maintenance activities and allocate resources efficiently.

However, the RAILER system has certain limitations. One major limitation is that it was designed primarily for low-volume tracks, which handle less traffic than mainlines or high-speed rail networks. These high-traffic tracks experience different types of stress, and the RAILER system may not fully account for the more demanding conditions they face. Another limitation is the system's reliance on manual inspections. Inspectors are required to physically inspect the tracks, which can be time-consuming and prone to human error. While manual inspections provide detailed observations, they are less efficient compared to automated technologies, such as track geometry cars or drones, which can collect data faster and more accurately.

The Ballast Condition Index (BCI) is a measure used to evaluate the quality and functionality of railway ballast, primarily by considering factors such as ballast thickness and the level of fouling, which refers to the contamination of ballast with fine particles (McDowell et al. 2004). Traditional assessment methods, such as the Ballast Fouling Index and Percentage Void Contamination (PVC), are commonly used to evaluate ballast fouling by focusing on fine particle contamination. While these methods provide essential insights, they have significant limitations. The FI primarily considers the ballast component, overlooking the specific gravity and type of fouling material, which can lead to inaccurate assessments. It also neglects the broader impact of fouling on other track elements like subgrade and drainage systems. The PVC method, while addressing void reduction, is time-consuming and does not account for particle gradation, potentially leading to an overestimation of fouling severity. The Relative Ballast Fouling Ratio (Rb-f) has been introduced by (Indraratna, Su, and Rujikiatkamjorn 2011b) to address these issues by incorporating both the specific gravity and gradation of fouling materials, but it also faces challenges in measurement precision and requires further validation. Therefore, it's crucial to develop more accurate and comprehensive assessment methods to fully evaluate track health. Consequently, while FI offers valuable insights into the condition of the ballast itself, it may not fully capture the broader impact

of fouling on overall track stability and performance. Other factors, such as the type of fouling material and its interaction with the subgrade, should also be considered to provide a more comprehensive evaluation of track health.

(Georgetown Rail 2022) has developed a tie rating system that uses autonomous track inspection technology to evaluate each tie's condition individually. The system examines more than 20 variables, such as plate cut, splitting, and internal decay, to assess tie conditions. Afterwards, each tie is graded on a scale from 1.0 to 4.0, with 1.0 indicating the best condition and 4.0 indicating failure. (J. M. Sadeghi and Askarinejad 2011) developed a quality index based on visual inspection to assess the structural condition of the track. The index includes the rail quality index (RQI), ballast quality index (BQI), sleeper quality index (SQI), and overall track quality index (TQI) and used a weighted deduction density model to determine the degree of deterioration based on distress density, type, and severity. The index illustrates three severity levels (low, moderate, and high) and their descriptions. Maintenance actions are organized by dividing the track line into management sections and segments for visual inspection.

Table 2: Summary of the condition rating systems

Condition Rating system	**Characteristics**
Track Quality Index (TQI)	Evaluates the geometry parameters such as gauge, cross-level, and surface alignment to assess overall track condition
Track Geometry Index (TGI)	Focuses on the standard deviations of key geometric parameters like unevenness, alignment, gauge, and twist, providing a statistical approach to track monitoring
Swedish National Railway Quality Index	Measures the standard deviation of the left and right profile and evaluates geometry defects to maintain consistent track quality
Federal Track Safety Standards (FTSS)	Assesses profile, alignment, cross-level, and gauge, ensuring compliance with federal safety standards in track geometry
Canadian Track Quality Index	Monitors gauge, cross-level, and surface alignment to detect irregularities and maintain safe track conditions

Netherlands's Q Index	Evaluates longitudinal levels, along with alignment and cross-level combinations, offering insight into the track
UK SD index	Tracks longitudinal levels, alignment, gauge, and twist to maintain smooth track geometry and safe operations
Generalized Energy Index (GEI),	Analyzes wheel-to-rail vibration wavelengths to assess dynamic interactions and identify irregularities in track geometry
Rail and Joint Condition Index (RJCI)	Evaluates the condition of rail joints, identifying issues that may affect track stability and performance
Ballast and Subgrade Condition Index (BSCI)	Assesses ballast fouling, drainage problems, and subgrade compaction to ensure the integrity of the track bed and support structures
Tie rating system	Measures internal decay, splitting, and plate cuts in railway ties, ensuring the structural integrity of the ties that support the rails

The various track condition rating systems as provided in Table 2, such as the Track Quality Index (TQI), Track Geometry Index (TGI), and the UK SD Index, provide different perspectives on assessing track health, each emphasizing specific parameters and methodologies. The Track Quality Index (TQI) focuses primarily on geometry parameters such as gauge, cross-level, and surface alignment, offering a straightforward evaluation of overall track condition. In contrast, the Track Geometry Index (TGI) provides a more statistical approach by measuring the standard deviations of key geometric factors like unevenness, alignment, gauge, and twist, which allows for a deeper analysis of track anomalies. The UK SD Index also tracks similar geometric properties but places particular emphasis on longitudinal levels and twists, helping to ensure smooth track geometry and safe operations. Other indices like the Swedish National Railway Quality Index and the Netherlands' Q Index similarly focus on specific geometry elements. However, they may integrate different combinations, such as evaluating alignment and cross-level.

Meanwhile, more specialized indices, such as the Rail and Joint Condition Index (RJCI) and the Ballast and Subgrade Condition Index (BSCI), delve into track components like rail joints and subgrade conditions, which complement the broader geometry-based approaches by ensuring structural stability at a more granular level. Together, these indices provide a comprehensive toolkit for maintaining track quality, emphasizing different aspects of track geometry and structural integrity. However, these systems overlook important factors such as crack location, tie plate condition, and the state of fasteners, which are crucial for a more complete assessment of track condition and safety.

Based on the studies mentioned earlier, most existing models focus on evaluating individual components or specific types of defects in railway infrastructure. They often overlook a comprehensive assessment of all railway components. For example, many models only assess track geometry conditions, neglecting other important defects and components. This limited approach can result in inefficient and inaccurate maintenance budget allocation due to the models' failure to represent the overall condition of railway components accurately. Most of the reviewed work relies on visual inspections, which are prone to human error. These models are crucial for decision-makers to prioritize the maintenance of multiple railway components across different projects based on their performance. However, one notable shortcoming is the oversight of inherent uncertainties and unexpected conditions encountered during the inspection process. Failing to address these uncertainties can lead to unreliable maintenance decision support systems.

Subsequently, this gap in comprehensive assessment was highlighted when analyzing the role of the crosstie in the rail-to-tie and tie-to-ballast load distribution. Crossties serve as intermediaries that distribute loads and resist the forces exerted by other track components. The effects of tie plates and the presence of ballast in the tie crack, particularly plate cutting and ballast abrasion, were identified as significant factors accelerating the deterioration of wooden ties, especially under the stress of faster and higher tonnage trains. The research was conducted to gather insights from various studies on the impact of spikes on ties. Studies by M. Dersch et al. (2019), Gao, McHenry, and Kerkhof (2018), and others emphasized the crucial role of spikes in maintaining tie integrity. Broken or missing spikes lead to direct stress transfer from the train to the tie, accelerating tie deterioration. The findings from these studies underscored the importance of including spike conditions—such as broken or missing spikes—in the proposed rating system. For Instance, the

main track derailment at Fabyan, Alberta (The Transportation Safety Board of Canada 2012), highlighted significant issues in rail safety practices, especially in inspecting and maintaining rail fastening systems. The subsequent investigation uncovered critical flaws, such as the failure of lag screws and the inability of traditional inspection methods to detect curve stress. While these findings led to more thorough inspection procedures, such as detailed curve inspections and geometry car printouts, there has been minimal research on the condition rating of ties, rails, and fastening systems. This gap highlights the need for a comprehensive rating system and a degradation prediction model integrated with automated inspection technologies. By using high-resolution cameras, sensors, and artificial intelligence to monitor rail conditions in real-time, this approach aims to proactively identify anomalies and potential risks, thereby reducing the likelihood of derailments and improving overall rail network safety.

2.4 Condition Prediction Models

Predictive models can be largely categorized as mechanistic, statistical or machine learning. Mechanistic are the earliest models for predicting degradation in railway tracks. They reflect physical phenomena, and track deterioration based on loads and material characteristics (Falamarzi, Moridpour, and Nazem 2019). Some of the variables used in these models to represent deterioration are Track settlement, Track deformation, Track geometry (e.g., gauge), and Track Quality Index (TQI). However, these models are limited by their inability to factor in the inherent uncertainty of track degradation behavior and their applicability to only a select number of track sections, rather than the entire network(Elkhoury et al. 2018). Developing mechanistic models can also pose a significant challenge as they require a considerable amount of physical data and time. A statistical model is a type of mathematical model which uses historical data to predict the pattern of deterioration. These models are widely used to predict the deterioration of railway tracks based on observations and the influencing factors such as traffic, track components and maintenance variables. Statistical models are divided into three main groups: deterministic, stochastic, and probabilistic.

2.4.1 Statistical Models

Statistical models can be classified into three main groups: deterministic, probabilistic, and stochastic. Deterministic models assume a direct and exact relationship between input and output variables without accounting for randomness. Stochastic models incorporate random variables and account for inherent randomness and variability in the system. Probabilistic models incorporate the influence of random events or actions to predict the likelihood of future outcomes. A deterministic model is a statistical model in which randomness is not involved in predicting future conditions. It is usually applied where relationships between components of the rail structure are identified(Md Saeed Hasan 2015). Deterministic models in railway condition assessment use data parameters such as train speed, rail geometry, rail operations (Audley and Andrews 2013), and accumulated tonnage (MGT) (Guler, Jovanovic, and Evren 2011). Studies have confirmed a correlation between track defects and train loads, measured in million gross tons (MGTs). For instance, (R. Liu, Xu, and Wang 2010) developed a Short-Range Prediction Model (SRPM) for China railway lines to predict track irregularities using a linear regressor model. However, this model accurately predicted only nine out of 25 sections and struggled with nonlinear surface changes. Similarly, (Guler, Jovanovic, and Evren 2011)created a model for predicting geometric degradation in Turkey, focusing on factors like Twist, Gauge, Alignment, Cant, and Level. However, this model did not adequately consider the effects of speed and load, which suggested that higher speeds and loads decreased deterioration rates. While these deterministic models identify general statistical patterns and influencing factors, they may overlook essential degradation factors and do not account for uncertainties in input parameters and model geometry (Elkhoury et al. 2018) and this can limit their effectiveness in making precise maintenance and system improvement decisions.

Probabilistic modelling is a statistical method that incorporates the influence of random events or actions to predict the likelihood of future outcomes. Probabilistic modelling provides forecasts or estimates of possible future results by considering the impact of chance occurrences. Previous researchers have approximated track degradation through the probabilistic model using different types of probabilistic approaches to normal distribution (J. Sadeghi 2010b), Weibull distributions (Caetano and Teixeira 2015), and (Shafahi and Hakhamaneshi 2009) Markov model, as indicated in Table 3. (J. Sadeghi 2010b) Developed track geometry indices based on normal data distribution

for parameters like gauge, twist, longitudinal level, and alignment. These indices, calculated separately for different track classes, aimed to evaluate track conditions and guide maintenance. However, the model's effectiveness is limited to the specific track classes and assumes that the data follows a normal distribution. This assumption may not hold for all railway systems, potentially affecting the model's accuracy if the data deviates from a normal distribution. (A.R. Andrade and Teixeira 2015) Developed a hierarchical Bayesian model (HBM) to predict the degradation of train tracks in Portugal. The model used data from a major train line between Lisbon and Oporto to evaluate the Standard Deviation of Longitudinal Level defects (SDLL) and the Standard Deviation of Horizontal Alignment defects (SDHA). Bayesian models treat parameters as random variables and incorporate uncertainty through prior distributions. This method combines the previous distribution with the likelihood of the observed data to compute the posterior distribution of the parameters. In practical applications, calculating the joint posterior distribution often involves complex numerical integration, typically using Markov Chain Monte Carlo (MCMC) methods. When applied to operational and maintenance data, the HBM proved to be a poor predictor of SDHA compared to SDLL. This indicates that horizontal alignment defects are less predictable, highlighting a limitation in the model's effectiveness for certain types of track geometry degradation.(Caetano and Teixeira 2015) Developed a model using the Weibull distribution to schedule maintenance and renewal of railway tracks, aiming to minimize life-cycle costs. Based on historical data from a rail line in Portugal, the model showed that optimal maintenance could be achieved by selecting suitable time intervals for renewals. However, the study noted that insufficient rail and sleeper degradation data might limit the model's accuracy in representing actual degradation rates. Markov models assess rail track conditions over time by considering tonnage, axle load, terrain, traffic conditions, and a combined track record index (Shafahi and Hakhamaneshi 2009). These models analyze track deterioration from optimal conditions to where maintenance is required, categorizing tracks into six classes based on traffic loads and geographical locations. However, this classification may not capture variations within each class, potentially leading to inaccuracies in predicting deterioration for specific segments. Markov models have limitations in capturing the random behavior of track deterioration and optimizing maintenance costs, and they rely heavily on data availability. Probabilistic models face challenges due to the often-limited historical data available, making accurate prediction of track deterioration difficult. A stochastic model is an approach used to predict statistical characteristics

of potential outcomes by considering the random fluctuations in one or more parameters over time. By accounting for the unpredictable variations, a stochastic model provides insights into the potential properties or patterns that may arise, and the different types of stochastic models and their summary is given in Table 3. (Vale and M. Lurdes 2013) Proposed a stochastic model to predict track degradation over time, focusing on a Northern railway line in Portugal. The study adhered to European Committee for Standardization (CEN) guidelines and conducted statistical and probabilistic analysis for various vehicle speed groups using the Dagum distribution. The researchers tested 52 probabilistic distributions with Easy Fit software, finding the Dagum model the best fit. The study examined 21-time intervals across three-speed groups, discovering similar left and right rails degradation rates in 63 cases. The Dagum distribution accurately modelled the longitudinal level degradation for these cases. However, the study noted that the 90-day interval between dynamic inspections might not capture rapid changes in track degradation. (Quiroga and Schnieder 2010) Developed an autoregressive model using the Auto-Regressive Moving Average (ARMA) method to predict railway track geometry deterioration. They focused on the standard deviation of the longitudinal level as an indicator of degradation, using previous values, section length, and length of tamped tracks for predictions. The researchers applied this model to a section of a French high-speed railway; the model showed promise for integration into tamping scheduling systems. However, while it adapted quickly after tamping, it may struggle with accuracy between tamping activities, potentially leading to inaccuracies in long-term predictions. (He et al. 2015a) developed a statistical deterioration model to depict the course of degradation of various track geometry defects. Based on exploratory data analysis, they employed an exponential link between the degradation rate and outside variables (tonnage carried, number of cars, trains, and inspection trips since the last red tag was spotted). (Alemazkoor, Ruppert, and Meidani 2018) Evaluated the probability of failure to predict the time transition from yellow to red tag using the survival analysis model. Comparing the two models revealed that the fine-scale defect-based model performed better than the coarse-scale segment-based model for survival. Therefore, fine-scale defect-based survival models are applied to predict the likelihood of at least one red tag defect in a segment. Stochastic models are commonly used to predict the deterioration of various applications. However, to enhance the accuracy of the models, a deeper understanding of the application and a more detailed explanation are required. These models are best suited for short—to medium-term predictions. They may not accurately capture complex nonlinear relationships.

Table 3 Summary of statistical models

Model Type	Method	Input Variable	Target variable	Accuracy	Reference
Deterministic	Linear regression	Tonnage, Speed, Initial inspection date, Inspection date after maintenance, gauge, cross-level, alignment, Surface, and twist.	Track surface irregularity Measurement	The average error for the actual and predicted track surface is 0.120 mm	(R. Liu, Xu, and Wang 2010)
Deterministic	Linear regression	Gradient, Curvature, Speed, Age, rail type, Rail length, Sleeper type, Flood, Falling rock, landslide and Snow	Twist Gauge Alignment Cant Level	0.62 R^2 0.71 R^2 0.69 R^2 0.77 R^2 0.68 R^2	(Guler, Jovanovic, and Evren 2011)
Deterministic	Linear regression (Three parameter Weibull distribution)	Maintenance data (Full renewal date, Tamping date), Tonnage and speed	Standard deviation of track quality	0.98 R^2	(Audley and Andrews 2013)
Probabilistic	Normal distribution	Gauge, Profile, Alignment, and Twist.	Track geometry index	0.80 R^2	(J. Sadeghi 2010b)

Probabilistic	Weibull distribution	Tonnage, Rail age, Sleeper age, Rail and Sleeper hazard rate, Time period of the failures, and Track segment.	Accumulated maintenance operations	0.97 R^2	(Caetano and Teixeira 2015)
Probabilistic	Markov	Tonnage, Design axle load, Terrain (Plain, Hilly, Mountainous), Traffic Condition, Combined track record index.	Track degradation Rate	0.83 R^2	(Shafahi and Hakhamaneshi 2009)
Stochastic	Dagum distribution	Initial geometrical quality, the degradation rate, Speed, Maintenance activity	Standard deviation of longitudinal level defects	0.79 R^2	(Vale and M. Lurdes 2013)
Stochastic	Time series model (The Auto-Regressive Moving Average)	Tamping schedule, Standard deviation of longitudinal level, Tonnage, Time period of the maintenance activity	Degradation rate	66.5% MSE	(Quiroga and Schnieder 2010)
Stochastic	Survival analysis	Tonnage, Number of cars and trains travelling over the inspection period, defect tag, Amplitude of the defect.	Cant Dip Gage	0.242 MSE 0.099 MSE 0.046 MSE	(He et al. 2015b)

2.4.2 Machine Learning Models

Over the past few years, machine learning models have been widely adopted for their exceptional ability to enhance statistical models in predicting rail track degradation. Machine learning is a branch of artificial intelligence that utilizes historical and current data to anticipate the future state of a system. These models utilize computer applications to replicate human-like intelligence and automate intelligent functions. By leveraging advanced computer techniques and reasoning algorithms, Machine Learning models surpass the limitations of current models and deliver superior results (Jovanovic, Guler, and Coko 2015). Common types of machine learning models applied in previous studies as mentioned in Table 4, include artificial neural networks (ANNs), Adaptive Neuro-Fuzzy Inference Systems (ANFIS), Support vector machines (SVM), and Random Forests (RF).

Numerous studies have implemented machine learning and statistical techniques to develop efficient predictive models in construction and infrastructure management (Herrero, Bayraktar, and Jiménez 2020) (Bhatia, Han, and Moselhi 2022). Several predictive models were developed for preventive track maintenance (Soares 2011) (Rahimikelarijani, Mohassel, and Hamidi 2020) (He et al. 2015a). (Liao et al. 2022) examined ANN (Artificial Neural Networks) and SVM (Support Vector Machine) models. They discovered that SVM models can still produce accurate predictions even with a small sample size (inspection data). A considerable amount of high-quality inspection data is required for ANN models as training data. Compared to earlier developed models (Falamarzi, Moridpour, and Nazem 2019b), the use of the random forest regression model resulted in more accurate predictions on track degradation. Extreme gradient boosting (XGBoost) has further excelled in other infrastructure deterioration prediction domains (Amini and Dziedzic 2022). (Sudhir Kumar Sinha, Sumit Raut, and Harshad Khadilkar 2015) employed a machine-learning approach. Predictive models of different types of geometry defects, which include XLEVEL, SURFACE, and DIP, performed well with logistic regression and decision tree. (Cárdenas-Gallo et al. 2017) employed ensemble classifier approaches and discovered that at least one ensemble classifier was the best in each defect. As a result, they selected the Stacking with Binary Logistic Regression for the XLEVEL defect, the Bootstrap Aggregating for the SURFACE defect, and the Stacking with Support Vector Machine for the DIP defect.

(Moridpour, Mazloumi, and Hesami 2017) developed an artificial neural network model to predict the degradation of tram tracks in curved sections using maintenance data. The study used the Melbourne tram network as a case study. The researchers applied a multilayer feed-forward ANN model with three layers to predict the target variable. Artificial Neural Networks (ANNs) are complex systems of interconnected neurons that communicate through weighted connections. These neurons are arranged in layers within the network, and each neuron's output is transmitted to the next neuron through a connection (Guler 2013). The model included variables such as rail type, rail profile, passing tonnage in MGT, and the installation year to predict the deviation of the track gauge parameter. The study found that the type of tracks and the last gauge measurement significantly impact the track geometry deviation. The developed model had reasonably good prediction accuracy.

(Javad Sadeghi and Askarinejad 2012) used an artificial neural network (ANN) to evaluate railway track quality by establishing links between track geometry defects and structural issues. The ANN model architecture used was multilayer feed-forward network with Standard Deviations of track geometry data as inputs and the predicted defect density of track structural components as outputs. The study found that the proposed ANN model was more accurate for low and medium-quality track conditions. The study highlighted the benefits of using automated inspections and neural networks to establish correlations between track structural conditions and inspection data. However, the study utilized a simplified neural network architecture and may not account for all the complexities of track structural conditions. In Turkey, (Guler 2014b) conducted a study that used an ANN model to predict rail track degradation. This case study was carried out for Turkish state railways and involved a thorough investigation over two years, covering a track length of approximately 180 km. Different variables were considered in the data collection process, including track structure, traffic characteristics, track layout, and environmental factors. The author developed separate ANN models for the leading track geometry parameters and conducted a sensitivity analysis to determine the importance of each predictor in determining the neural networks.

Table 4: Summary of Machine Learning Models

Method	Input variable	Target variable	Accuracy	Reference

Artificial Neural Networks (ANN)	Month since last inspection, Gauge last inspection, MGT, Trips, Route, Rail profile, Rail type, Curve radius, Repair history, Year installation of track	Gauge value change per month	1.58 mm RMSE	(Moridpour, Mazloumi, and Hesami 2017)
Artificial Neural Networks (ANN)	Standard deviation of Gauge, Alignment, Profile, Twist	Defect density of rail Defect density of sleeper Defect density of ballast Defect density of Fasteners	0.75 R^2 (Class B track) 0.79 R^2 (Class B track) 0.77 R^2 (Class B track) 0.74 R^2 (Class B track)	(Javad Sadeghi and Askarinejad 2012)
Artificial Neural Networks (ANN)	Gradient, Curvature, Speed, Age, rail type, Rail length, Sleeper type, Flood, Falling rock, land slide and Snow	Twist Gauge Alignment Cross-level Levelling	0.72 R^2 0.79 R^2 0.76 R^2 0.83 R^2 0.74 R^2	(Guler 2014b)
Adaptive Neuro fuzzy Inference System (ANFIS)	Standard deviation of longitudinal level, Alignment, Cross-level, the number of tamping works previously carried out, and the number of days from elapsed from the last tamping	The number of days from the last tamping to the next one.	Measurement error not greater than 18% in all cases and in absolute terms not greater than 23 days over 131 days.	(Dell'Orco et al. 2008)

Adaptive Neuro fuzzy Inference System (ANFIS)	Gauge values for the previous two years (s-2 and s-1), MGT	Gauge values for the year (s)	Curves 0.6 R^2 Straights 0.78 R^2	(Karimpour et al. 2018)
Support vector machines (SVM)	Historical data from hot box detectors, Wheel impact load detectors	Alarm prediction	True positive rate 97.5% False positive rate 5.65%	(Hongfei Li et al. 2014)
Binary logistic regression	Standard deviation of longitudinal level, Kurtosis of longitudinal level, Time interval, defects which exceeds the planning limit	UH2 defects	Sensitivity and specificity 89%	(Soleimanmeigouni et al. 2020)
Random Forest	Previous Track deterioration index (TDI), Track surface, Rail type.	Track deterioration index	0.90 R^2	(Falamarzi et al. 2018)
Support vector Machine (SVM)	Previous track longitudinal measurements of the left and right rail of different wavelengths (3-25m is called D1) and (25-70m is called D2) from September 2018 to December 2019.	Track longitudinal level (January 2020)	D1L 0.951 R^2 D1R 0.941 R^2 D2L 0.636 R^2 D2R 0.601 R^2	(Han et al. 2024)

Deep neural network (DNN)	Previous Standard deviation of track longitudinal measurements of the left and right rail of different wavelengths (3-25m is called SDD1) and (25-70m is called SDD2) from September 2018 to December 2019.	The standard deviation of track longitudinal level (January 2020)	SDD1L 0.962 R^2 SDD1R 0.968 R^2 SDD2L 0.955 R^2 SDD2R 0.978 R^2	
	Previous track longitudinal measurements of the left and right rail of different wavelengths (3-25m is called D1) and (25-70m is called D2) from September 2018 to December 2019.	Track longitudinal level (January 2020)	D1L 0.980 R^2 D1R 0.976 R^2 D2L 0.961 R^2 D2R 0.959 R^2	
	Previous Standard deviation of track longitudinal measurements of the left and right rail of different wavelengths (3-25m is called SDD1) and (25-70m is called SDD2) from September 2018 to December 2019.	The standard deviation of track longitudinal level (January 2020)	SDD1L 0.962 R^2 SDD1R 0.951 R^2 SDD2L 0.968 R^2 SDD2R 0.964 R^2	

An ANFIS model combines the use of ANN and a Fuzzy Inference Engine (FIS). This integration allows for the principles of both fuzzy logic and neural networks to be utilized within a single framework, resulting in potential benefits from both(Zimmermann 2010). (Dell'Orco et al. 2008)

developed an ANFIS model to optimize rail track maintenance and planning. The model considers geometry parameters such as alignment, longitudinal level, and cross-level, as well as the number of days since the latest tamping and the number of previous tamping works. Its output is the number of days between tamping works. The study found that the model accurately predicted maintenance dates that met or exceeded the maintenance threshold.

In a study conducted by (Karimpour et al. 2018), an ANFIS model was developed to predict rail track degradation using the gauge parameter. The findings revealed that a precise model can accurately predict the long-term performance of rail tracks. The main parameters in the model development were gauge deviation parameters from the previous year and two years ago. The results indicate that the model can predict the gauge deviation for the upcoming year with satisfactory accuracy. There have been limited studies that compare the effectiveness of statistical and machine learning models. According to (Shafahi, Masoudi, and Hakhamaneshi 2008) research, a Markov chain model performed better than ANN and ANFIS models. This suggests that the higher computational complexity of ANN and ANFIS may not always be necessary. However, as autonomous track inspection programs become more prevalent and data collection increases, these methods may become more appealing.

It is possible to use other machine learning models to predict rail track degradation. In a study by (Hongfei Li et al. 2014), machine-learning models were used to forecast defects and alarms of critical components of rail cars. The study developed learned rules based on historical data to predict which rail cars were likely to have problems and to predict intensive existing alarms before an actual alarm event to decrease instant train stops. The development of the model involved five steps: feature extraction, dimension reduction, model training, prediction and confidence estimation, and rule simplification. To evaluate the results, the proposed SVM model and a decision tree were compared against the same data. Based on the results, the customized SVM model performed better than the decision tree for alarm prediction.

(Soleimanmeigouni et al. 2020) developed an analytical methodology using data that could predict track geometry defects in a railway line section in Sweden. Specifically, it focused on predicting UH2 defects, a track geometry defect that can cause safety problems and derailments. These defects have a linear degradation pattern, which was modelled using linear regression, along with

binary logistic regression to predict the probability of UH2 defects. The study also analyzed the impact of factors such as standard deviation and kurtosis of longitudinal level on the occurrence of UH2 defects. The results indicated that the developed models effectively predicted the occurrence of UH2 defects.(Falamarzi et al. 2018) have developed a Random Forests (RF) model to predict the future deterioration index. The Melbourne tram network has been used as the case study, and the gauge deviation parameter has been selected as the primary parameter to develop the index. The research findings suggest that the proposed model has a considerably high adjusted R2 value, and the prediction error is negligible, demonstrating its reasonable performance in predicting the deterioration index.From the abovementioned models, random forest and ANN have reasonably good predictions. These models can be used with the recent evolution of autonomous track inspection programs to maintain the railway track assets and their condition and continuously improve the track. The lack of literature and complex model structures are major drawbacks of these relatively new degradation prediction models.

2.5 Limitations of Existing Studies

The literature review on condition assessment in railways focuses narrowly on individual defects or specific components, such as track geometry, neglecting a comprehensive evaluation of all critical railway elements. This approach can lead to inefficient allocation of maintenance budgets and inaccurate representation of railway conditions. Relying solely on visual inspections introduces human error and fails to account for uncertainties and unexpected conditions, undermining the reliability of maintenance decision support systems. In a main track derailment in British Columbia in 2021, the Transportation Safety Board of Canada (TSB 2021) examination revealed that the gauge-side spikes of the high rail of the 8° curve had lifted away from the tie plates, causing the high rail to roll outward and increase the gauge, leading to the derailment. Despite the track undergoing inspection by a heavy track geometry test car nine days before the occurrence, the subsequent track inspection did not reveal track geometry defects, even though signs of gauge widening were likely present. This highlights the need for a comprehensive rating system and a degradation prediction model integrated with automated inspection technologies by utilizing high-resolution cameras, sensors, and artificial intelligence to monitor rail conditions in

real time. This literature review also explores the use of various condition prediction models in multiple case studies. There is an opportunity to explore machine learning models that not only predict the overall condition of the rail track but also quantify the size and severity of defects using machine learning and statistical techniques. For instance, employing algorithms such as random forests, XG Boost can enable the identification of complex patterns in historical inspection data. These models can be trained to recognize early signs of deterioration, enhancing defect predictions' accuracy. Moreover, integrating these predictive models with automated inspection technologies, such as high-resolution imaging systems and non-destructive testing methods, can facilitate extracting detailed features related to the physical characteristics of railway components. This integration would allow for real-time monitoring and assessment of rail conditions, leading to proactive maintenance strategies.

Chapter 3. Methodology

The methodology has two main parts, as described in the Figure 5. The first part, to the left, involves developing a condition rating, while the second, to the right, focuses on developing a predictive model. This section describes each step of the method in detail.

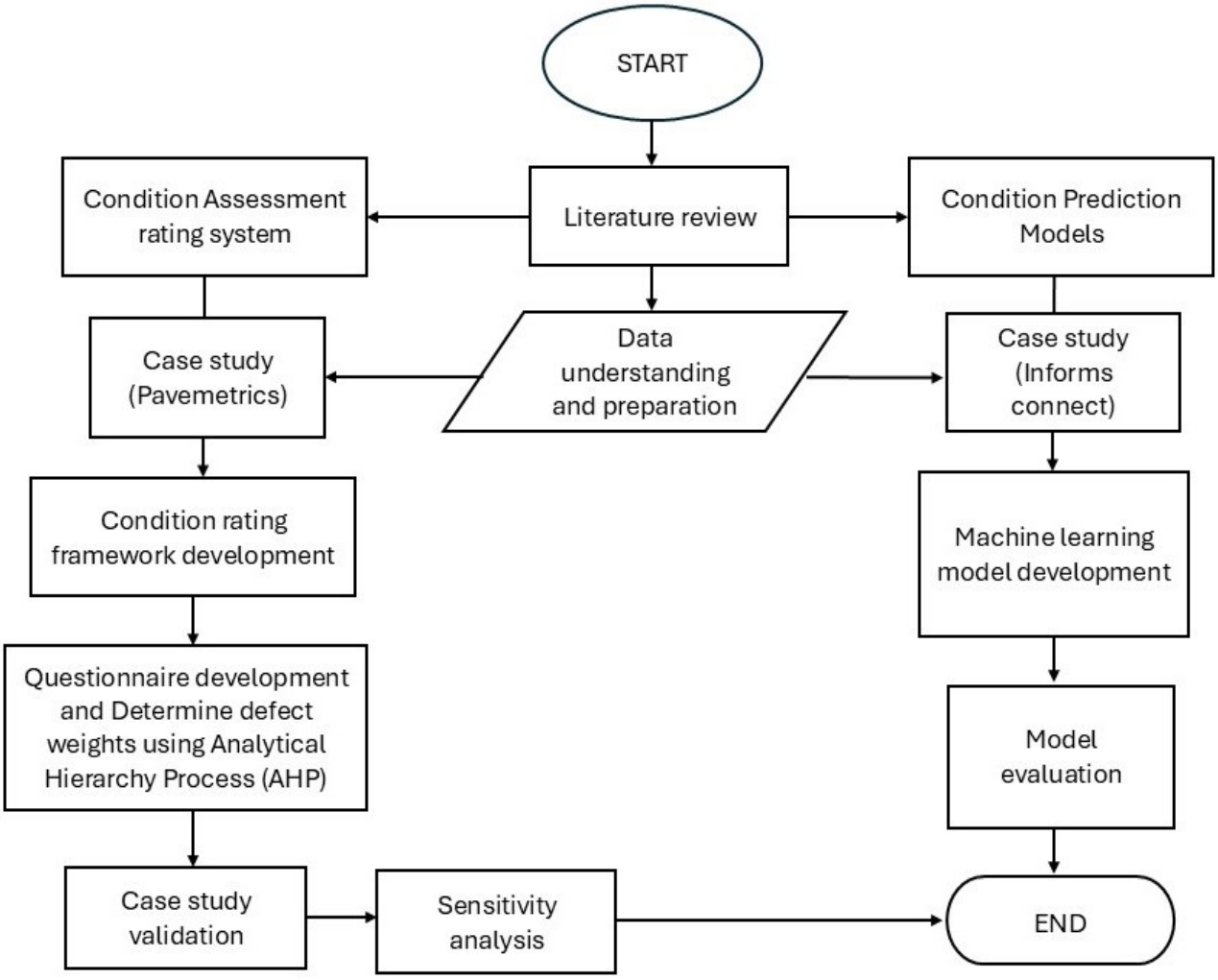


Figure 5: Overview of Research Methodology

3.1 Condition Rating System

The study utilized data obtained from Pavemetrics' automated track inspection technology. It employed the Analytic Hierarchy Process (AHP) to compare and prioritize multiple criteria

systematically, enabling a structured approach to decision-making. The overall methodology encompassed (1) Data understanding and preparation, (2) Framework development, (3) Questionnaire survey analysis using AHP and (4) Case study validation.

3.1.1 Case Study Description

The data for this study was obtained from Pavemetrics, a company specializing in automated railway track inspections using their L-RAIL technology. This technology uses 3D laser triangulation to scan railway tracks in detail, capturing a comprehensive dataset that includes various railway components such as rails, ties (sleepers), fasteners, spikes, tie plates, and ballast, as shown in the Figure 6. The L-RAIL system can automatically inspect various railway asset properties and defects and operates day or night at up to 180 km/h.

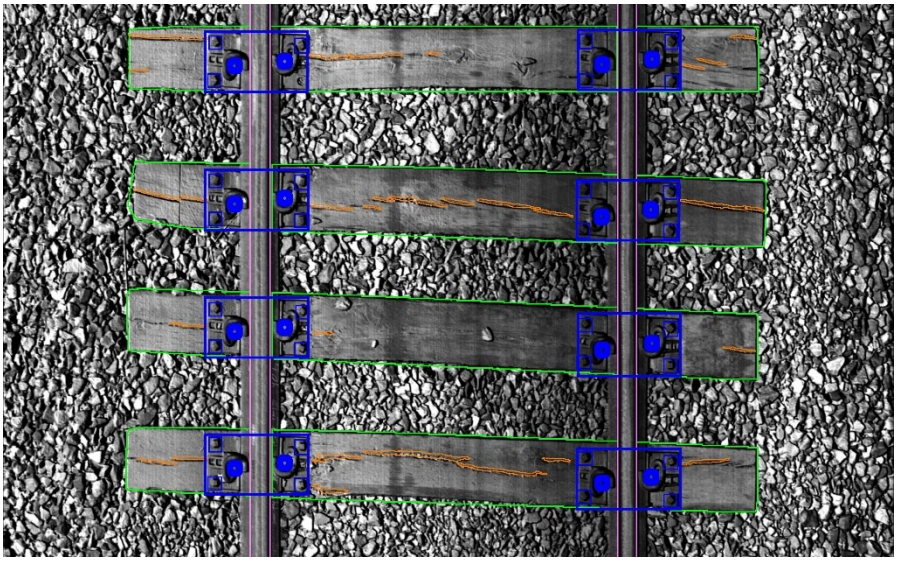


Figure 6: Sample image from condition assessment case study data

The following section provides an overview of data understanding and preparation. The accompanying meta-table summarizes the relevant attributes. The dataset used in this study covers a section of track from an anonymous location. The inspected distance totals 0.2 kilometers, encompassing 329 ties with 1,417 identified tie cracks. In addition to cracks, the dataset includes key characteristics of the railway infrastructure, such as the condition of spikes, tie plates, and ballast presence. This data forms the foundation for evaluating railway tie conditions and

conducting further predictive analysis on track integrity. The following section provides a detailed overview of data understanding and preparation. The accompanying meta-table summarizes the relevant attributes for this analysis.

3.1.2 Data Understanding and Preparation

Data understanding and preparation were critical steps in the data analysis process. The data was provided in various formats, including XML files, shapefiles, and CSV files. While some data was already in CSV format, additional information needed to be extracted from the XML files to enhance the dataset's comprehensiveness. The data from the XML files was extracted using Python scripts that parsed the XML structure and converted the relevant information into CSV format for easier manipulation and analysis. Additional information, such as In line with spike, Side, Number of anchors, Number of Fasteners, Number of Tie plates, Number of spikes, Spike ID, Mean Height Spike, Condition Spike, and Tie Plate ID, was extracted from the XML and converted into CSV format for easier manipulation and analysis and the detailed description of the features are mentioned Table 5.

Table 5: Meta data table for Pavemetrics data.

Features	Description	Type
Survey ID	Each survey is identified by a unique identification number (Survey ID)	Categorical
Section ID	Each survey section is labelled with a sequential number (Section ID).	Categorical
Tie ID	The ID of the tie in the current section.	Categorical
Distance	The linear distance from the beginning of the survey to the detected tie	Float
Tie Length	The length of the tie	Float
Tie Width	The width of the tie	Float
Askew Angle	The skew angle of the tie-in degrees	Float

Is At Border	Indicates if the tie is in the middle of 2 sections	Int
Tie Material	The element indicates the material of the tie: wood, concrete or undefined	Categorical
Covered Area Percentage	Element reports the percentage of the tie's surface covered by ballast, debris, and other materials	Float
Number of Crack	Reports the total number of cracks detected for a given tie	Int
Area	The area of the crack	Float
Width	The width of the crack	Float
Depth	The depth of the crack	Float
Length	The length of the crack	Float
Angle degree	The angle of the crack	Float
Presence of Ballast	Indicates if there is ballast present in the opening of a crack	Categorical
In line with the spike	Indicates whether the crack is in line with the spike	Float
Side	Indicates whether the crack is in the field or gauge side	Categorical
Number of anchors	The total number of anchors detected for the given tie	Int
Number of Fasteners	The total number of fasteners detected for the given tie	Int
Number of Tie plates	The total number of tie-plates detected for the given tie	Int
Number of spikes	The total number of spikes detected for the given tie	Int
Spike ID	Id of the spike in the current section	Categorical
Mean Height Spike	the height of the spike	Float
Condition Spike	indicates the status of the spike as good or high	Categorical
Tie Plate ID	Id of the tie-plate in the current section	Categorical

3.1.3 Rating System Framework Development

The existing Pavemetrics rating system, developed by their experts, evaluates tie cracks based on length, depth, and height, with minimum thresholds of 5mm depth, 10mm height, and 50mm length, as shown in Table 6. Cracks below these thresholds are ignored. While this system effectively identifies major cracks, it does not account for other critical factors, such as the condition of spikes and tie plates or the location of cracks in these components. In comparison, Georgetown Rail (2022) developed a tie rating system that evaluates over 20 variables, including internal decay, plate cuts, and splitting, grading ties on a scale from 1.0 (best) to 4.0 (failure). This system provides a more detailed assessment than Pavemetrics, covering a broader range of conditions. However, it does not consider the exact location of cracks. Crack location is an important factor in understanding the impact on track stability and performance, as cracks near spikes or tie plates can cause quicker damage and increase risks. By leaving out crack location, the Georgetown Rail rating system may miss important information needed for a more complete evaluation of tie condition and long-term track performance. Additionally, Sadeghi and Askarinejad (2011) introduced a quality index based on visual inspection, which evaluates the structural condition of the track through indices like rail quality, ballast quality, and sleeper quality. Their holistic approach addresses the overall track condition rather than focusing only on cracks. However, a limitation is that it relies heavily on visual inspections, which can be subjective and may miss smaller, less visible defects such as internal cracks or early signs of wear. This reliance on visual assessments can reduce the accuracy and consistency of the evaluation compared to more precise, technology-driven methods. The proposed system aims to provide a more comprehensive evaluation by incorporating spike- and tie-plate-related factors, along with crack location and size. This approach enhances the assessment of track conditions by addressing components the existing system does not consider, leading to a more detailed understanding of tie and rail fastening health.

Table 6 : Current rating system

Parameter	Value (depth, width, length)	Description
Wooden tie rating “3” very severe	20,50,600(unit mm)	This parameter allows the user to set a threshold for classifying a defect on a wooden tie as very severe. It must contain three

		values for depth, height, and defect length, all in mm.
Wooden tie rating “2” severe	15,30,180(unit mm)	This parameter allows the user to set a threshold for classifying a defect on a wooden tie as severe. It must contain three values for depth, height, and defect length, all in mm.
Wooden tie rating “1” moderate	5,15,100 (unit mm)	This parameter allows the user to set a threshold for classifying a defect on a wooden tie as moderate. It must contain three values for depth, height, and defect length, all in mm.
Wooden tie rating “0” light	Not applicable	light. The defects do not meet conditions “1”, “2” and “3”

The framework development incorporated the specific conditions observed on rail tracks, where high axle loads often led to spikes becoming loose over time, enlarging spike holes, and exposing the tie to moisture and decay, highlighting the necessity of considering the distance of cracks from spikes, the presence of cracks in spike holes, and the direction and alignment of cracks relative to the spikes. Geographic Information System (GIS) software, specifically QGIS, was utilized to ensure precise measurement of the crack's proximity to critical components. QGIS enabled the accurate mapping and measurement of the distance between cracks and critical elements, such as tie plates and spikes, as shown in Figure 7. The proposed rating system was designed to integrate all these factors, aiming to assess the potential for tie splitting and other forms of deterioration by examining the proximity of cracks to critical components. This comprehensive approach was developed with input from Pavemetrics Inc.'s principal consultant to ensure its relevance and applicability, as shown in the Figure 8.

Tie cracks are evaluated using two key components: crack size and crack location. Crack size is assessed based on sub-factors such as depth, width, length, and the presence of ballast in the crack. Crack location is evaluated based on factors like spike distance from the crack, tie plate distance from the crack, crack direction, whether the crack is in line with the spike, and whether the crack

is on the field or gauge side. For each sub-factor, a score is assigned based on its condition. The total scores for crack size and crack location are then multiplied to reflect the combined impact on the overall condition of the tie. This combined score represents the tie crack rating.

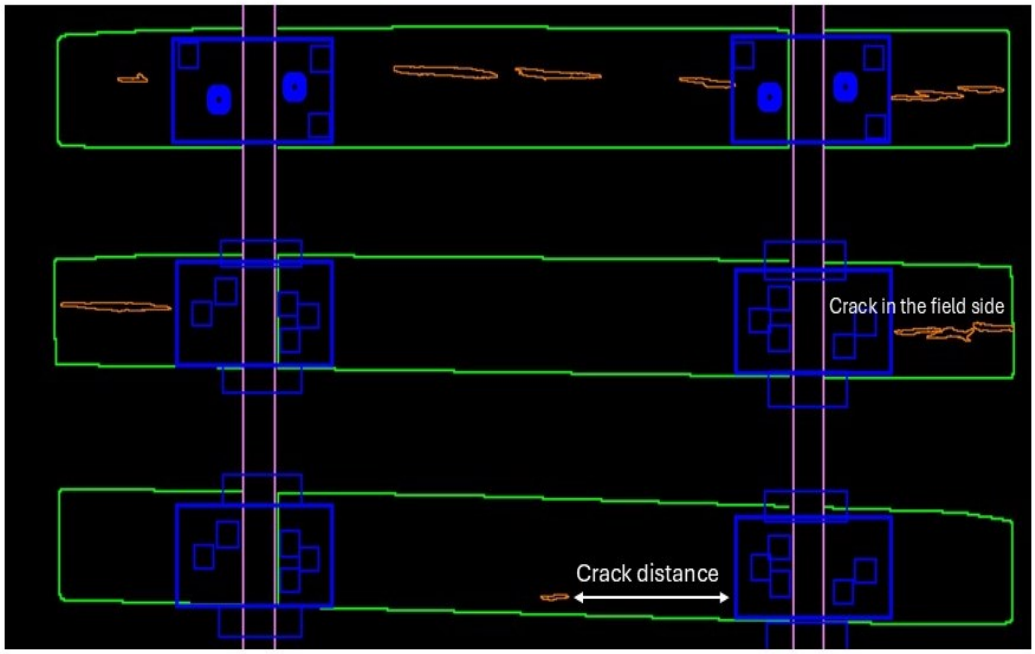


Figure 7: crack distance from the components.

The rating for tie plates is based on the number of missing tie plates. The score reflects the condition of the tie plates, with higher scores indicating worse conditions. For spikes, the rating is determined by spike height and whether the spike is missing or broken. A score is assigned for each factor based on the spike's condition. Once the individual scores for tie cracks, tie plates, and spikes are calculated, they are combined to generate the overall rating of the tie and rail fastening system. This overall rating provides a comprehensive assessment of the track's condition, with higher scores indicating more significant defects and lower scores representing better conditions. The weighted scoring system thoroughly evaluates the track's tie and rail fastening condition.

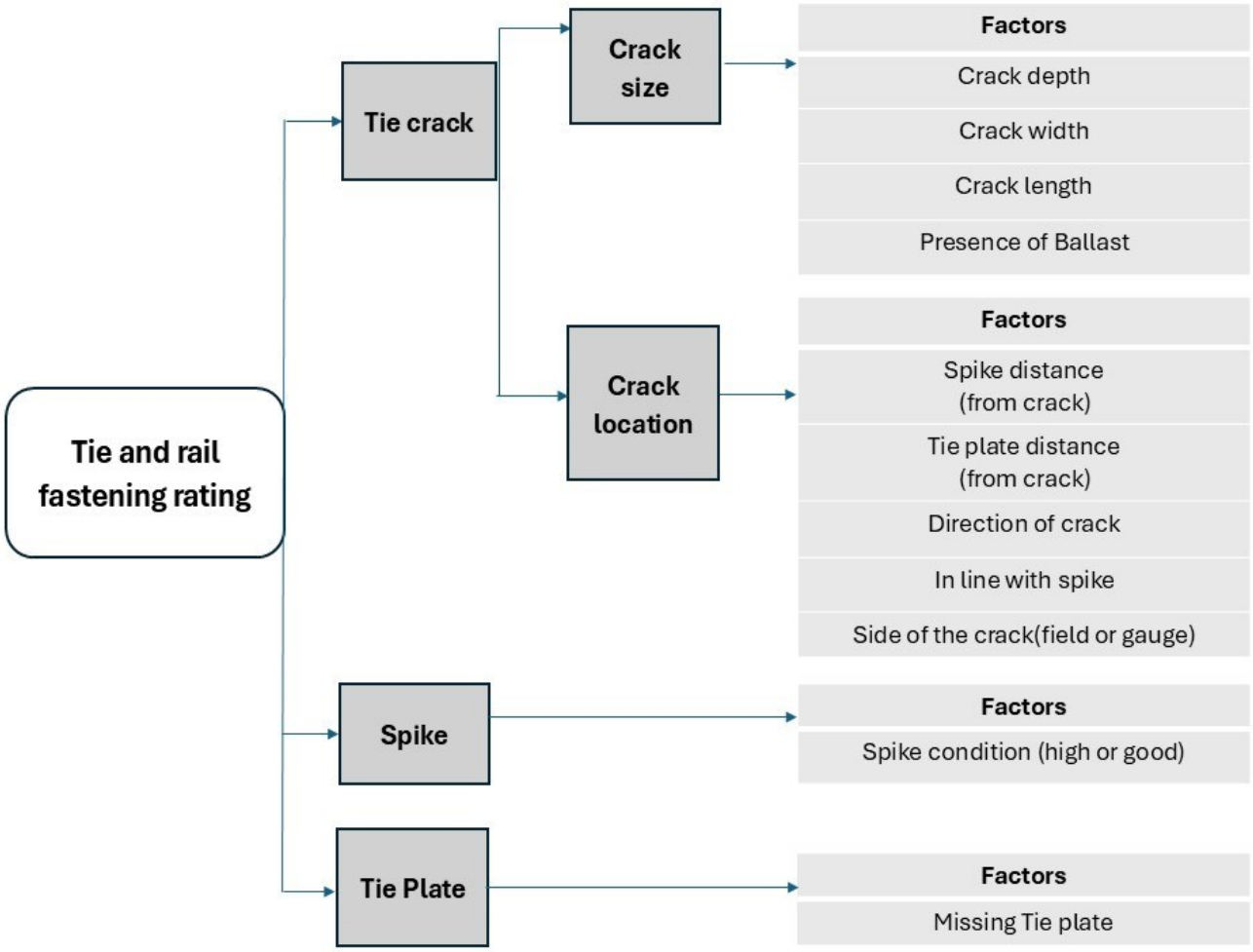


Figure 8: Proposed tie and rail fastening system framework.

3.1.4 Questionnaire Analysis Using the Analytical Hierarchy Process (AHP)

In infrastructure asset management, multi-criteria decision-making techniques are commonly used to make robust decisions by combining technical information with expert opinions. These techniques analyze data and weights of different options to generate a single index representing the asset's condition. This approach enables a comprehensive assessment by combining objective data with subjective expert insights, which helps develop effective management and maintenance strategies (Kabir, Sadiq, and Tesfamariam 2014). The Analytic Hierarchy Process (AHP) stands out among these techniques for its structured approach, which breaks down complex decisions into a hierarchy of simpler sub-problems, facilitating more manageable and accurate analysis. AHP

effectively integrates quantitative and qualitative criteria through pairwise comparisons, capturing nuanced preferences and providing a consistent check to ensure logical coherence. This makes AHP flexible and user-friendly, enabling decision-makers to understand and repeat the process easily. The Analytic Hierarchy Process (AHP) provides objective mathematics to process an individual or group's inescapably subjective and personal preferences in making a decision (Saaty and Vargas 2012b). AHP provides objective mathematics to process an individual or group's subjective and personal preferences in making a decision, making it particularly suitable for infrastructure asset management.

Compared to other methods, AHP offers distinct advantages. Unlike the Delphi method, which is time-consuming and relies on iterative rounds of consensus-building among experts, AHP provides a more streamlined and practical tool for decision-making in project management contexts. AHP's wide application in project management further validates its reliability and effectiveness, contrasting with the Delphi method's limitations (Vidal, Marle, and Bocquet 2011). Similarly, AHP is more accessible and conducive to decision-making consensus than Multi Attribute Utility Theory (MAUT). While MAUT requires specifying utility functions and scaling constants through probabilistic scenarios, often causing frustration due to their complexity(Bard 1992) . AHP uses straightforward pairwise comparisons and a ratio scale to arrive at cardinal rankings of alternatives. This allows for easier integration and synthesis of subjective judgments. Furthermore, AHP provides a clear, logical framework that facilitates the tracing and revising of individual responses, making it more acceptable for decision-makers without extensive training in statistics or utility theory (Bard 1992)

The Analytic Hierarchy Process (AHP) and the Analytic Network Process (ANP) offer unique advantages and applications in decision-making methodologies. AHP is well-known for its simplicity and structured hierarchical approach, making it user-friendly for those without advanced expertise. It is particularly suitable for problems with well-defined criteria and minimal interdependencies and is supported by various user-friendly software tools. The hierarchical structure of AHP ensures consistency in judgments and transparency in decision-making, allowing for easier explanation and justification of decisions to stakeholders. Furthermore, AHP generally requires less data and fewer pairwise comparisons, reducing the time and effort needed for

information gathering and processing and making it practical for various applications (Vaidya and Kumar 2006).

In contrast, ANP is better suited for complex decision scenarios, as it models interdependencies and feedback loops within a network structure. While ANP can handle intricate problems with interrelated factors, it comes with increased complexity and data requirements. The network structure and comprehensive analysis capabilities of ANP require more specialized knowledge and extensive data collection, making it potentially less transparent and more challenging to implement than AHP. Despite these differences, both methodologies play valuable roles in multi-criteria decision analysis, with AHP favoured for its clarity and ease and ANP for its detailed handling of interdependencies (Sipahi and Timor 2010).

Other methods, like the Weighted Sum Model (WSM), lack AHP's detailed comparison mechanism. While the Delphi Method can be insightful, it does not offer the same mathematical rigour or efficiency. Therefore, AHP's ability to incorporate subjective judgments and its comprehensive evaluation capabilities make it the most suitable choice for developing reliable rating system frameworks, such as those assessing tie and rail fastening conditions.This present study utilizes the Analytical Hierarchy Process (AHP), developed by Saaty in the 1980s, to assign weights to rating system framework using the following steps (Mu and Pereyra-Rojas 2017).

In the Analytic Hierarchy Process (AHP), experts make pairwise comparisons between elements or criteria within the same group to establish the relative importance of one factor over another regarding a significant criterion. These judgments help determine the relative importance weights, which are then used to create a pairwise comparison matrix. Pairwise comparisons are made using a questionnaire based on Saaty's (1-9) scale, where 1 represents equal importance, and 9 represents extreme importance. The reciprocal property in AHP ensures that if element x is "j" times more important than element y, then y is 1/j times less important than x. Consistency in these comparisons is evaluated by calculating the consistency index (CI) and the consistency ratio (CR), with a CR less than 0.1 indicating a consistent matrix. Saaty's average random index values, based on matrix size, are used to validate these comparisons, with an example value of 0.89 for a matrix size of 4.

$$\boldsymbol{CR = \frac{CI}{Random\ Index}} \qquad \textbf{(1)}$$

$$\boldsymbol{CI = \frac{\lambda - n}{n-1}} \qquad \textbf{(2)}$$

where

λ is the eigenvalue of the pairwise comparison matrix, and

n is the matrix size.

The tie and rail fastening rating system survey compares different types of defects and the factors affecting their seriousness. Participants were asked to compare different types of defects, such as spike defects, tie plate defects, and tie cracks, as well as considerations like the location and size of the cracks. For example, participants were asked to rate the severity of spike defects compared to tie cracks, with options ranging from "significantly more severe" to "significantly less severe." This setup allows for the collection of detailed comparative data. The complete survey is in the appendix.

Participants were asked to indicate their experience in the field with categories ranging from less than five years to more than 20 years, enabling analysis based on expertise in rail infrastructure issues. They also selected their work location from Western, Central, Eastern, Northern Canada, and International, reflecting diverse environmental and regional factors. Participants identified their areas of expertise, such as maintenance, operations, engineering, safety compliance, or administration, to provide insight into how different perspectives assess defect severity. Additionally, they specified their role in decision-making, whether directly, indirectly, or not involved at all, highlighting how authority influences perceptions of defect severity. Finally, participants indicated their organizational representation, such as short line, Class I, government, academia/research, railway supplier or service provider, and design/consultant. This categorization helped evaluate how organizational roles impact the assessment of rail defects, ensuring a comprehensive analysis of defect severity.

Determining the weights from the survey results using the Analytic Hierarchy Process (AHP) involved respondents first performing pairwise comparisons of criteria and sub-criteria, such as the severity of defects in ties, tie plates, and spikes. These comparisons, rated on a scale of relative importance, formed pairwise comparison matrices. The matrices were normalized, and the eigenvectors (priority vectors) were calculated to determine the relative weights of each criterion. A consistency check was performed using the Consistency Index (CI) and Consistency Ratio (CR) to ensure the comparisons were logically consistent. If multiple respondents provided data, their judgments were aggregated using the geometric mean. The final weights, which were normalized to ensure they summed to one, represented the importance of each criterion and sub-criterion in evaluating tie and rail fastening conditions. This structured approach converted subjective assessments into quantifiable weights, aiding in decision-making for maintenance and risk assessment. For example, consider the survey question: "How do you rate the severity of spike defects compared to tie cracks?" Respondents might have rated spike defects as moderately more severe than tie cracks. This comparison was then translated into a numerical value (e.g., 3 on a scale where 1 meant equal importance and 9 meant extreme importance). These values were used to fill the pairwise comparison matrix, and the eigenvector was calculated to determine the relative weights of spike defects and tie cracks.

3.1.5 Case Study Validation and Sensitivity Analysis

The first step in developing the new tie and rail fastening rating system involved using the Analytic Hierarchy Process (AHP) to determine the relative importance, or weights, of critical factors that influence the overall condition of the fastening system. These factors include the condition of spikes (e.g., spike height), tie plates (e.g., missing tie plates), and ties (e.g., crack size and location). The AHP process helped prioritize these factors based on their impact on track safety and performance, resulting in weights that reflect expert judgment. These weights ensured the most critical factors significantly contribute to the final condition rating.

Once the weights were set, they were applied to the scores given to each factor, ranging from 0 to 10, to measure how serious each defect is. For example, a too high spike would get a score of 10, while a spike in good condition would get a score of 0. This matches the findings from (Gao,

McHenry, and Kerchof 2018a), who highlighted the important role of spikes in keeping ties in good shape. Similarly, a missing tie plate would get a score of 10, while having all the required tie plates would get a score of 0, which aligns with research by (M. Dersch et al. 2019) and the derailment in Alberta (The Transportation Safety Board of Canada 2012), showing the importance of tie plates in preventing damage to ties.

Crack location scores are also given for the crack near the tie plate and spike. When the crack is close to a spike, the score is 10, and the same applies when the crack is close to a tie plate because the crack is more likely to cause damage. This follows evidence that cracks near spikes or tie plates cause more harm to ties, as seen in the British Columbia derailment (TSB 2023), where lifted spikes on the field side contributed to the accident. Cracks on the field side are scored ten due to higher impact on safety, while cracks on the gauge side are scored 0 due to lower severity. Based on their impact on track safety, this scoring system prioritizes cracks near spikes, tie plates, and on the field side.

Three scenarios were evaluated to compare the rating system. In the first scenario, as shown in Table 7, the thresholds for crack measurements are based on values provided by Pavemetrics experts. For crack depth, a depth of 5 mm receives a score of 0, while a depth of 20 mm receives a maximum score of 10. Similarly, a width of 10 mm is scored 0 for crack width, while a width of 50 mm is scored 10. Crack length follows the same approach: a length of 50 mm receives a score of 0, while a length of 600 mm receives a score of 10.

Table 7: Scenario one: Pavemetrics (Industry thresholds)

Factor	Value/class	Score
Spike height	Cut spike	
	≤25 mm	0
	>25 mm	10
	Screw spike	
	≤32 mm	0
	>32 mm	10

Number of Tie plates	0 1 2	10 5 0
Crack depth	5 mm ≥ 20 mm	0 10
Crack width	10mm ≥ 50mm	0 10
Crack length	50 mm ≥600 mm	0 10
The presence of ballast inside the crack	Yes No	10 0
Crack is in line with spike	Yes No	10 0
Crack distance from spike	0 m 0.5	10 0
Crack distance from tie plate	0 m 0.6 m	10 0
The direction of the crack	X Y	10 0
Side of the crack	Field Gauge	10 0

In Scenario Two, referred to as the "without outliers" scenario, the thresholds for crack measurements are based on the mean and standard deviation of values observed in a case study dataset. This approach excludes outliers, defined as over 3 standards deviations from the mean.

In the third scenario, as shown in Table 8, the thresholds for crack measurements are based on the maximum values observed in the case study dataset, including outliers. The scoring system accounts for the most significant defects in the dataset, reflecting not just typical conditions but also the most severe defects. Because only crack depth, width and length have outliers or values beyond the ranges established by Pavemetrics, it is only their scores that change in these scenarios. In all scenarios, the same scoring system is applied to spike height, the number of tie plates, the presence of ballast in cracks, the distance of spikes and tie plates from the cracks, crack direction, and crack location on the tie (field or gauge side). The overall condition rating is determined by multiplying each factor's score by its respective weight, resulting in a weighted score that reflects both the factor's importance and the defect's severity.

Table 8: Scenario with and without outliers

Scenario	Factor	Value/class	score
1. Scenario two without outliers	Crack depth	5 mm	0
		≥ 74.84 mm	10
	Crack width	10mm	0
		≥ 47.32 mm	10
	Crack length	50 mm	0
		≥592.43 mm	10
2. Scenario three with outliers	Crack depth	5 mm	0
		≥ 200 mm	10
	Crack width	10mm	0
		≥ 110 mm	10
	Crack length	50 mm	0
		≥1075 mm	10

To validate the framework, all scenarios were compared to assess the potential impact on maintenance decisions. The Pavemetrics rating system, based on crack width, depth, and length, is called "The Tie Crack Size Condition Rating Scale," while the proposed scale is called the "Tie and Rail Fastening Condition Rating Scale." Both scales were compared by converting the resulting values to a scale from 1 to 4, where 1 represents "Light" and four represents "Very Severe." The classification is as follows: scores of 0, 1, and 2 are categorized as "Light," scores of 3, 4, and 5 as "Moderate," scores of 6, 7, and 8 as "Severe," and scores of 9 and 10 as "Very Severe" A sensitivity analysis was also conducted to evaluate the effect of adjusting the weights assigned to various components in the railway track rating system, including tie crack, tie plate, spike, and sub-factors related to the size and location of cracks. This analysis aimed to understand how changes in these weightings impact the overall condition rating, ensuring that the system is robust and reliable. The weight adjustments were made according to Saaty's (Saaty and Vargas 2012a) Analytic Hierarchy Process (AHP), which provided a structured approach to systematically vary the weights and assess their effects. The analysis began by assigning equal weights to all components and subcomponents, establishing a balanced baseline scenario. The main components considered were tie cracks, tie plates, and spikes. At the same time, subcomponents were further broken down into factors related to crack size (depth, width, length, presence of ballast) and crack location (distance from spike, distance from tie plate, alignment with spike, direction, and side). From this balanced baseline, the next step involved incrementally increasing the weight of the highest-weighted factor within each group and redistributing the weights of the remaining factors accordingly. This allowed for the simulation of various scenarios where one factor became more significant while others became less, creating a diverse range of weighting scenarios. The overall rating of the railway track condition was calculated for all data points, reflecting the new weight distributions. The percentage distribution of the condition rating scale—"Light," "Moderate," "Severe," and "Very Severe"—was calculated for each scenario, providing insights into how the overall ratings fluctuated as the weights were adjusted. The baseline rating, derived from the original AHP-assigned weights, was used as a benchmark to further evaluate the system's sensitivity. Finally, the sensitivity analysis results were interpreted by comparing the percentage changes across all scenarios. This comparison helped identify which components and subcomponents had the most impact on the overall rating.

3.2 Condition Prediction Model

A second case study was conducted to develop a condition prediction model. The data for this study was obtained from an open dataset of track characteristics and defects provided by the "INFORMS 2015 Railway Applications Section Problem Solving Competition."(RAS problem 2015). The Pavemetrics dataset from the first case study was not used here because it lacks key variables. While the Pavemetrics dataset provides detailed information on track conditions, it does not include tonnage, speed, or traffic density, nor changes in track conditions over time, which would be essential for building a reliable prediction model, as highlighted in the literature review. Therefore, the second dataset was more suitable for developing the defect prediction model. A summary of the applicable attributes is provided in Table 9. The overall methodology comprises (1) data understanding and cleaning, (2) modelling, and (3) evaluation. The steps are described in detail in the following sections.

Table 9: Summary of the attributes

Attribute	Description	Type	Min	Max
Line segment number	Every track has a unique line segment number; using this number and the milepost, you can identify any location on the system.	Cat	1	4
Track standard number	Distinguish individual track segments—Mainline & branch numbers: 0=single track, 1-9=multiple train lines.	Cat	0	3
Milepost	The point on the track.	Num	2.436350	444.27274
Total car east	The total number of cars travelling East over a month.	Num	0	54048

Total car west	The total number of cars travelling west over a month.	Num	0	41676
Total train east	The total number of trains travelling East over a month.	Num	0	1136
Total train west	The total number of trains travelling west over a month.	Num	0	1126
Total deflection	The sum of total gross tons travelling across the section.	Num	0	11.31
Test date	The date on which testing was performed.	mm/dd/yy	-	-
Defect number	Every defect detected by a Geometry car gets a unique ID.	ID	2.300000e+01	2.070360e+08
Geometry car	Geometry cars names	Cat	-	-
Defect Tag	The tag is categorized into Yellow or red based on FRA standards.	Cat	-	-
Defect length	Length of defect in feet, as reported by the measurement car.	Num	1	798
Defect amplitude	The maximum size of the defect in inches	Num	-3.59	4.63
Track code	Track codes including tangent, spiral and curve.	Cat	-	-
Class	Track class, representing categories of operating speed	Cat	2	5

	limits for passenger and freight traffic.			
Speed(passenger)	Operating speed for passenger trains (in Mph).	Num	0	90
Speed(freight)	Operating speed for freight trains (in Mph).	Num	11	70
Defect type	Defect type--the geometric defect type such as XLEVEL, SURFACE, DIP.	Cat	-	-

3.2.1 Data Understanding and Cleaning

Understanding the data is a critical first step in data preprocessing. Identifying the necessary data-cleaning processes and selecting appropriate analysis methods is vital. The study's dataset consists of two datasets: 1. track geometry and 2. tonnage. The process starts by thoroughly comprehending the two data sheets and identifying their common attributes.

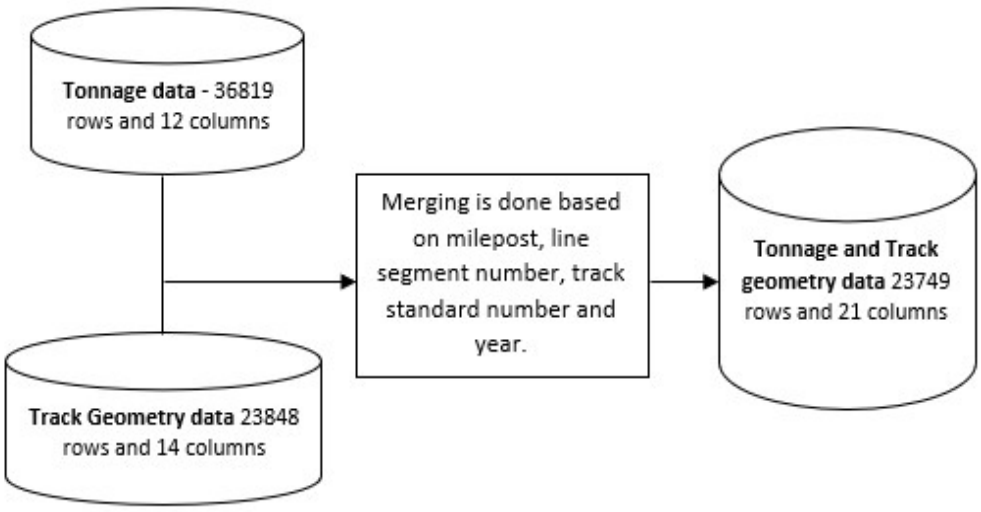


Figure 9: Data merging.

The track geometry dataset includes attributes such as track segment number, milepost start and end numbers, defect amplitude, defect type (cross-level, dip, surface), operating speed, and defect

tags (yellow and red). On the other hand, the tonnage dataset incorporates data on total deflection and tonnage for all track directions (east and west). Additionally, attributes such as milepost, line segment number, track standard number and year are common features in both datasets. These shared attributes were utilized to merge the data, as depicted in Figure 9.

After merging the datasets, a thorough check for missing values was conducted, and none were found, indicating a complete dataset. Additionally, a validation check was performed to ensure the accuracy of the data merge. This involved verifying that the milepost numbers in the merged dataset fell within the range defined by the milepost start and end values. This step was essential to ensure that the data from both datasets were accurately aligned, confirming that the merged records correctly corresponded to their intended track segments. By validating the milepost numbers, we ensured that each record was placed in the right location, which is crucial for reliable analysis. Next, non-relevant attributes such as month, year, and milepost start, and end were removed from the analysis. These attributes were optional for the specific analyses conducted and removing them helped streamline the dataset. The cleaned and merged dataset and a summary of its attributes are provided in Table 9.

The categorical variables were encoded into numerical values to make the data compatible with statistical models and analysis algorithms, which require numerical inputs. Encoding the variables simplifies their interpretation by algorithms, enabling efficient computation and comparison. For example, the defects—dip, surface, and level—were encoded as 0, 1, and 2, respectively, to assign each defect a distinct numerical value for clear differentiation. Similarly, the yellow and red defect tags were encoded as 0 and 1 to represent levels of tags, allowing the model to distinguish between conditions that require caution and those needing immediate attention. Label encoding was chosen for its simplicity and clarity in the model. However, other encoding techniques, such as one-hot encoding, could have been used, where each category would be represented as a separate binary column (e.g., "dip," "surface," and "level" would each have their column with a value of 1 or 0). One-hot encoding is particularly useful when categories are not ordinal or when there are many distinct categories(Alakh 2024). In this case, label encoding was sufficient because the categories represent distinct but comparable conditions, and the model can easily differentiate between them without needing a more complex encoding scheme.

3.2.2 Modelling and Evaluation

Correlation analysis has been performed on data to realize the relationship between the attributes and the targets. Since correlation analysis reveals the relationship between numerical attributes, the converted categories were used in the analysis for categorical variables. The correlation analysis was performed on the merged and cleaned dataset using Python. The correlation matrix is created by dividing each element of the covariance matrix by the product of the standard deviations of the corresponding variables. It provides information about the relationships among different sets of variables. The Pearson correlation is a commonly used method for creating a correlation matrix. The Pearson product-moment correlation can be calculated using equation (3). In this correlation, the values in the matrix range from -1 to 1(David Nettleton 2014). Larger absolute values indicate a stronger relationship between the two variables, with the sign indicating whether the relationship is direct or inverse. Values close to zero indicate little to no correlation between the variables.

$$\boldsymbol{r_{xy} = \frac{cov(x,y)}{\sigma_x\ \sigma_y}} \qquad \textbf{(3)}$$

Where:

r_{xy}, Pearson correlation coefficient between variable x and y.

After conducting a thorough correlation analysis to identify significant relationships between the variables, several machine learning algorithms were employed for classification and regression tasks. Simpler algorithms, such as Logistic Regression for the classification of multiple linear Regression and Decision Trees for Regression, were initially applied to establish baseline models. These simpler models provide insight into the relationships between the variables but often lack the complexity needed for more intricate prediction tasks. As mentioned in the literature earlier, linear regression has been widely applied to infrastructure deterioration modelling. For example, (R. Liu, Xu, and Wang 2010) used linear Regression to model track deterioration by incorporating factors such as tonnage, speed, inspection dates, and track surface irregularities like gauge, cross-level, alignment, surface, and twist. Similarly, (Guler, Jovanovic, and Evren 2011) employed linear Regression to model the deterioration of railway tracks based on attributes such as gradient, curvature, speed, age, rail type, and environmental factors like floods and landslides, predicting various track irregularities including twist, gauge, alignment, and cant. On the classification side,

(Soleimanmeigouni et al. 2020) applied Binary Logistic Regression to predict track geometry defects, specifically UH2 defects, using features like the standard deviation and kurtosis of the longitudinal level, time intervals and defects exceeding the planning limit.

Logistic Regression is a widely used supervised learning algorithm for binary and multiclass classification problems. It estimates the probability that a given input belongs to a particular class by modelling the relationship between the input features and the log odds of the output (Hosmer, Lemeshow, and Sturdivant 2013). The model applies the logistic function, also known as the sigmoid function, to the weighted sum of the input features, transforming it into a probability between 0 and 1. For binary classification, logistic Regression predicts the probability of one of two possible outcomes, such as predicting the presence or absence of defects in infrastructure. Logistic Regression is valued for its simplicity, interpretability, and ability to model linear relationships between features and the log odds of the outcome, making it practical for many real-world classification problems. However, one disadvantage of Logistic Regression is that it assumes a linear relationship between the input features and the log odds of the output, which may not always hold for more complex datasets (Kleinbaum and Klein 2010).This limitation makes the model less effective when the proper relationship between the features and the target variable is highly nonlinear. More advanced algorithms like decision trees or ensemble methods may perform better in such cases.

Multiple Linear Regression (MLR) extends the basic linear regression model by incorporating multiple independent variables to predict a continuous dependent variable. It assumes a linear relationship between the dependent variable and a set of independent variables, allowing for more complex modelling where multiple factors contribute to an outcome (Draper and Smith 1998). MLR is particularly useful in situations where a single outcome is influenced by several predictors, such as in infrastructure modelling, where factors like track gradient, curvature, age, and environmental conditions can collectively impact deterioration rates or condition scores (Montgomery, Peck, and Vining 2012). The strength of multiple linear regression lies in its simplicity and interpretability. The model allows for easy identification of the relationship between individual predictors and the outcome, making it a widely used tool for prediction and decision-making. However, its effectiveness depends on several key assumptions: linearity (the relationship between the dependent variable and each independent variable is linear), homoscedasticity

(constant variance of the residuals), and the absence of multicollinearity (independence among the predictor variables). Violations of these assumptions can lead to biased or inefficient estimates, limiting the accuracy of the model in real-world scenarios with complex interactions between variables (Kutner, Nachtsheim, and Neter 2004).

Decision Trees are a non-parametric supervised learning method for classification and regression tasks. The model builds a tree-like structure, where each internal node represents a decision based on a feature, branches represent the outcomes of those decisions, and leaf nodes represent the final prediction or outcome (Quinlan 1986). Decision Trees use criteria like Gini impurity or information gain (for classification) and variance reduction (for regression) to decide how to split the dataset at each node. For example, in classification, the Gini impurity measures how "pure" a node is to minimize impurity and make the resulting subgroups as homogeneous as possible (Breiman et al. 2017). One of the strengths of Decision Trees is their interpretability—it is easy to visualize and understand how the model arrives at a prediction. However, without regularization methods like pruning or setting a maximum depth, Decision Trees can overfit the training data, leading to poor generalization on unseen data (Rokach and Maimon 2005). Despite this, Decision Trees are the foundation for more advanced ensemble methods like Random Forests and Gradient Boosting.

To improve performance, advanced ensemble techniques such as Random Forest, XGBoost, and CatBoost were employed for classification and regression tasks. As mentioned in the literature, these models were chosen because they are highly effective and reliable for predicting infrastructure deterioration. (Falamarzi, Moridpour, and Nazem 2019) demonstrated that Random Forest performs exceptionally well in predicting track deterioration with high accuracy and low error, as shown in the Melbourne tram network case study. Similarly, (Amini and Dziedzic 2022) showed that XGBoost, an advanced form of gradient boosting, excels in predicting infrastructure deterioration across different fields. Both models are known for handling complex datasets with multiple input variables, making them ideal for the given dataset and the prediction tasks at hand. Random Forest is a supervised learning approach for classification and regression analysis. It consists of multiple decision trees that work together to make more accurate predictions than a single tree. Each decision tree has multiple nodes, and features in these nodes determine how the dataset should be divided into sub-classes. The method selects internal features to minimize

impurity based on specific criteria. For classification, impurity is based on Gini impurity or entropy, while regression is based on variance reduction. The most significant impurity decrease is used to select attributes as internal nodes assigned the highest weights. The final prediction is based on the average of all trees or majority votes for classification and regression. Since the method is built from multiple trees with random predictors, it can assign weights to each feature and determine important features for prediction (Cooper, Kotys-Schwartz, and Reamon 2012). Furthermore, the random forest method is not affected by multicollinearity.

The Gini impurity measure is used in decision tree algorithms to determine the best split starting from a root node and subsequent splits (Steven Loaiza 2020). It measures how effectively a split separates the total samples of binary classes in a particular node. This criterion for a target variable with C classes can be formulated as equation (4).

$$\boldsymbol{Gini = 1 - \sum (p_i)^2} \qquad \textbf{(4)}$$

Where:

C, Number of classes; and

p(i), probability of picking a datapoint with class i.

Entropy impurity measures the amount of variance in data. This measurement can be expressed as an equation (5).

$$\boldsymbol{Entropy = - \sum p_i \,.\, \log 2\, p_i} \qquad \textbf{(5)}$$

Extreme Gradient Boosting (XGBoost) is a robust supervised machine learning algorithm used to solve regression and classification problems (Long et al. 2023). Like Random Forest, XGBoost can effectively identify the most critical features of the target variable. In XGBoost, the main objective is to combine predictions from multiple simple models to predict a classification or regression target accurately. This is achieved by combining and training several trees in the model. The XGBoost training process involves iteratively adding new trees to forecast the errors or residuals of previous trees. These new trees are then integrated with the previous ones for the final prediction. XGBoost is an iterative process where residuals are calculated during each iteration, and subsequent predictors are adjusted to optimize a specific loss function.

CatBoost, a gradient-boosting algorithm specifically designed to handle categorical data efficiently, was also explored. Unlike traditional gradient boosting methods that require extensive preprocessing of categorical variables (such as one-hot encoding), CatBoost is particularly adept at managing these variables natively. It uses a novel approach called target-based statistics to transform categorical features into numerical values based on the target variable, which helps prevent information leakage and ensures better generalization. This approach allows CatBoost to efficiently process high-cardinality categorical features without requiring heavy preprocessing, which is often computationally expensive and prone to overfitting. One of Cat Boost's advantages is its ability to perform ordered boosting, which avoids bias by ensuring that each data point is processed without information from future data points. This prevents the model from overfitting to the training data, a common problem in standard boosting techniques. As a result, CatBoost tends to be more robust and stable on small or imbalanced datasets compared to other gradient-boosting algorithms (Prokhorenkova et al. 2017). CatBoost provides built-in support for advanced regularization techniques, which improves the model's generalization capability and reduces overfitting. It also handles missing values effectively and offers efficient implementations, making it suitable for classification and regression tasks across various domains, including infrastructure and transportation modelling (Dorogush, Ershov, and Gulin 2018). The algorithm's combination of high performance, native handling of categorical features, and computational efficiency makes it an attractive option for datasets with numerous categorical variables, such as those frequently encountered in infrastructure deterioration prediction.

Logistic Regression, Random Forest, XGBoost, and CatBoost were applied for classification tasks, while Multiple Linear Regression, Decision Trees, Random Forest, XGBoost, and CatBoost were employed for regression tasks to predict each target based on different attributes, as shown in Table 10. The classification models were designed to predict defect tags and types using the attributes associated with each defect. These models categorize defects into specific tags and types, allowing for a detailed understanding of each defect's nature. The regression models forecast the numerical values of defect length and amplitude. The regression models provide a detailed analysis of the defects by predicting these continuous values, such as a defect's extent and amplitude. An approach described in Table 10 The predicted amplitude was an additional input attribute to predict the

defect tag. This strategy leverages the model's predicted values to enhance the accuracy of defect tag prediction, leading to more reliable insights and predictions.

Table 10: Input Attributes to predict the target.

Target	Input attributes
Defect tag	Total car west, total train east and west, total deflection, defect amplitude, class, freight speed, passenger speed.
Defect type	Line segment number, milepost, track standard number, total car east and west, total train east and west, total deflection, class, freight speed and passenger speed.
Defect amplitude	Line segment number, milepost, track standard number, total car east and west, total train east and west, total deflection, class, freight speed, passenger speed and defect type.
Defect length	Line segment number, milepost, track standard number, total car east and west, total train east and west, total deflection, defect amplitude, class, freight speed, passenger speed, and defect type.
Increase in Defect length	Defect amplitude, previous defect length, time gap and defect type.

The dataset included two additional columns to predict the growth in defect length over time. The first column, "time gap," represents the interval between the first and subsequent test dates. This interval is calculated by determining the difference between the test dates, providing insights into the time-based progression of defects. The second column, "previous defect length," records the length of the defect from the prior test date for the same defect number. This approach, along with the unique defect numbers assigned to each defect, facilitates the identification and tracking of recurring defects over time, contributing to a detailed analysis of the defect data. The entire dataset was split into training and testing sets to build and evaluate the models. Various data splits were examined to determine the most effective way to train and test the models. For instance, one approach involved a random split, where 80% of the data was used for training and 20% for testing.

Another approach trained the models on data collected from 2007 to 2012 and tested them on data from 2013.

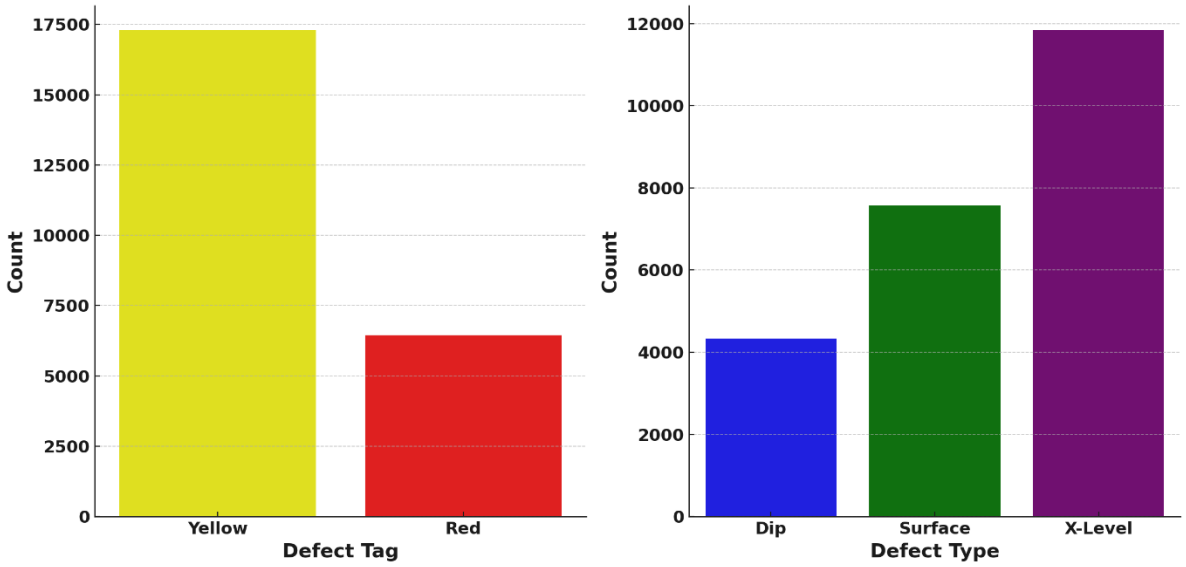


Figure 10: Distribution of the defect tag and defect type

During the modelling process, it became evident that the dataset exhibited a significant class imbalance in the classification tasks, as illustrated in the Figure 10. The left chart shows the distribution of defect tags, where most data points are labelled as non-defective (yellow), while a smaller proportion is labelled as defective (red). Similarly, the correct chart displays the distribution of different defect types (Dip, Surface, and X-Level), with the X-Level defect type being the most frequent and the Dip defect type being the least frequent. This imbalance can bias models toward the majority class, reducing their ability to predict the minority class. To address the class imbalance issue, the following techniques were applied:

Class weighting involves assigning higher importance (or weight) to the minority class and lower importance to the majority class during training. This technique modifies the loss function so that misclassifications of the minority class are penalized more heavily than those of the majority class. The idea is to shift the model's focus toward learning from the underrepresented data points and improving the performance of the minority class, which might otherwise be ignored due to its smaller representation. This approach is efficient in algorithms like Logistic Regression and Random Forest, which support weighted loss functions. By using class weighting, these models can adapt their decision boundaries to accommodate imbalanced datasets better, ensuring that the

majority and minority classes are well represented in the final predictions. (King and Zeng 2001) showed that class weighting can significantly improve prediction accuracy in rare events data by mitigating the bias toward majority classes, a common problem in imbalanced classification tasks.

Oversampling is a popular technique to address class imbalance by increasing the number of instances from the minority class. This can be done by duplicating existing examples or generating synthetic ones. SMOTE (Synthetic Minority Over-sampling Technique) is one of the most widely used oversampling methods. Instead of simply duplicating minority class instances, SMOTE creates synthetic data points by interpolating between existing minority class examples. It generates new, synthetic examples along the line segments that connect neighboring minority class samples in feature space, creating a more diverse representation of the minority class. Oversampling techniques like SMOTE help reduce model bias toward the majority class by providing a more balanced class distribution during training. By creating new instances, SMOTE can also improve the robustness of the model without overfitting specific data points, which can occur with simple duplication methods. (Chawla et al. 2002) demonstrated that SMOTE not only improves classification accuracy for minority classes but also helps prevent overfitting, resulting in better generalization of unseen data.

Cross-validation is a standard technique to evaluate model performance by splitting the dataset into k subsets (or folds), training the model on k-1 folds, and validating it on the remaining fold. The model's performance is averaged across all k iterations to ensure consistency and prevent overfitting. Stratified k-fold cross-validation is a variation of this approach explicitly designed for imbalanced datasets. It ensures that each fold has the same proportion of instances from each class as the original dataset. This is important for imbalanced datasets, as it guarantees that the minority class is well-represented in the training and validation sets. It prevents situations where the model is trained on only majority class data or tested without adequate minority class examples. By maintaining consistent class distributions across folds, stratified k-fold cross-validation provides a more accurate assessment of model performance, mainly when dealing with imbalanced classes. (Wong and Yeh 2020) highlighted the importance of stratification in cross-validation, showing that it significantly improves the stability of evaluation metrics in classification tasks involving imbalanced data.

Classification models were evaluated using the F1 score and Weighted F1 score, while regression models were assessed based on R Square (R^2) and Root Mean Square Error (RMSE). Classification models categorize data into distinct groups and are typically assessed with metrics such as the F1 score and Weighted F1 score. The Weighted F1 score accounts for class imbalance by giving more weight to classes with more instances. It is calculated as:

$$Weighted\ F1\ score = \frac{1}{N} \sum \left(\frac{2(\text{Precision} \times \text{Recall})}{(\text{Precision} + \text{Recall})} \right) \quad (6)$$

Where,

N is the total number of instances, and precision and recall are calculated for each class.

The F1 score offers a more detailed assessment. It is calculated as the harmonic mean of precision and recall. Precision is the ratio of correctly predicted positive observations to the total predicted positives, while recall (or sensitivity) is the ratio of correctly predicted positive observations to all observations in the actual class (Abdusalomov et al. 2021).

The Precision and Recall are formulated as (7) and (8)

$$\boldsymbol{Precision} = \frac{\boldsymbol{TP}}{\boldsymbol{TP+FP}} \quad \textbf{(7)}$$

Where

TP indicates the number of values that are positive and are predicted as positive and

FP indicates the number of negative values and is incorrectly predicted as positive.

$$\boldsymbol{Recall} = \frac{\boldsymbol{TP}}{\boldsymbol{TP+FN}} \quad \textbf{(8)}$$

Where

FN indicates the number of positive values and incorrectly predicted as negative.

Regression models predict continuous outcomes and are typically assessed using the R-Square (R^2) metric. R-Square, also called the coefficient of determination, quantifies the amount of variance in the dependent variable that can be anticipated from the independent variables. It provides insight into how well the model aligns with the data. The R-Square value falls between 0 and 1 (Sharma and Singh 2018). It can be calculated as

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y}_i)^2} \quad (9)$$

Where

y_i represents the actual values, $\hat{y}_i$ represents the predicted values, and $\bar{y}_i$ represents the mean of the actual values.

RMSE measures the average magnitude of the error between predicted and actual values, providing insight into how well the model's predictions match the actual outcomes. Unlike R^2, which indicates the proportion of variance explained by the model, RMSE quantifies the actual error in the model's predictions. RMSE is particularly useful for interpreting the model's performance in the same units as the target variable, making it easier to understand the prediction errors in practical terms. The lower the RMSE, the better the model's predictive accuracy (Chai and Draxler 2014).

The formula for RMSE is as follows:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \quad (10)$$

Where

y_i represents the actual values, $\hat{y}_i$ represents the predicted values, and n is the total number of data points.

Chapter 4. Results

The results section is divided into two key parts: the Condition Assessment Model and the Condition Prediction Model.

4.1 Condition Assessment System

This section contains the outcomes of the Condition Assessment Model. It involves analyzing the survey results using the Analytical Hierarchy Process (AHP) and comparing the new rating system with the case study.

4.1.1 Questionnaire Survey Analysis

The survey was sent to over 50 individuals through the Railway Research Advisory Board (RRAB) in Canada, the Western Canadian Short Line Railway Association (WCSLRA), and some from the American Railway Engineering and Maintenance-of-Way Association (AREMA). These organizations include people from academia, government, Class I railways, short lines, railway suppliers/providers, and consultants. In the end, 21 responses were received. The following section provides a detailed analysis of the survey participants.

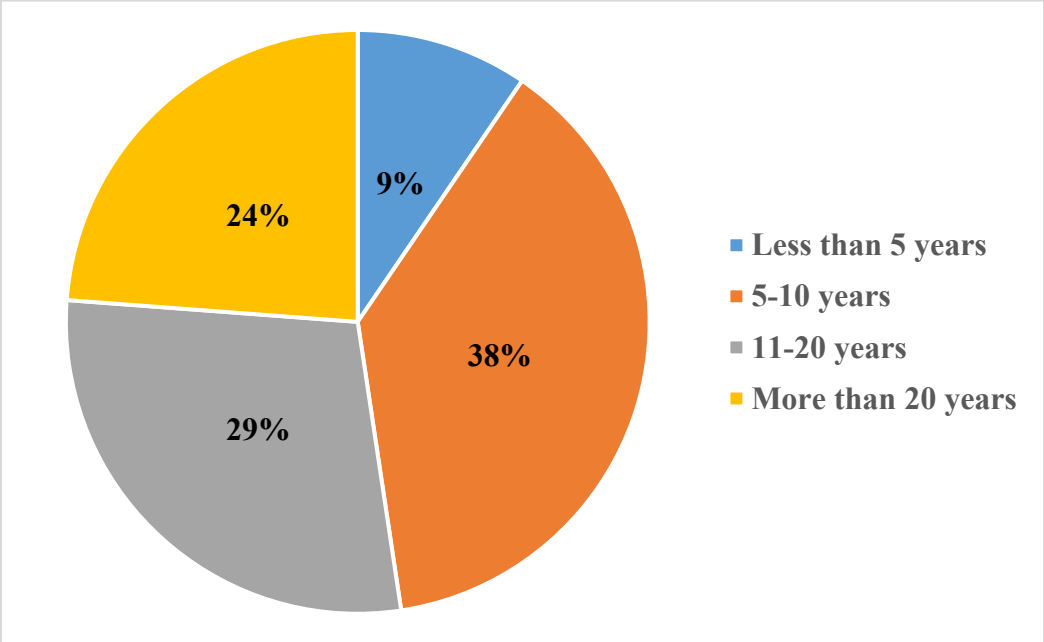


Figure 11: Years of Experience of Survey Respondents.

The survey participants have a wide range of experience in the railway sector, with most having considerable expertise. The largest group (38%) has over 20 years of experience, providing a deep understanding of industry practices, as shown in Figure 11. A significant portion has 11-20 years of experience, contributing solid knowledge of rail infrastructure management. Participants from both international and Canadian regions shared their perspectives, offering diverse insights. Various backgrounds ensure the feedback covers different conditions and challenges, from varying regulations to unique rail infrastructure issues. The high number of participants with more than 20 years of experience suggests that the insights gathered are based on extensive practical knowledge, providing a solid foundation for understanding rail track conditions and management. The range of experience, from less than five years to over 20 years, also brings fresh ideas and long-term expertise. Regarding decision-making involvement, a large group (43%) is directly involved in making strategic decisions about railway track management.

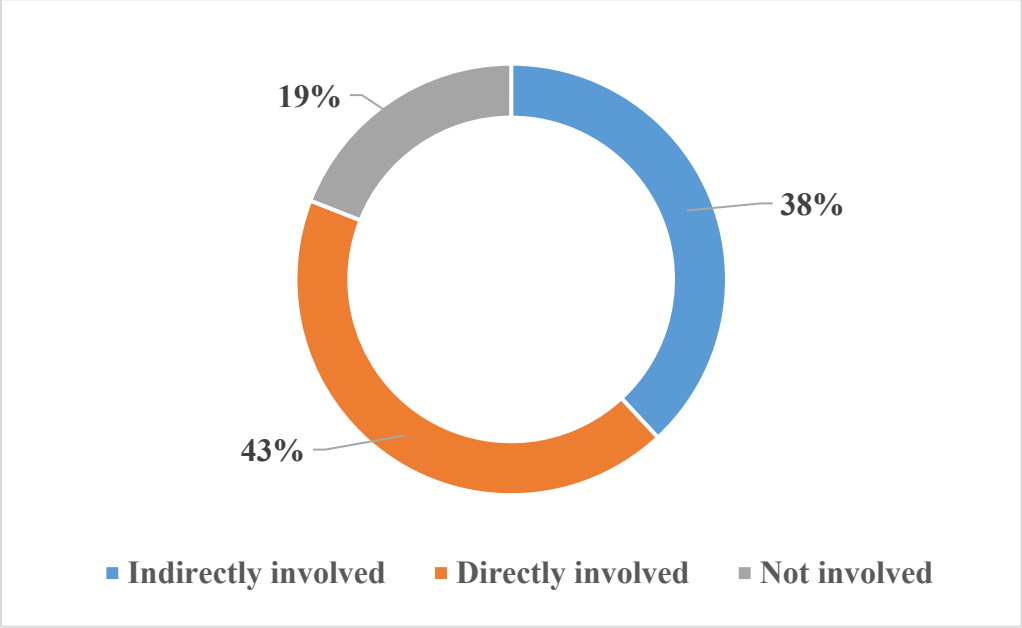


Figure 12: Role in the decision-making of Survey Respondents.

In contrast, others contribute indirectly or provide technical expertise without having decision-making power, as shown in the Figure 12. This mix of roles is important for providing a well-rounded perspective, including those who make decisions, those who influence them, and those who ensure decisions are technically sound. The variety of roles shows the collaborative nature of railway track assessments and management decisions, with input from different levels of expertise.

The balance between those directly involved (43%) and those indirectly involved (38%) suggests that while decision-making is focused, advisory input remains important, ensuring decisions are informed by a wide range of technical and operational considerations.

The survey included participants from diverse professional backgrounds, with representation from design and consultancy firms, academia, research institutions, railway suppliers, Class I and short-line railways, and government roles. A detailed breakdown is provided in the Figure 13. The respondents reported a wide range of expertise in the railway industry, providing insights into rail system management and safety. Maintenance, engineering, operations, safety compliance, and infrastructure design were among the highlighted areas of expertise, demonstrating a comprehensive approach to managing rail safety and addressing defects.

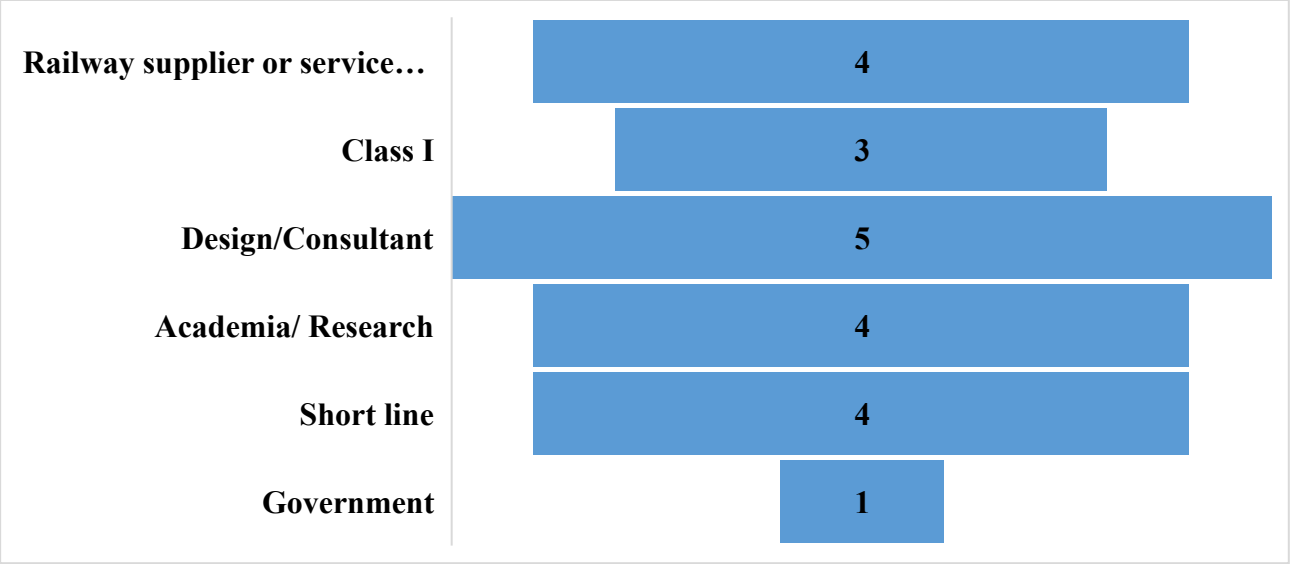


Figure 13: Organization Affiliation of Survey Respondents.

4.1.2 Analytical Hierarchy Process (AHP) Analysis

This section presents the analysis of weight determination for the various factors obtained from the survey for the Condition Assessment Framework using the Analytic Hierarchy Process (AHP).

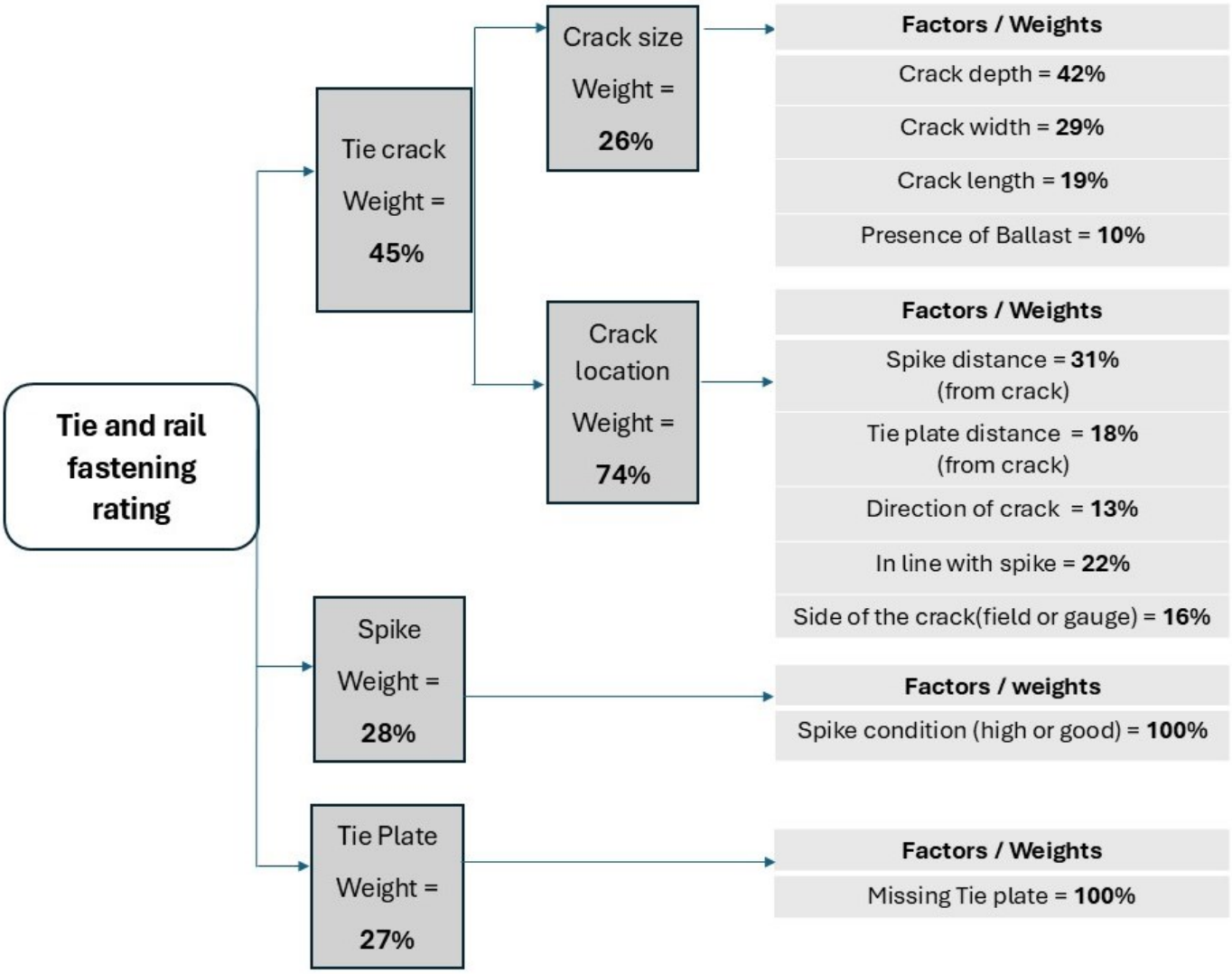


Figure 14: Tie and rail fastening framework with weights

The framework uses the Analytic Hierarchy Process (AHP) to evaluate railway track defects within the tie and rail fastening system and determine track conditions. The framework assigns weights of 45% to tie cracks, 28% to spikes, and 27% to tie plates, as shown in the Figure 14, indicating the importance of these elements for track safety. Tie cracks carry the highest weight, demonstrating their significant impact on the track's long-term stability. Additionally, the framework specifies that crack location is more critical than crack size within the tie crack category. The focus on crack location over size suggests that where a crack form is more dangerous than its size. This makes sense because cracks that appear near important components like spikes

and tie plates can damage the track's stability more. Even tiny cracks close to these key parts can cause problems, while larger cracks farther away may not have as much impact.

4.1.3 Case Study Validation and Sensitivity Analysis

This section presents the results of a new rating system that utilizes data from Pavemetrics. The system's calculations are grounded in weights derived from the Analytic Hierarchy Process (AHP), which computes scores for each factor in the framework. The purpose of the analysis is to compare the proposed rating system's performance with the existing one while also conducting a sensitivity analysis to understand how changes in the system affect the overall ratings.

The results are provided for three distinct scenarios. In the first scenario, the thresholds for crack measurements—such as width, depth, and length—are based on Pavemetrics' threshold. This approach serves as a baseline comparison against the other scenarios. The second scenario, the "without outliers" scenario, establishes thresholds based on the mean and standard deviation of crack measurements observed in a case study dataset. This scenario aims to exclude extreme values, ensuring that the thresholds represent more typical conditions. In the third scenario, with outliers, the thresholds for crack measurements are set according to the maximum values observed in the same case study dataset. Across all scenarios, the remaining factors in the framework—such as spike condition and tie plate presence—remain consistent, ensuring that only the crack measurement thresholds differ between scenarios.

Table 11: Scenario one: Statistical summary of the scores for the Pavemetrics threshold.

Factors	Mean	Stdev	Min	50%	Max
Width	1.78	2.01	0	1.1	10
Depth	5.91	3.32	0	5.73	10
Length	2.29	2.05	0	1.73	10
Presence of Ballast	1.03	3.04	0	0	10
Tie plate Distance from the crack	7.83	2.41	0	8.72	10
Spike Distance from the crack	7.4	2.36	0	8.27	10
The direction of the crack	9.97	0.53	0	10	10
Side of the crack	4.69	4.99	0	0	10

Crack In-Line with Spike	2.1	4.08	0	0	10
Spike height	0.19	1.37	0	0	10
Tie plate score	0.26	1.17	0	0	10

The statistical summary of the Scenario One score, using the Pavemetrics thresholds to evaluate the case study dataset, is shown in Table 11. The Size of the Crack category, which includes attributes such as width, depth, length, and the presence of ballast, generally reveals low average scores, indicating that most cracks in the dataset are relatively small. For instance, the average width of cracks is 1.78, with a standard deviation of 2.01, highlighting that while most cracks are narrow, some variability exists. The presence of ballast is particularly rare, as the data shows very few instances of ballast within cracks (mean 1.03). In the Location of the Crack category, attributes like Tie Plate Distance and Spike Distance display a notable range, suggesting variability in how close cracks are to these critical components. The average Tie Plate Distance is 7.83, showing that cracks often form relatively close to tie plates. The consistently high scores for Direction (mean 9.97) and Side suggest uniformity in the crack orientation, possibly indicating a standard pattern in how cracks develop or are recorded in the field. Regarding Spike Height and Tie Plate Count, the low average scores (mean spike height of 0.19) suggest that most spikes are in good condition, and missing tie plates are not a frequent issue in this dataset. These low values reflect a well-maintained track system in terms of fastening components. The analysis indicates that certain variables, such as crack depth and proximity to tie plates, display significant variability. In contrast, others, like crack direction and spike height, consistently show low or uniform values. This offers valuable insights into the condition of the track and emphasizes areas, such as cracks near critical components, that may require more attention during maintenance.

Table 12: Scenario two and three: Statistical summary of the scores with and without outliers.

Scenario	**Factors**	**Mean**	**Stdev**	**Min**	**50%**	**Max**
1. Scenario two without outliers	Width	1.896308	2.105634	0.00268	1.178992	10
	Depth	1.586685	1.43874	0.004296	1.229954	10
	Length	2.31636	2,062372	0.000369	1.751378	10
	Width	0.742456	0.989704	0.000994	0.437463	10

2. Scenario three with outliers	Depth	0.612418	0.872004	0.001538	0.440513	10
	Length	1.279453	1.339403	0	0.927403	10

The statistical summary of Scenario Two (without outliers) and Scenario Three (with outliers) shows how crack measurements behave in each scenario. In Scenario Two, as shown in Table 12 the average values for width, depth, and length are generally higher than in Scenario Three, indicating that cracks are considered more severe when outliers are removed. For example, the average width is 1.89 in Scenario Two, compared to only 0.74 in Scenario Three, suggesting that the remaining cracks are classified as more severe once extreme values are excluded. In Scenario Three, where outliers are included, the average values for all factors are lower, indicating that the cracks are significantly smaller compared to the outlier values, but not necessarily smaller. The standard deviations are also lower in Scenario Three, meaning there is less variation in crack sizes. For example, the depth in Scenario Three has a standard deviation of 0.87, while in Scenario Two, it is higher at 1.44, showing more variation in crack sizes when outliers are removed. This suggests that the inclusion of outliers skews the data, making the remaining crack sizes appear smaller in comparison.

While scenario three gives a wider view of track conditions, it might hide some more severe issues, potentially delaying necessary repairs because the average scores make the defects seem less severe. When comparing all three scenarios, the Pavemetrics threshold in Scenario One is quite similar to the scores from Scenario Two without outliers, suggesting that this standard method (mean width of 1.78 and mean length of 2.29) effectively highlights severe defects without needing further changes. In contrast, with lower averages, Scenario Three might hide important issues, making it less useful for quick and effective track assessments.

Table 13: score ranges of factors categorized by light, moderate, and severe Ratings

Factors	**Range of the ratings**		
	Light	**Moderate**	**Severe**
Width	0.0025 - 10.0	0.0025 - 10.0	0.72 - 10.0

Depth	0.02 - 10.0	0.09 - 10.0	3.15 - 10.0
Length	0.002 - 10.0	0.00036 - 10.0	3.55 - 10.0
Presence of ballast	0 - 10	0 - 10	0 - 10
Size of the crack	0.15 - 8.24	0.19 - 9.91	2.44 - 10.0
Spike distance	0.0 - 10.0	1.69 - 10.0	8.03 - 10.0
Tie plate distance	0.0 - 10.0	1.87 - 10.0	8.37 - 10.0
In line with the spike	0 - 10	0 - 10	0 - 10
Direction of the crack	0 - 10	0 - 10	10 - 10
Side of the crack	0 - 10	0 - 10	0 - 10
Location of the crack	1.50 - 7.17	2.29 - 10.0	5.77 - 9.96
Spike height	0 - 0	0 - 10	0 - 10
Tie plate	0 - 0	0 - 5	0 - 10

The score distribution across the Light, Moderate, and Severe categories provides key insights into how different factors affect the severity of track conditions as shown in

Table 13. The width, Depth, and Length factors have wide score ranges, but Depth and Length are particularly important in the severe category, where scores start at 3.15 and 3.55. This shows that deeper and longer cracks are more likely to be considered severe. The fact that the Light and Moderate categories have similar ranges suggests that crack size only becomes a major issue when it reaches higher values. This emphasizes the importance of Depth and length in identifying severe track defects. The crack size score in the Severe category starts at 2.44, while smaller cracks tend to stay in the Light or Moderate category. This means that larger cracks are more likely to cause concern and lead to severe ratings. The difference between the categories shows that size plays a

key role in the overall safety assessment, with bigger cracks representing greater risks to track stability.

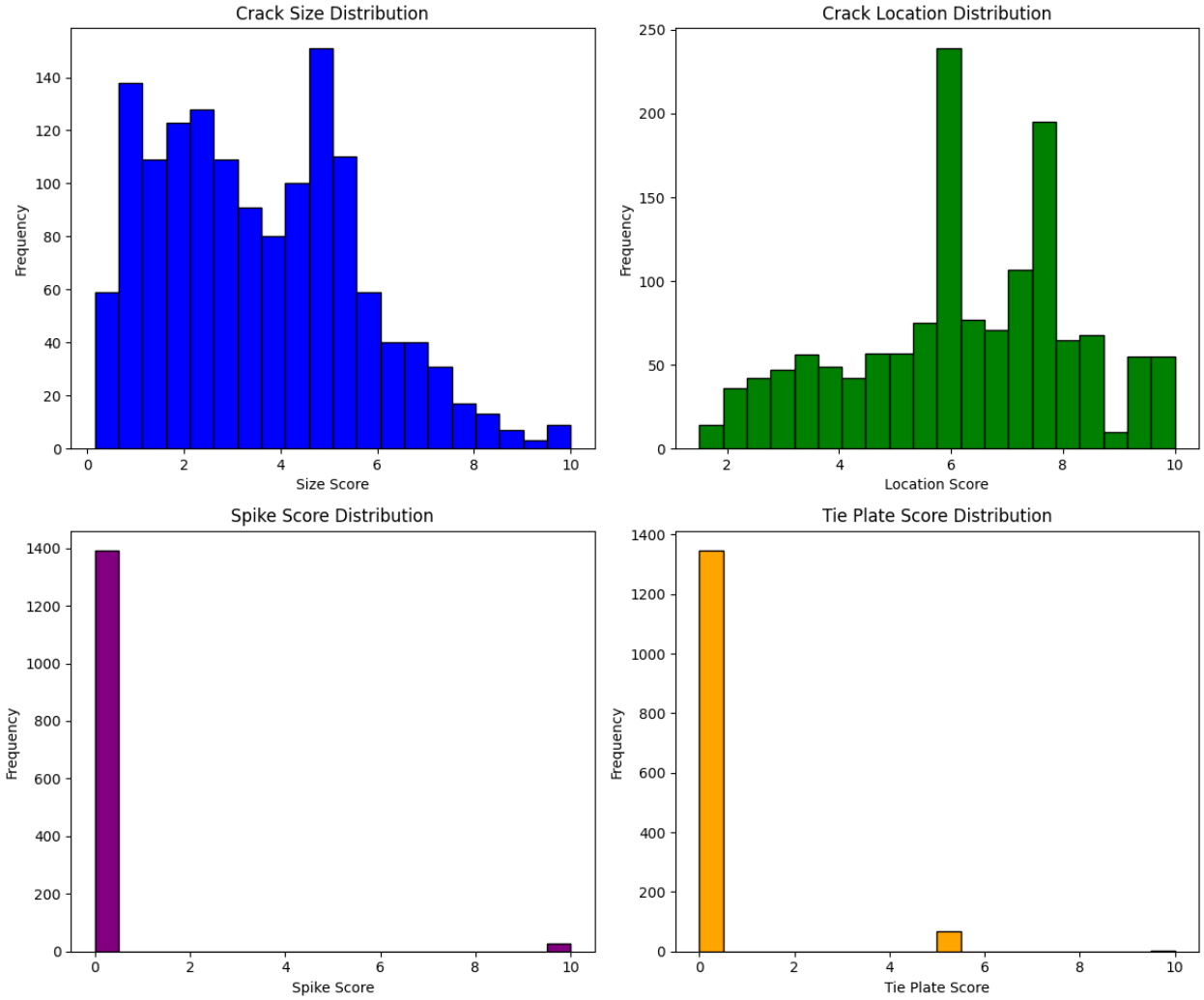


Figure 15: Score distribution of location, size, tie plate and spike

The Spike and Tie Plate Distance scores jump significantly in the Severe category, starting at 8.03 and 8.37, respectively. This suggests that cracks located farther away from these fastenings are more likely to be rated as severe. Cracks in unsupported areas can weaken the track, so their location relative to the spikes and tie plates is critical for evaluating overall track safety. The location of cracks becomes important for severe ratings, with scores starting at 5.77 in the Severe category. Cracks near key areas, like spikes and tie plates, are more likely to lead to severe problems because these areas take on more stress. Even moderate-sized cracks can be classified as

severe when they occur in critical locations, highlighting the importance of considering where the crack is located when evaluating track conditions.

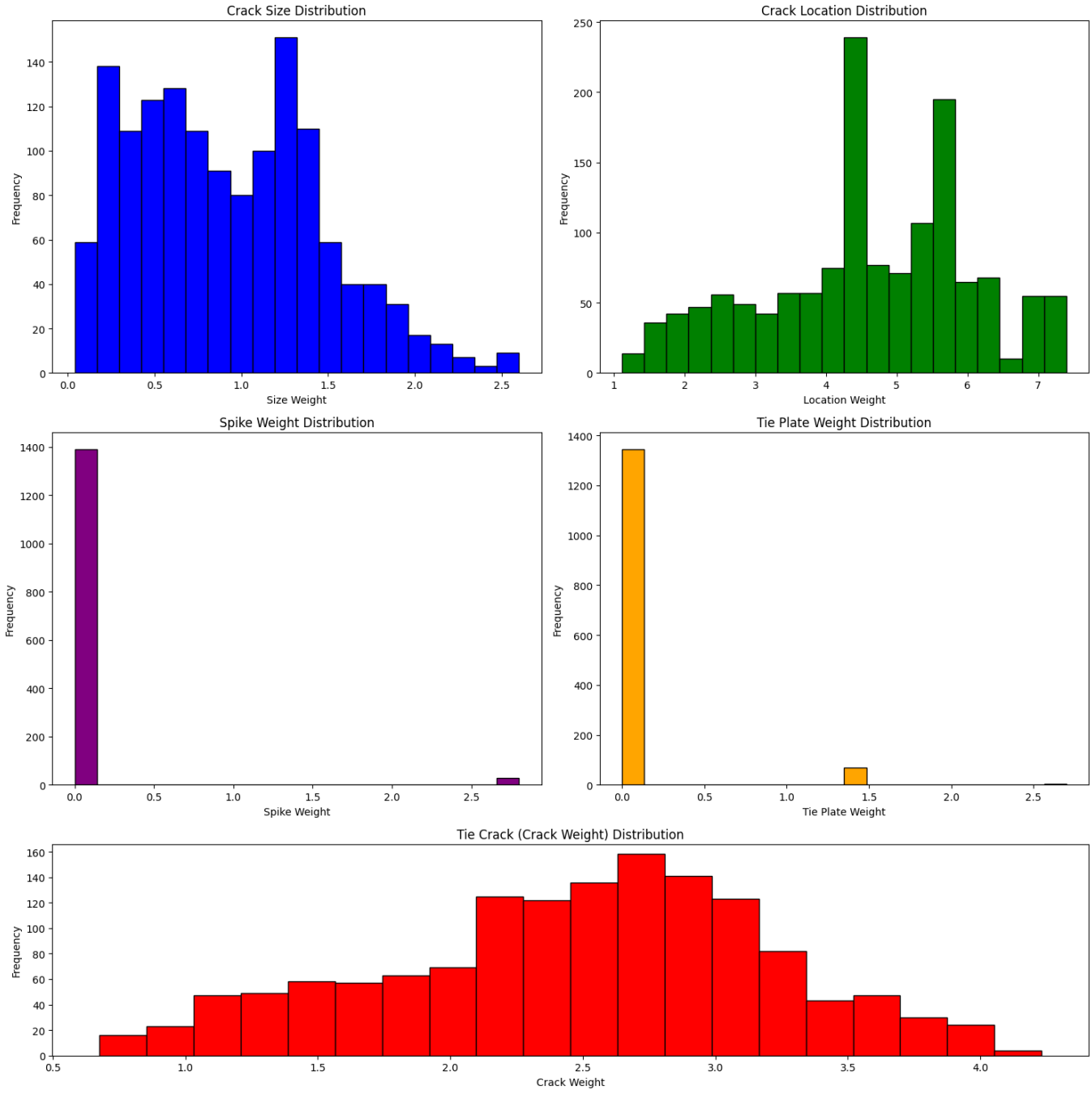


Figure 16: Weight distribution of location, size, tie plate, spike and the sum of the tie crack factors

For Spike Height, the scores in the Severe and Moderate categories range from 0 to 10, while there are no scores in the Light category. This suggests that issues with spike height tend to be more

serious. Since spikes are crucial for holding the rails in place, any irregularities in their height can signal problems with the track's stability. This makes spike height a key factor in detecting more severe track defects. Similarly, Tie Plate scores appear in the Moderate and Severe categories, with scores ranging from 0 to 10 for Severe and 0 to 5 for Moderate. The fact that Tie Plate issues are only seen in these higher severity categories indicates that problems with tie plates, such as missing or damaged ones, are typically linked to more serious track conditions.

The weighted score using the Pavemetrics threshold for the Tie and Rail Fastening Rating, as shown in the Figure 16 , highlight the key factors affecting track condition. In this assessment, raw scores are multiplied by their respective weights to ensure that each factor's impact is proportional to its importance in evaluating track safety.

Crack size is critical in the overall rating, with crack depth being the most significant factor in this category (with most scores ranging between 1.0 and 2.5). This highlights how variations in crack depth can lead to serious safety issues across different sections. Although crack width (scores generally range from 0.0 to 1.5) and crack length (within the same range) play minor roles, they still contribute to the overall evaluation, as even small cracks can worsen if left untreated. The presence of ballast within cracks has the smallest influence (as seen in the lower weight distribution), but it becomes important if it reaches higher levels of severity.

Crack location is the dominant factor in the assessment, with most scores clustering between 4.0 and 6.5, accounting for 74% of the crack-related score. Spike distance (with most cracks near spikes showing scores below 0.5) is particularly concerning, as cracks near spikes can cause rapid deterioration in critical areas. Similarly, tie plate distance (scores mostly around 0.0, but with a few higher outliers) and crack direction play crucial roles in determining the track's safety. Cracks that are in line with spikes or on specific sides of the tie (with minor scores below 1.0) also contribute to the overall evaluation, though to a lesser degree. Cracks account for 45% of the total weight in the Tie and Rail Fastening Rating, underscoring their importance in track safety. The variation in crack-related factors, especially depth and location, suggests that some track sections are more at risk than others, meaning targeted maintenance is essential to prevent further deterioration. While spike condition (most scores around 0.0 but with minor outliers) and tie plate presence (again, mostly 0.0) are also part of the evaluation, their influence is much smaller than

crack-related factors. Spikes and tie plates generally remain in good condition unless there is a significant defect, so they are less frequently a cause for concern.

One important insight from the analysis is the variation in the crack depth and location scores, showing that defects are not evenly spread across the track. This variation highlights the need for focused inspections and repairs, especially in areas with cracks near key parts like spikes and tie plates. These defects can quickly worsen without targeted action, leading to bigger safety risks.Tie plate presence, though showing lower scores and generally being less of a concern, still needs to be monitored to maintain track integrity. The weighted scoring system is a valuable tool for identifying the most severe defects. It provides a clear process for prioritizing maintenance, ensuring critical issues are addressed quickly. The variation in crack-related factors shows the need for regular and detailed inspections to keep the track safe and prevent minor defects from becoming major safety risks.

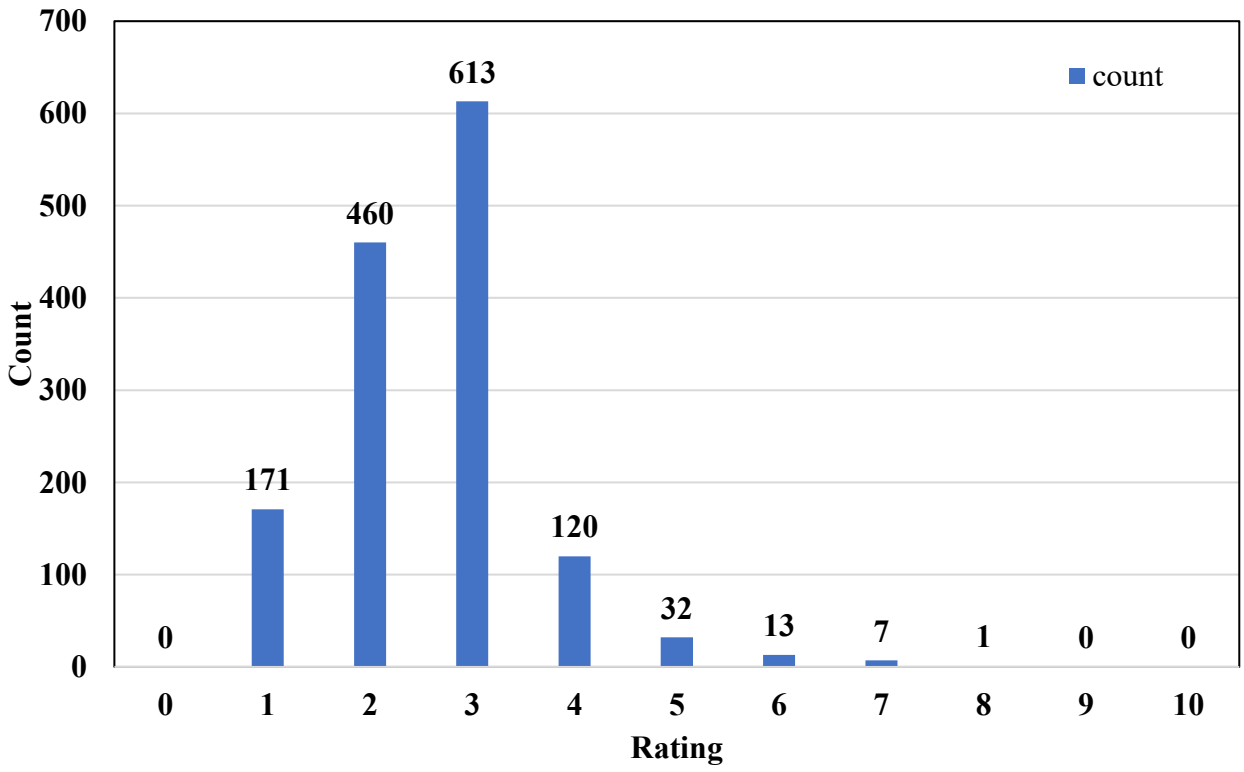


Figure 17: Distribution of the rating for the Pavemetrics threshold

The distribution of the final ratings indicates that most track sections fall within the moderate range, with most ratings between 2 and 3, as shown in Figure 17 . This suggests that, while the tracks are not in critical condition, they still require attention to prevent further degradation. The clustering of ratings in this range implies that maintenance is needed to ensure these sections do not worsen over time. The relatively few sections rated at 1 imply that some areas are in better condition, though they still need to be flawless. These sections may require less immediate attention but should still be monitored regularly to maintain their condition. The inclusion of new factors, particularly crack location relative to spikes and tie plates, has shifted some ratings upward. Cracks near critical components, even if they are not large, now receive more attention, as their proximity to these elements increases the risk of damage. This change in the rating system highlights potential problem areas that may have been overlooked in previous assessments, ensuring that maintenance is directed to sections that are more vulnerable.

It's interesting to note that very few track sections have higher ratings (5 and above), which suggests that severe defects are rare. This indicates that the track infrastructure is generally stable, with only a few isolated areas of concern. However, even a few sections with higher ratings could indicate the need for targeted interventions in specific areas to address more severe issues before they escalate. The absence of sections rated at 0 or 10 is important. No track sections are considered perfect, suggesting that even the best sections still have minor defects or wear that require ongoing maintenance. On the other hand, the lack of any sections rated at ten shows that the track network has no very severe areas, which is a positive indication of its current state. The numerical ratings calculated for the overall condition assessment are transformed into condition scales for easier comparison with the industry rating system developed by Pavemetrics (Tie crack rating). The condition scales are categorized as follows:

- **0, 1, 2:** Light condition
- **3, 4, 5:** Moderate condition
- **6, 7, 8:** Severe condition
- **9, 10:** Very severe condition

This transformation allows for a clearer and more intuitive interpretation of the numerical ratings by grouping them into specific condition categories. By mapping the calculated numerical ratings to these scales, the track condition assessment can be more easily compared with established benchmarks such as those provided by Pavemetrics. Using these scales helps identify track sections requiring different levels of maintenance. For sections rated as "light" (1), routine maintenance such as minor repairs or preventative care would be sufficient. Sections rated as "moderate" (2) may require more focused maintenance, including localized repairs to prevent further deterioration. For "severe" (3) rated sections, immediate corrective actions such as replacing ties or repairing cracks would be necessary to ensure safety. Finally, sections rated as "very severe" (4) would require urgent repairs, possibly including tie replacements, fastening system or significant track realignment, to prevent potential failures.

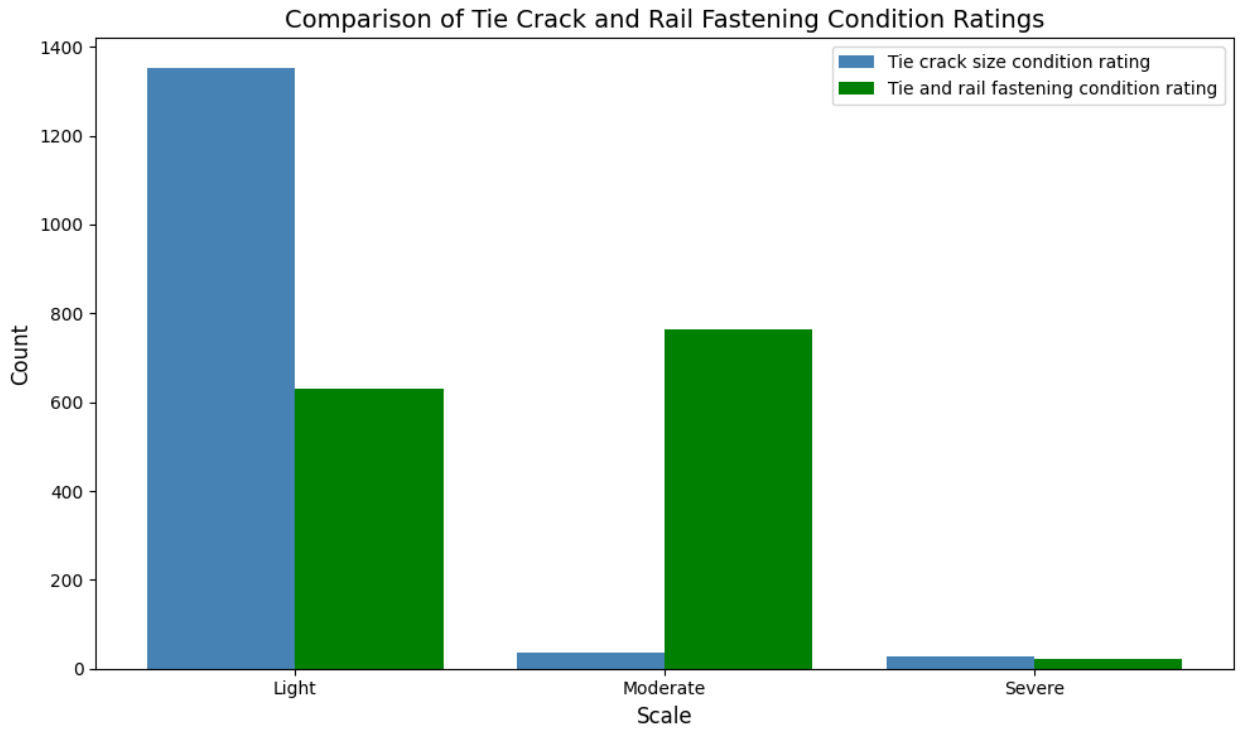


Figure 18: Comparison of Tie crack rating vs Tie and Rail Fastening condition rating

The Tie crack size condition rating scale and the Tie and Rail fastening condition rating scale (proposed scale) differ in assessing track conditions, as shown in the Figure 18. The Tie crack size rating mainly focuses on the size of the cracks, looking at width, depth, and length, which means the assessment is limited because it only considers the severity of the cracks based on these three

measurements. As a result, it may miss other important factors that affect the overall condition of the track. However, the new scale offers a more detailed and comprehensive assessment by considering various factors that influence the condition of the track. It evaluates the size of cracks (width, depth, and length) and the condition of the spikes, considering factors such as missing or broken spikes and spike height. This comprehensive approach is crucial for ensuring the stability and alignment of the track, as missing or damaged spikes can lead to issues such as misalignment and increased wear and tear.

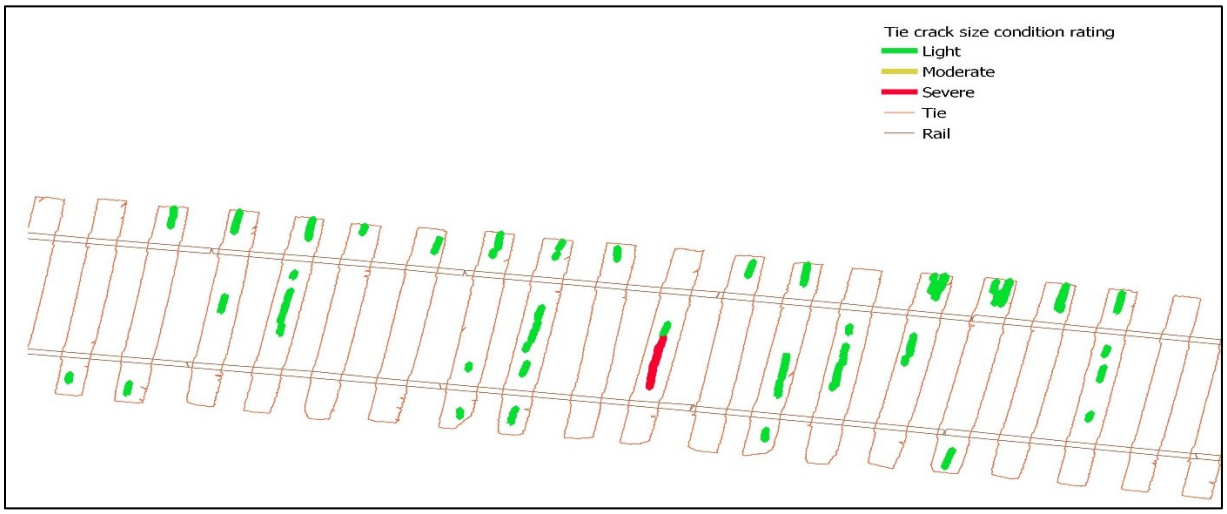


Figure 19: Tie crack size condition rating scale

In Figure 19, the Tie crack size rating scale focuses on the severity of cracks in the ties. Most sections are marked in green, indicating light cracking, and only one section is highlighted in red, showing severe cracks. This provides a narrow evaluation useful for analyzing and addressing cracks in isolation. In contrast, Figure 20 shows the Proposed Scale, which incorporates tie cracks and the condition of spikes and tie plates. This results in a more comprehensive assessment of track health. In this figure, there are noticeably more sections marked in yellow (moderate rating), indicating that when considering the broader context of spikes and tie plates, certain areas require more attention than initially suggested by just the crack data. Additionally, some sections previously rated green (light) in the Tie crack rating scale now appear red (severe) in the Proposed Scale. This shift indicates that while the cracks alone may not be severe, the overall condition of the track, including defects in the spikes and tie plates and the crack's location on the tie plate, has worsened, so urgent repairs are needed in those areas.

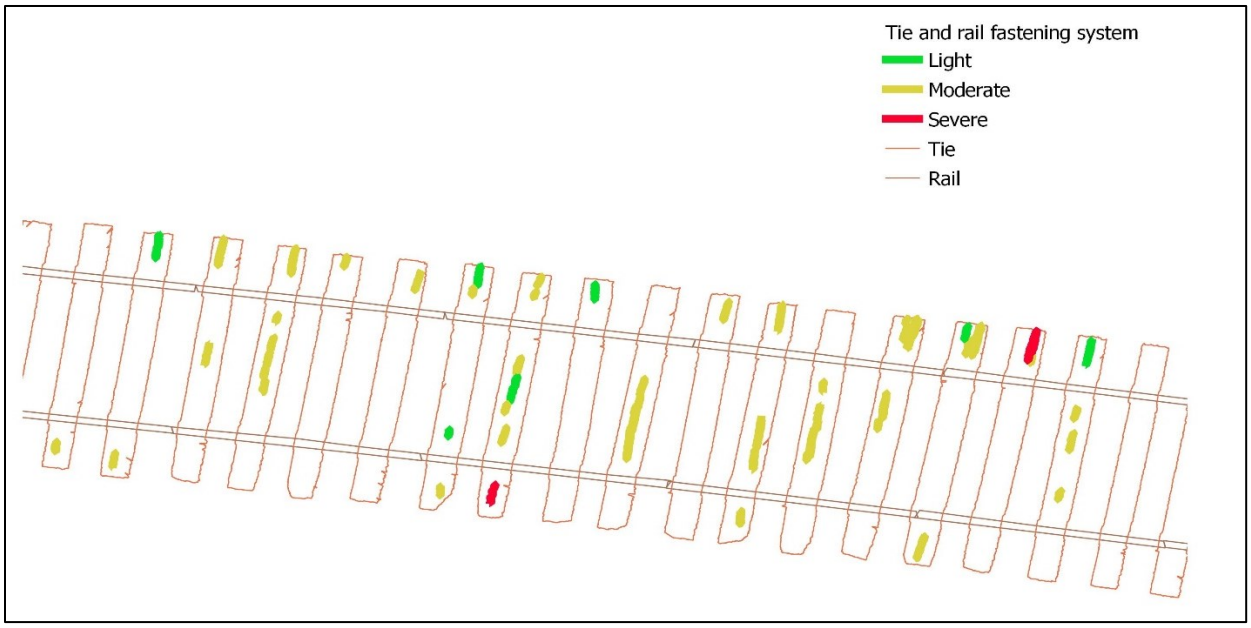


Figure 20: The proposed scale (Rail and tie fastening system)

Thus, the Proposed Scale offers a more holistic view, flagging areas for maintenance that might have been overlooked when focusing solely on tie cracks. This broader perspective is beneficial for overall maintenance planning, while the Tie crack rating scale remains ideal for targeted inspections focused purely on cracks. The following section presents the results of a sensitivity analysis. The analysis compares various scenarios of weight changes and provides valuable insights, including tie crack, tie plate, and spike. Factors such as the location and size of the crack, as mentioned in the methodology, are also considered. This analysis helps us understand how each component impacts the overall ratings, which enables more informed decision-making in condition assessment and evaluation.

A comprehensive sensitivity analysis is presented in Table 14, illustrating how changes in the weights of the Tie Crack, Spike, and Tie Plate impact the overall track condition ratings: Base Weights, Equal Weights, Shifted Weights, and Tie Crack 100%. In the Base Weights scenario, the ratings are primarily moderate. Under Equal Weights, where all components are equally important, more light defects indicate a better overall track condition—the Shifted Weights scenario, which focuses on the Tie Plate, results in even more light defects.

Table 14: Sensitivity analysis of the Tie crack, Tie plate and spike

Scenarios	Weights			Rating percentage			
	Tie crack	Spike	Tie plate	Light	Moderate	Severe	Very Severe
1. Base weights	0.45	0.28	0.27	44.53	53.99	1.48	0
2. Equal weights	0.33	0.33	0.33	85.18	13.69	1.13	0
3. Shifted weights	0.27	0.28	0.45	93.15	6.42	0.42	0
4. Tie crack 100%, others 0%	1	0	0	4.23	41.07	51.87	2.82

In contrast, when the focus is solely on Tie Cracks in the Tie Crack 100% scenario, more defects are rated as severe, even though the actual severity of these cracks might not justify this higher rating. This occurs because the system evaluates only the cracks without considering the condition of other important components, such as spikes and tie plates. The proposed scale provides a more comprehensive and accurate assessment by including these additional elements in the rating system. This approach ensures that maintenance efforts are prioritized more effectively, addressing the overall track condition rather than focusing disproportionately on tie cracks alone. As a result, maintenance decisions can be better targeted to the area's most urgently needed, improving the track's long-term performance and safety. The sensitivity analysis results for scenarios with different weights assigned to Location and Size in the track condition evaluation are shown in the Table 15. The scatter plot represents the Base Weights scenario, where experts have assigned greater importance to crack location over crack size. In this scenario, location is prioritized as a key factor in assessing track conditions, reflecting the belief that cracks in critical locations are more impactful. As a result, the weighting leads to more defects being rated as moderate rather than light, with only a small percentage classified as severe. This shows how emphasizing location influences the overall condition assessment.

Table 15: Sensitivity analysis: Location of the crack and size of the crack

Scenario	weights		Rating percentage		
	Location of the crack	Size of the crack	Light	Moderate	Severe
1. Base weights	0.74	0.26	44.53	53.99	1.48
2. Equal weights	0.5	0.5	65.42	33.24	1.34
3. Shifted weights	0.26	0.74	75.65	23.08	1.27

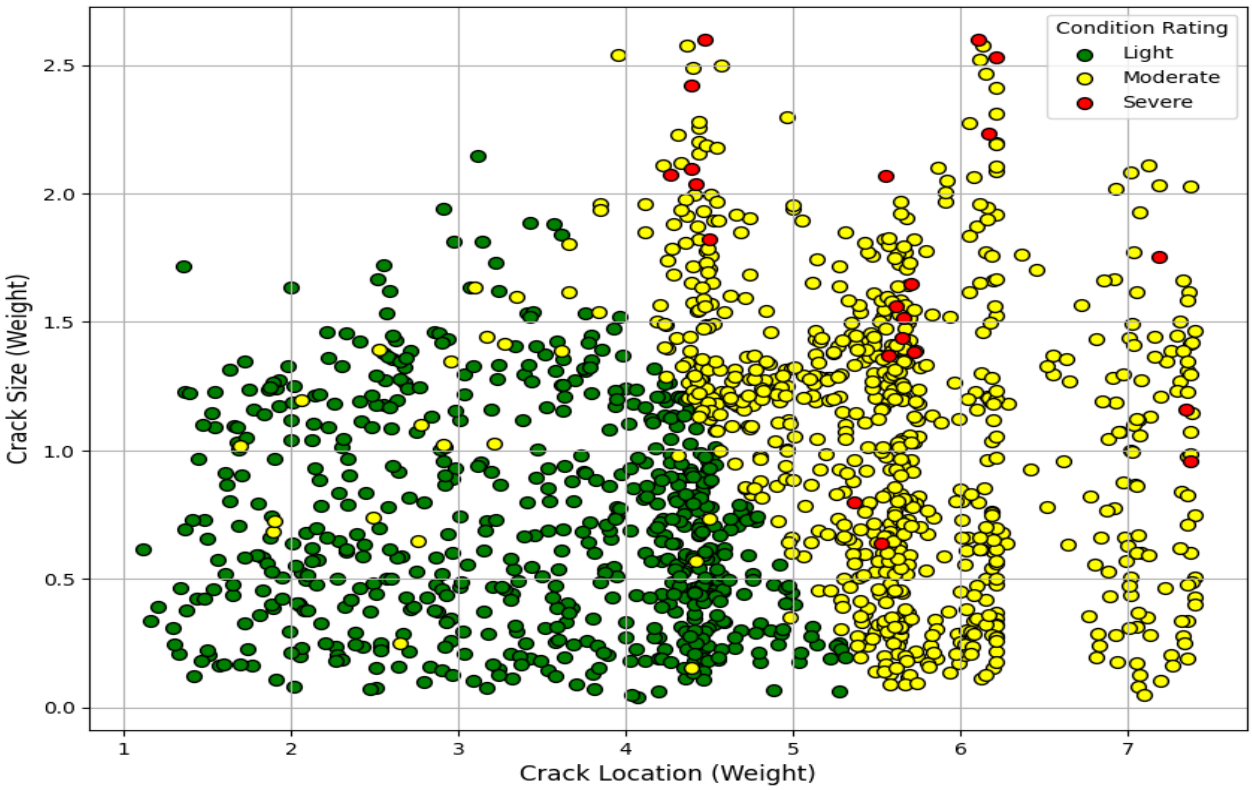


Figure 21: Crack location vs size (base weight scenario)

In the Equal Weights scenario, assigning equal importance to location and size results in more defects being classified as light, suggesting a more balanced assessment with a slight reduction in

moderate ratings. However, even with equal weights for location and size, the condition tends to be rated lighter, even when cracks are located near critical areas such as the tie plates, as shown in Figure 22. This indicates that, despite the cracks being in important positions, the balanced weighting results in less severe condition classifications.

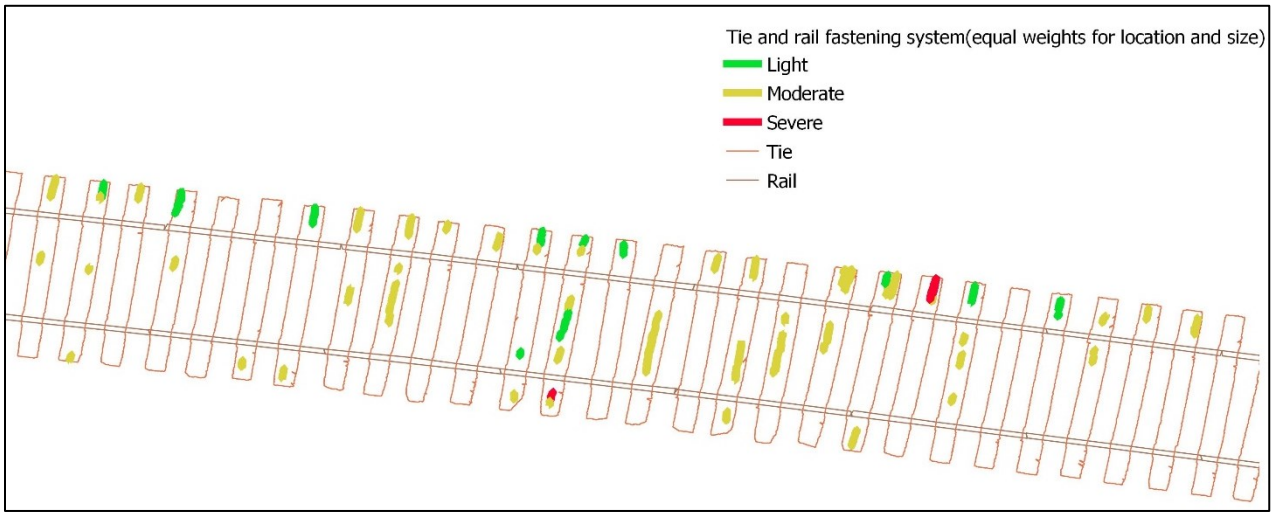


Figure 22: location vs size factors (Tie and rail fastening system condition scale for equal weights scenario)

The Shifted Weights scenario, which emphasizes the size of the crack, leads to the highest number of light defects, showing that when crack size is prioritized, the overall assessment tends to be less severe. This comparison highlights how shifting the focus between crack location and size can influence the severity ratings, with more emphasis on size leading to lighter assessments and more focus on location, resulting in more moderate issues.

Table 16: Sensitivity analysis of the crack size factors

Scenario	Weights				Rating Percentage		
	Depth	Width	Length	Presence of Ballast	Light	Moderate	Severe
1. Base weights	0.42	0.29	0.19	0.10	44.53	53.99	1.48
2. Equal weights	0.25	0.25	0.25	0.25	50.04	48.62	1.34

3. Shifted weights	0.10	0.19	0.29	0.42	53.85	44.81	1.34

The sensitivity analysis results for weights assigned to Width, Length, Presence of Ballast, Spike Distance, Tie Plate Distance, Crack Location, and Direction are shown in Table 16 and Table 17 . In the Base Weights scenario, more focus on depth and spike distance leads to more moderate defects, showing the importance of these factors. In the Equal Weights scenario, where all factors are treated equally, the assessment is more balanced, with a slight increase in light defects. In the Shifted Weights scenario, with more emphasis on ballast presence and crack direction, there is a slight increase in moderate defects, but overall, the ratings do not change much. This shows that adjusting the weights has little impact on defect classification, with depth and crack direction leading to a slightly more cautious assessment.

Table 17: Sensitivity analysis of the crack location factors

Scenario	**Weights**					**Rating percentage**		
	Spike distance	**Tie plate distance**	**In line with the spike**	**Side of the crack**	**Direction**	**Light**	**Moderate**	**Severe**
1. Base weights	0.31	0.18	0.22	0.16	0.13	44.53	53.99	1.48
2. Equal weights	0.20	0.20	0.20	0.20	0.20	40.65	57.87	1.48
3. Shifted weights	0.13	0.16	0.18	0.22	0.31	35.43	63.09	1.48

In summary, the sensitivity analysis across various factors influencing track conditions—including Tie Crack, Tie Plate, Spike, crack dimensions (Width, Length, Depth, Presence of Ballast), and location-related aspects (Spike Distance, Tie Plate Distance, Side of the Crack, and Direction)—reveals important insights into the adaptability and stability of the evaluation framework. The

analysis shows that the framework can detect small differences in track conditions, allowing for a detailed review when needed. The framework can focus on what is most important, such as crack-related factors, fastenings, or the exact location of defects. The analysis confirms that factors like Tie Crack and crack location are important in determining the overall condition ratings, underscoring their importance in maintenance planning and safety assessments. This adaptable framework is reliable and effective for evaluating track conditions and prioritizing maintenance actions based on the most important factors.

4.2 Condition Prediction Model

This section contains the outcomes of the Condition Prediction Model. It involves analyzing the correlation matrix and evaluating the performance of different machine-learning models. The classification models predict defect tags and types, while the regression models focus on forecasting defect amplitude and length.The descriptive statistics of the target features shown in the Table 18 highlights the distribution of the variables. The defect length shows a high variability, with a mean of 12.13 and a standard deviation of 18.03, ranging from 1 to 798, suggesting that while most defects are small, there are few significantly longer defects. Similarly, the Defect amplitude has a mean of -0.06 and a standard deviation of 1.23, with values ranging from -3.59 to 4.63. Regarding class imbalance, the defect tag feature shows a significant imbalance, where the yellow tag appears in 17,298 instances compared to only 6,450 instances for the red tag. Similarly, the defect type is also imbalanced, with the cross-level defect type (possibly the most common defect type) having 11,843 instances, compared to 7,575 for surface and only 4,330 for dip.

Table 18: Statistical description of the targets

Targets	mean	std	min	max
Defect tag	0.271602	0.444795	0	1
Defect type	1.316363	0.76221	0	2
Defect length	12.12591	18.03041	1	798
Defect amplitude	-0.05589	1.231548	-3.59	4.63

4.2.1 Correlation analysis

This section provides the findings of the correlation matrix, which are shown in the Figure 23. The correlation analysis was conducted to identify the relationships between various attributes of the rail track and defect characteristics. It determined which attributes are most strongly linked to defects, enabling the prioritization of key factors in predictive models. Focusing on variables with the most significant impact on targets enhances the accuracy and reliability of predictions regarding rail track conditions. The matrix shows moderate positive correlations between geographic features like line segment number, track standard number, and milepost. This suggests that these features are related sequentially and qualitatively along the track. For example, the line segment number and milepost correlate 0.51, and the track standard number correlates 0.64 with the milepost. This means that higher track standards are found in specific geographic locations. Additionally, the correlation of 0.52 between track standard number and passenger speed suggests that higher track standards are linked to higher speeds.

Regarding tonnage and speed attributes, total car and train metrics (east and west) exhibit extremely high correlations (0.99), indicating redundant measurements that might not provide distinct information for models. These metrics, reflecting the load and usage intensity of the tracks, however, show minimal direct influence on the nature or severity of track defects. Instead, speed metrics like passenger and freight speeds show a nuanced relationship with defects. Passenger speed, for instance, correlates negatively with defect type (-0.18) and amplitude (-0.18), suggesting that tracks frequented at higher speeds are subject to different maintenance standards or material characteristics that reduce certain defects.

Defect characteristics—tag, type, amplitude, and length—demonstrate varying degrees of independence from broader operational and geographic features. Defect tag and type show only moderate correlations with defect length (0.30) and amplitude (0.30), implying that while defects' severity and physical dimensions influence their categorization, these aspects are not heavily dependent on the geographic or operational settings. The independence is particularly notable in defect amplitude, which displays low correlations with most attributes, suggesting that the physical expression of defects might be driven more by localized track conditions or inherent material properties than by external factors.

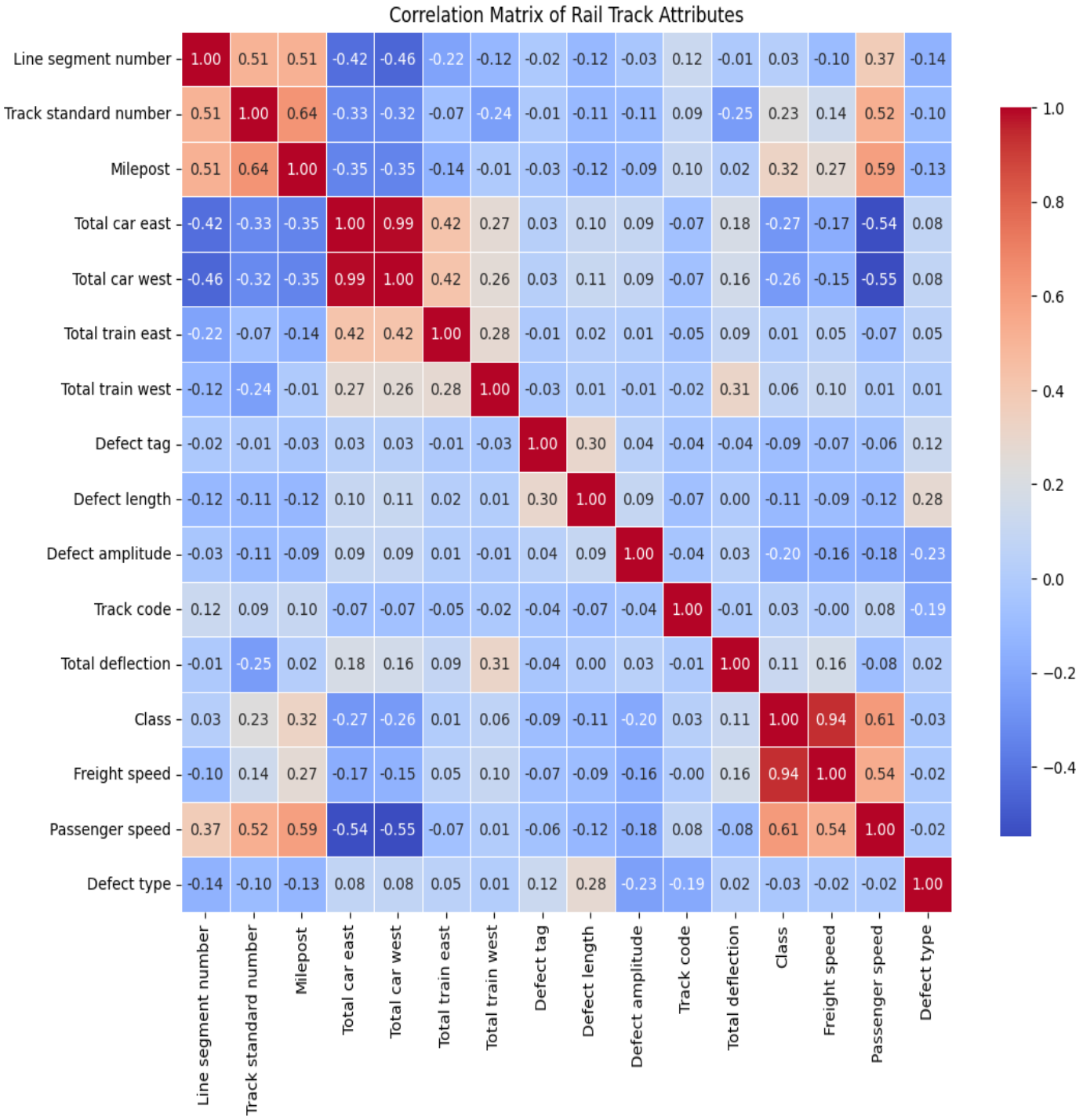


Figure 23: Correlation matrix for the condition prediction model

The analysis shows a strong relationship between the class of track and the speed of freight trains (0.94) and a moderate relationship with the speed of passenger trains (0.54). This suggests that the speeds at which trains operate impact how tracks are classified. It may indicate that there are rules

or design standards to ensure that tracks can safely handle the speeds of trains. The correlation between the speeds of freight and passenger trains (0.54) also highlights that there are common operational factors that influence how tracks are used and maintained.

The analysis shows how different factors relate to rail track defects. Defect tags, which classify defects, are influenced by how severe the defect is, the condition of the track, and how fast trains travel on that track. This helps improve systems that label defects based on their risk and the train's speed. Defect types are linked to how long and severe the defects are and how much train traffic there is. This helps maintenance teams better understand and fix specific types of defects. Defect severity and length are closely related to how much a track is used and how much stress it undergoes. Tracks that see a lot of train traffic and stress are more likely to have severe and prolonged defects. This information is crucial for creating models that predict when and where defects might happen, helping to fix tracks before problems worsen. These findings help target maintenance efforts more effectively, improving the tracks' safety and condition.

4.2.2 Defect tag and defect type detection models

The Classification model results for detecting defect tag show that both Random Forest and XGBoost models perform very well using current defect data, as shown in Table 20. The classification model results for detecting defect tags show that both Random Forest and XGBoost models perform very well using current defect data, as shown in

Table 20. They achieved 0.92 and 0.94 weighted F1 scores, respectively, and had high individual F1 scores, especially for the 'Yellow' category (0.95 for Random Forest and 0.96 for XGBoost). These models demonstrate strong potential for reliable detection. They also performed well when splitting the data based on the test date (2007–2012) to train the model and test on the year 2013, handling changes over time effectively (0.92 weighted F1 score and 0.95 F1 for Random Forest; 0.94 weighted F1 score and 0.96 F1 for XGBoost).

The confusion matrix shows the model's performance in predicting defect tags using Cat boost, as shown in Figure 24, where class 0 represents the "Yellow Tag," and class 1 represents the "Red Tag." The matrix indicates that the model correctly predicted 3,499 instances of the Yellow Tag

(true positives for class 0) and misclassified only eight instances as Red Tag (false positives for class 0). On the other hand, 994 instances of the Red Tag were correctly identified (true positives for class 1). In comparison, 249 instances of Red Tag were mistakenly predicted as Yellow Tag (false negatives for class 1). This suggests that while the model performs well in identifying Yellow Tags, there is some difficulty in accurately identifying all instances of Red Tags, as shown by the false negatives for class 1.

Table 19: Confusion matrix for defect tag prediction using cat boost

	Predicted (yellow tag)	**Predicted (Red Tag)**
Actual (yellow tag)	3499	8
Actual (Red Tag)	249	994

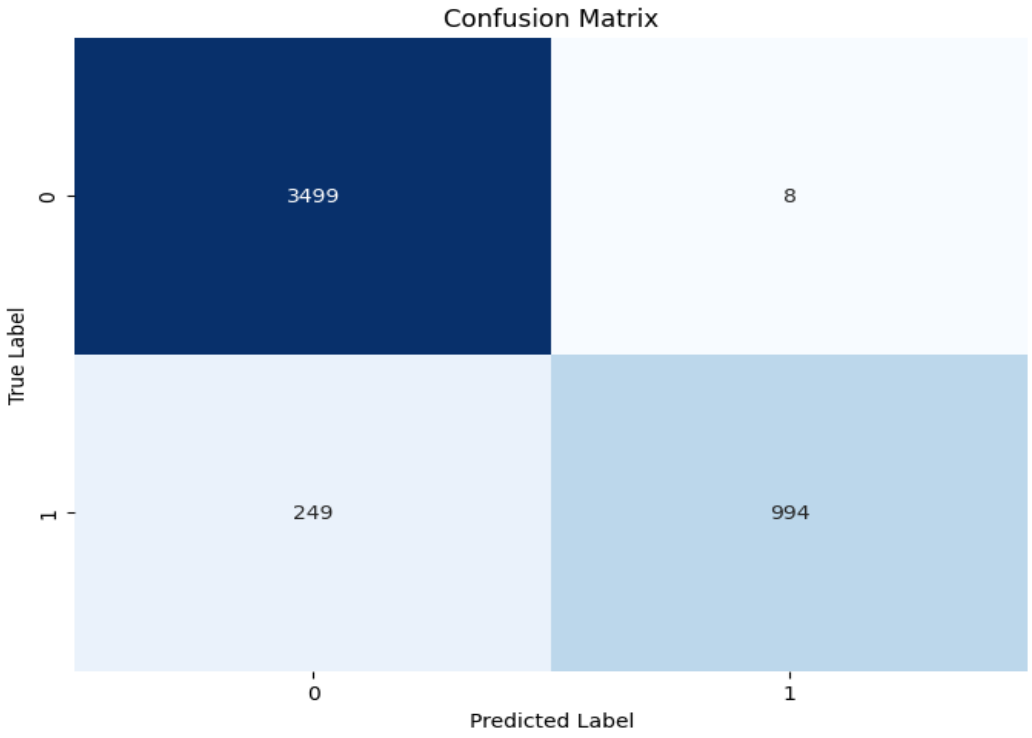


Figure 24: Confusion matrix for defect tag prediction using cat boost

The feature importance analysis highlights that "Defect Amplitude" is identified as the most important factor, followed by "Class" and "Freight Speed." This shows that the physical characteristics of defects play a significant role in influencing the classification outcomes. While "Defect Amplitude" is an important factor in the models' decision-making process, its limitations in predicting 'Red' tags indicate that including other factors or improving the current features could help the models better distinguish between different defect severity levels. Despite these challenges, the models provide a solid foundation for automating defect classification and supporting the consistent assessment of track conditions.

Table 20: Classification Model Results for Defect Tag Detection

Data preparation	Target	Method	F1 – Score (weighted average)	F1-Score (Yellow/Red)
1. Random split 80% training and 20% test data	Defect tag	Logistic regression	0.64	0.75/0.34
		Random Forest	0.92	0.95/0.82
		XGBoost	0.94	0.96/0.88
		Cat Boost	0.95	0.96/0.89
2. Based on test date (2007 – 2012 on training and 2013 on test data)	Defect tag	Random Forest	0.92	0.95 /0.83
		XGBoost	0.94	0.96/0.88
		Cat boost	0.94	0.96/0.88

The classification results for predicting defect types using both Random Forest and XGBoost models show varying levels of accuracy, and F1 scores across different scenarios are provided in Table 22. For Defect Type (Random Split 80/20), XGBoost slightly outperforms Random Forest, with a weighted F1 score of 0.62 compared to Random Forest's 0.63. Regarding individual F1 scores, XGBoost performs better for predicting "X-Level" defects, with a score of 0.74 compared to 0.73 for Random Forest. However, Random Forest performs slightly better in predicting "Dip"

(0.38 vs. 0.36) and "Surface" defects (0.62 vs. 0.58). This suggests that while XGBoost is better at identifying more severe defects like "X-Level," Random Forest may handle less severe defect types more effectively. CatBoost performs similarly to Random Forest and XGBoost in the random split scenario, with a weighted F1 score of 0.62. It performs well in predicting "X-Level" defects, with an F1 score of 0.73, slightly lower than XGBoost but close to Random Forest. However, CatBoost falls slightly behind in predicting "Dip" defects (0.33) and "Surface" defects (0.61), showing that it, like the other models, finds it more challenging to predict these less severe defect types accurately.

In the second scenario, where data is split based on 2007–2012 for training and 2013 for testing, both Random Forest and XGBoost see a slight drop in weighted F1 scores, with Random Forest at 0.57 and XGBoost at 0.58. CatBoost, on the other hand, achieves a weighted F1 score of 0.57. For individual F1 scores, CatBoost performs best for "Surface" defects, with a score of 0.62, higher than Random Forest (0.57) and XGBoost (0.52). However, CatBoost lags for "X-Level" defects (0.64), trailing Random Forest and XGBoost. CatBoost does show better results for "Dip" defects (0.26) compared to Random Forest (0.15) and XGBoost (0.26), indicating it handles this defect type slightly better. While XGBoost tends to outperform Random Forest in overall weighted F1 score and identifying severe defect types like "X-Level," CatBoost shows strengths in predicting "Surface" and "Dip" defects in specific scenarios.

Table 21: Confusion matrix for defect type prediction using random forest

	Predicted (Dip)	**Predicted (Surface)**	**Predicted (X-Level)**
Actual (Dip)	276	469	121
Actual (Surface)	150	1198	167
Actual (X-level)	174	660	1535

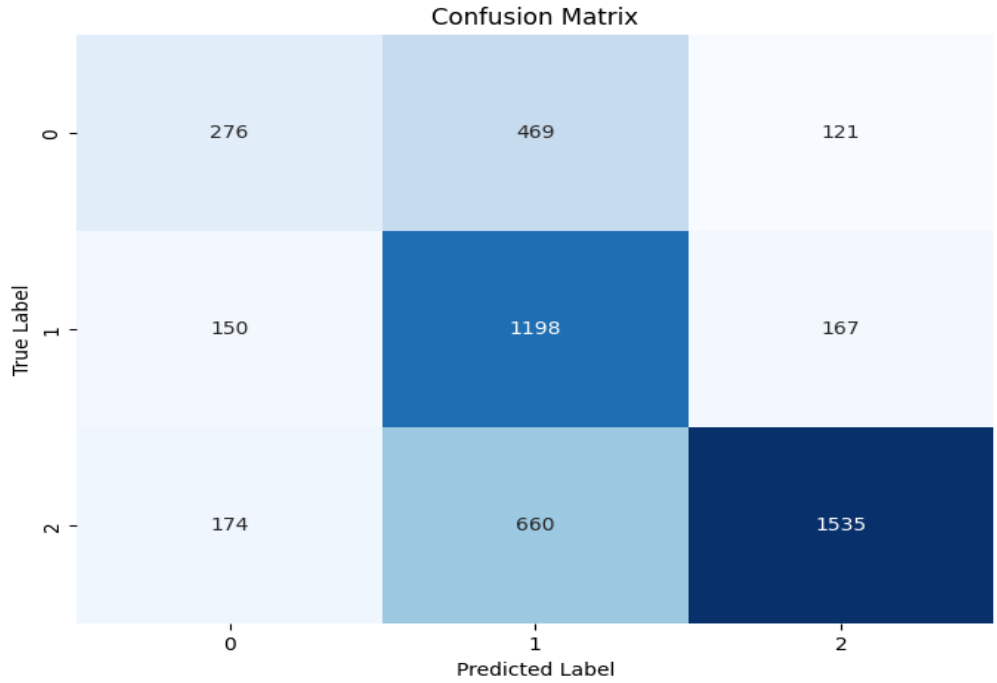


Figure 25: Confusion matrix for defect type prediction using random forest

The confusion matrix for predicting "Defect Type" across three classes—Class 0 (Dip), Class 1 (Surface), and Class 2 (X-level)—shows mixed model performance, as shown in Figure 25. The model struggles with accurately predicting the Dip class (Class 0), with only 276 correct predictions and 469 misclassified as Surface (Class 1) and 121 as X-level (Class 2). For the Surface class (Class 1), the model performs reasonably well, correctly predicting 1,198 instances but misclassifying 150 as Dip and 167 as X-level. The best performance is observed in predicting the X-level class (Class 2), with 1,535 correct predictions, though 174 instances were misclassified as Dip and 660 as Surface. These results indicate that while the model performs reasonably well for the Surface and X-level classes, it faces challenges distinguishing between Dips and other defect types, leading to many misclassifications.

Table 22: Classification Model Results for Defect Type Prediction

Data preparation	**Target**	**Model**	**F1 score Weighted average**	**F1 score Dip**	**F1 Score Surface**	**F1 Score X-level**

1. Random split 80% training and 20% test data	Defect type	Logistic regression	0.51	0.23	0.53	0.61
		Random Forest	0.63	0.38	0.62	0.73
		XGBoost	0.62	0.36	0.58	0.74
		Cat boost	0.62	0.33	0.61	0.73
2. Based on test date (2007 – 2012 on training and 2013 on test data)	Defect type	Random Forest	0.57	0.15	0.57	0.71
		XGBoost	0.58	0.26	0.52	0.72
		Cat boost	0.57	0.26	0.62	0.64

The feature importance analysis reveals that "Class" is by far the most influential factor in predicting defect types, followed by "Line segment number" and "Freight speed." These key features contribute significantly to the models' ability to make accurate predictions. Other factors, such as "Milepost," "Passenger speed," and "Track standard number," also play a role but with less impact. The fact that the models can identify these critical features shows that they can effectively learn from the data and prioritize the most relevant factors to defect prediction. The ability of these models to achieve relatively high accuracy and F1 scores, especially in predicting severe defect types like "X-Level," makes them highly valuable for proactive maintenance. Railway maintenance teams can prioritize repairs and reduce the risk of more extensive failures by accurately identifying the most critical defects. Furthermore, the models' reliance on key features such as "Class" and "Freight speed" provides valuable insights into the underlying factors contributing to defect formation, which can guide future preventive measures.

4.2.3 Defect amplitude and length prediction models

The regression results for defect amplitude, as shown in Table 23, demonstrate the performance of Random Forest, XGBoost, and other models under two different data splitting scenarios: an 80% training and 20% test data split and a split where data from 2007 to 2012 was used for training and data from 2013 for testing. In both scenarios, Random Forest consistently outperformed XGBoost. For the 80% training and 20% test data split, Random Forest achieved an R-square of 88% in training and 82% in testing, with an RMSE of 0.42 (training) and 0.52 (testing). This indicates robust performance, though the slight drop in R-square and increase in RMSE on the test set suggests minor overfitting. On the other hand, XGBoost, while achieving a slightly lower R-square of 83% in training, maintained a relatively close performance in testing with an R-square of 79%, with an RMSE of 0.50 (training) and 0.56 (testing). This demonstrates better generalization compared to Random Forest, as XGBoost's test performance is more consistent with its training performance.

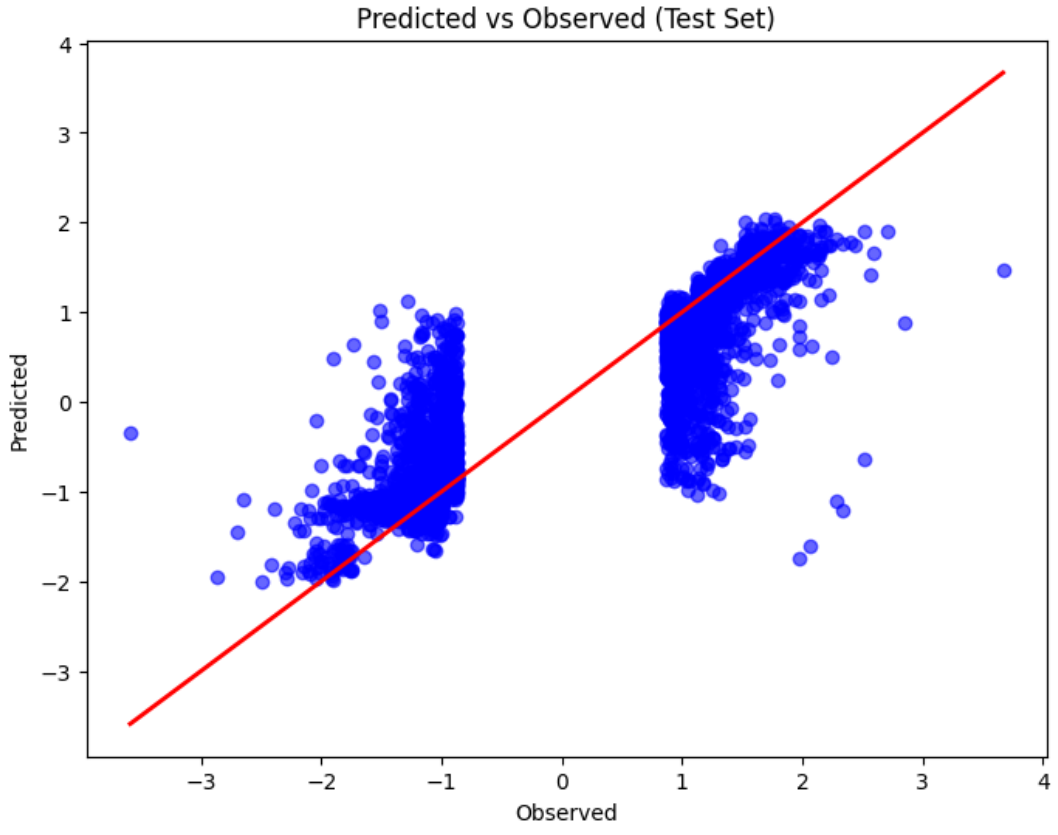


Figure 26: scatter plot for test set to predict the defect amplitude using the random forest

The scatter plot of predicted versus observed defect amplitudes shows that while the model generally follows the trend of actual values, there are some discrepancies and clustering, particularly in specific ranges using the Random Forest, as shown in the Figure. Ideally, the points should align closely along the red diagonal line, representing perfect predictions (where predicted equals observed). In this case, the data points are somewhat clustered around the line, indicating that the model can reasonably predict some portions of the test set. However, visible gaps and deviations, especially around specific ranges of observed values, suggest the model struggles with specific data segments. Notably, the predicted values form two distinct clusters, which implies that the model may have difficulty capturing the full range of the defect amplitude, possibly underfitting or misrepresenting patterns in specific regions of the observed values. Despite these deviations, the model achieved a strong R^2 score of 82%, indicating that the model well captures 82% of the variance in defect amplitude.

CatBoost performed similarly to XGBoost, with an R-square of 83% on the training set and 78% on the test set and corresponding RMSEs of 0.50 (training) and 0.57 (testing). This result indicates that CatBoost also generalizes well but slightly underperforms compared to both Random Forest and XGBoost in terms of test accuracy. However, its performance remains competitive, especially in reducing the training-test gap, as seen in its slightly lower increase in RMSE from training to testing.

In the second scenario, where data from 2007–2012 was used for training and 2013 for testing, Random Forest performed well on the training set with an R-square of 78%, but its performance dropped to 62% on the test set, with RMSEs of 0.57 (training) and 0.75 (testing). This larger increase in RMSE suggests overfitting to the training data. CatBoost, with an R-square of 76% in training and 66% in testing, demonstrated better generalization than Random Forest, with lower RMSEs of 0.59 (training) and 0.72 (testing), showing that CatBoost outperformed Random Forest in this scenario. XGBoost, with 72% R-square on the training set and 65% on the test set, had slightly higher RMSEs of 0.64 (training) and 0.72 (testing), indicating stable performance but slightly behind CatBoost in terms of test accuracy and error reduction.

Table 23: Regression Model Results for Defect Amplitude Prediction

Data preparation	Target	Method	R square in % Train / Test	RMSE Train / Test
1. Random split 80% training and 20% test data	Defect amplitude	Multiple linear	11 / 11	1.16 / 1.15
		Decision trees	90 / 82	0.37 / 0.52
		Random Forest	88 / 82	0.42 / 0.52
		XGBoost	83 / 79	0.50 / 0.56
		Cat Boost	83 / 78	0.50 / 0.57
2. Based on test date (2007 – 2012 on training and 2013 on test data)	Defect amplitude	Random Forest	78 / 62	0.57 / 0.75
		XGBoost	72 / 65	0.64 / 0.72
		Cat Boost	76 / 66	0.59 / 0.72

Decision Trees, while showing strong training performance with an R-square of 90%, dropped to 82% on the test set, with RMSEs of 0.37 (training) and 0.52 (testing), indicating strong training performance but a larger gap between training and testing, suggesting higher sensitivity to overfitting compared to Random Forest and CatBoost. On the other hand, Multiple Linear Regression performed poorly, with R-squares of 11% on both training and test sets and RMSEs of 1.16 (training) and 1.15 (testing), showing that simpler models struggle to capture the complexity of defect amplitude prediction. The feature importance analysis shows that "Defect type" is the most influential factor in predicting defect amplitude, with an importance score exceeding 0.6. This indicates that the type of defect plays a crucial role in determining the amplitude. Other key features include "Class" and "Freight speed," which also significantly impact predictions, though not as dominant as defect type. Factors like "Milepost," "Passenger speed," and "Total deflection" contribute to a lesser extent. Meanwhile, features such as "Total train east," "Total car west," and "Total train west" show minimal importance in influencing the prediction outcomes.

The regression results for defect length prediction across three scenarios are presented in the Table 24. In the Random Split (80% Training, 20% Testing) scenario, Random Forest achieved an R-square of 65% on the training data and 56% on the test data, indicating a reasonable balance between training and testing performance. However, the RMSE values of 10.86 (train) and 11.02 (test) suggest there is room for improvement in reducing the prediction error. XGBoost performed slightly better on the training data, with an R-square of 72%, but its test R-square remained the same at 56%, with corresponding RMSE values of 9.74 (train) and 11.08 (test), showing similar levels of overfitting as Random Forest. CatBoost had a training R-square of 65% and test R-square of 56%, with RMSEs of 10.80 (train) and 10.97 (test), similar to Random Forest in terms of R Square but a slight difference with the RMSEs of 10.80 (train) and 10.97 (test).

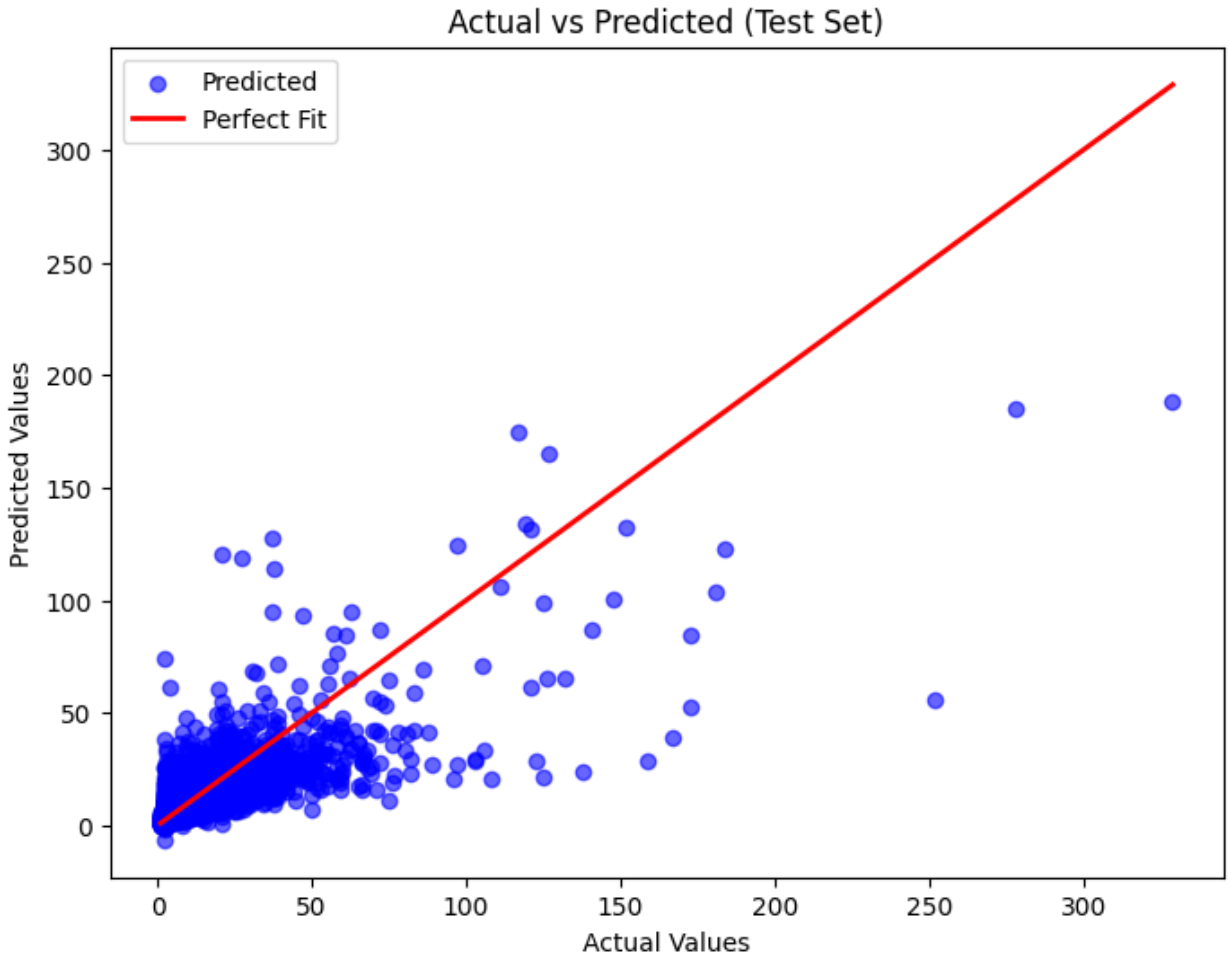


Figure 27: scatter plot for test set to predict the defect length using the XGBoost

The scatter plot comparing actual versus predicted defect lengths using the XGBoost model shows that while the model captures the general trend, there is significant variation in the predictions, as shown in the Figure 27. Many predicted values cluster closely around lower actual values, with increasing dispersion as the defect lengths grow larger. This pattern suggests that the model is less accurate at predicting higher defect lengths, as several points deviate substantially from the red line representing perfect predictions. The R^2 score of 56% indicates that the model explains 56% of the defect length variance, suggesting moderate predictive performance but with room for improvement, particularly in capturing larger defect lengths.

In the Repeated Defects Based on Defect Numbers (2007–2012 for Training, 2013 for Testing) scenario, all models experienced a more significant drop in test performance. Random Forest had an R-square of 68% on the training set, which dropped to 41% on the test set, indicating overfitting, with corresponding RMSEs of 10.82 (train) and 18.02 (test), showing a significant increase in prediction error on the test data. XGBoost had a lower R-square of 55% on the training data but achieved a slightly better test result with an R-square of 44%. However, its RMSE increased from 12.84 (train) to 17.38 (test), indicating a similar struggle in generalization. Cat Boost performed similarly to Random Forest, with a training R-square of 66% and a test R-square of 42%, and RMSEs of 11.13 (train) and 17.84 (test), indicating all models struggled to generalize well, with similar patterns of error growth in test performance.

Table 24: Regression Model Results for Defect Length Prediction

Data preparation	**Target**	**Method**	**R square in %** **Train / Train**	**RMSE** **Train / Test**
1. Random split 80% training and 20% test data	Defect length	Multiple linear	11 / 13	17.32 / 15.47
		Decision trees	50 / 47	12.13 / 13.02

		Random Forest	65 / 56	10.86 / 11.02
		XGBoost	72 / 56	9.74 / 11.08
		Cat Boost	65 / 56	10.80 / 10.97
2. Repeated defects based on defect number (2007 – 2012 on training and 2013 on test data)	Defect length	Random Forest	68 / 41	10.82 / 18.02
		XGBoost	55 / 44	12.84 / 17.38
		Cat Boost	66 / 42	11.13 / 17.84
3. Increase in time, which includes previous defect length and time gap) (80% training and 20% test data)	Defect length	Random Forest	70/ 45	15.22 / 21.89
		XGBoost	69 / 44	15.41 / 22.12
		Cat Boost	56 / 43	18.53 / 22.30

In the (Including Previous Defect Length and Time Gap) scenario, Random Forest showed improved training performance with an R-square of 70%. However, its test performance dropped to 45%, with corresponding RMSEs of 15.22 (train) and 21.89 (test), continuing the pattern of overfitting. XGBoost performed similarly, with a 69% R-square on the training set and 44% on the test set, with RMSEs of 15.41 (train) and 22.12 (test). CatBoost achieved a lower training R-square of 56% and test R-square of 43%, with RMSEs of 18.53 (train) and 22.30 (test), lagging the other models in Training but maintaining comparable test performance.

An evaluation of feature importance showed that defect type and freight speed influenced defect length predictions across all models. In Random Forest, defect type was the most significant feature, while in XGBoost and CatBoost, freight speed played a more important role, followed by defect type. This highlights the importance of operational factors like speed and defect characteristics in predicting defect behaviour over time. Defect amplitude was used as a more

dependable input than defect length predictions to enhance prediction accuracy. This is because amplitude is closely linked to defect type, a crucial factor in many models. Although defect amplitude cannot be directly used to predict defect type due to the unknown nature of amplitude until defects are identified, we can improve prediction accuracy by incorporating related variables such as speed, tonnage, milepost, and line segment number. This approach allows for more practical and accurate forecasting by focusing on known variables while maintaining the strong relationship between defect type and amplitude for better long-term predictions.

4.2.4 Defect tag prediction model using predicted amplitude

The Classification results for defect tag prediction using the predicted amplitude are presented in the Table 25. When using predicted amplitude to classify defect tags, Random Forest performed the best, achieving a 0.75 weighted F1 score with balanced individual F1 scores of 0.75 for both yellow and red defects, making it the most consistent model across both categories.

Table 25: Classification model results of defect tag using the predicted amplitude

Data preparation	Target	Method	F1-Score (Weighted average)	F1-Score (Yellow/Red)
3. Based on predicted amplitude	Defect tag	Random Forest	0.75	0.75/0.75
		XGBoost	0.71	0.69/0.72
		Cat boost	0.70	0.68/0.73

XGBoost, with a 0.71 weighted F1 score, showed a slight drop in performance for yellow tag (F1 score of 0.69) but performed better with red tag (F1 score of 0.72). CatBoost had the lowest weighted F1 score at 0.70, but it maintained a relatively balanced performance, with F1 scores of 0.68 for Yellow and 0.73 for Red, showing that while its overall performance was slightly lower, it remained stable across both defect categories.

Chapter 4. Discussions

This section provides detailed discussions of the Condition Assessment Framework and prediction model.

4.1 Condition Rating System

The Tie and rail fastening condition rating significantly improves previous studies by combining physical factors into a comprehensive condition rating system. Earlier models, such as the Track Quality Index (TQI) (R.-K. Liu et al. 2015) and Track Geometry Index (TGI) (Mundrey, J. S 2009) primarily focus on geometric parameters like gauge, alignment, cross-level, and surface condition but often overlook the physical health of the track's components. This model addresses that gap by integrating a component-focused assessment that considers the physical health and the location of tie crack, spikes, and tie plates. In addition to geometric indices like TQI and TGI, other models such as the Swedish National Railway Quality Index and the UK SD Index primarily assess the standard deviation of geometric parameters, like unevenness, alignment, and gauge, to monitor track quality. While these systems are effective at capturing track irregularities, they do not provide the same level of detail about the condition of track components, such as spikes and tie plates, which are crucial for ensuring long-term track stability. In this model, AHP provides a structured, transparent approach to assigning importance to various track features. This helps in decision-making regarding which factors require the most attention during maintenance. This framework offers a balanced and comprehensive view of track health by weighing crack-related factors alongside component conditions (like spikes and tie plates). It allows maintenance teams to focus on the areas that matter most, offering practical insights for maintenance planning and risk mitigation.

The Table 26 presents a detailed breakdown of the weights assigned to various factors by different respondents (Academia/Research, Design Consultants, Class I Railways, Short-line Operators, and Railway Service Providers). The factors evaluated include tie cracks, spikes, tie plates, crack location, crack size, spike distance, tie plate distance, and others, with each respondent group assigning different levels of importance to these factors. Different groups have varying opinions regarding the importance of crack size in assessing track conditions. Design consultants (39.77%)

and railway service providers (31.39%) assign much more weight to crack size compared to other groups like Class I railways (24.45%) and short-line operators (9.73%). Design consultants give higher weight to crack size because they focus on optimizing track design to prevent future maintenance issues. Cracks, particularly those related to size, signal underlying structural problems that could affect long-term track stability and integrity, making it a priority in design considerations. Railway service providers, on the other hand, prioritize crack size because they are directly responsible for track maintenance and repairs. Larger cracks require immediate attention, as they could lead to safety risks such as derailments or disruptions in service. The practical experience of railway service providers likely drives their focus on crack size, as they deal with the day-to-day consequences of track deterioration and must prioritize the factors that could lead to operational failures. These groups, being hands-on with track issues, view crack size as a critical indicator of impending failure, explaining their emphasis on it.

Academia and Class I railways rate spikes and tie plates similarly. For spikes, Academia assigns 36.29% and Class I assigns 35.09%. Similarly, Academia assigns 39.54% for tie plates, while Class I assigns 32.21%. However, short-line operators emphasize spikes (50.96%) more than tie plates (34.01%). Short-line operators likely focus more on spikes because spikes are critical in securing the rails to the ties, ensuring track stability, especially on smaller, less robust networks. Short lines typically operate with fewer resources and deal with older infrastructure. This makes spikes a priority for them, as loose or damaged spikes can lead to track misalignment and greater instability. Additionally, given the reduced traffic on short lines, the stresses placed on tracks may be less evenly distributed, making well-maintained spikes essential to ensuring that the track remains securely fastened. Any failure in the spike system could lead to more significant disruptions on short-line tracks, which are already operating with limited maintenance resources.

All respondent groups agree that crack location is crucial in assessing track conditions. Academia (70.89%), Class I railways (75.55%), short-line operators (90.27%), and railway service providers (68.61%) consistently prioritize the location of the crack, recognizing that cracks near critical components such as spikes and tie plates can compromise track stability and safety. This broad consensus highlights the critical role crack location plays in preventing further track degradation and ensuring the integrity of the track structure. There is also widespread agreement regarding the

importance of spike distance, with most groups assigning it significant weight. Academia (36.22%), Class I railways (28.33%), short-line operators (40.90%), and railway service providers (26.89%) agree that spike distance is a vital factor. However, design consultants assign relatively less weight to spike distance (17.81%), possibly because their primary focus is on the overall design of the track, aiming to optimize for long-term resilience rather than addressing the practical, day-to-day maintenance concerns like spikes. For them, crack location and size may take precedence over spike distance because design flaws in these areas could compromise the structural integrity of the track over time. Regarding crack depth, there is general agreement across the board that it is one of the most critical factors. Academia (39.31%), Class I railways (51.57%), short-line operators (45.09%), and design consultants (40.30%) all highlight crack depth as essential to determining track stability. Crack depth poses an immediate threat to the structural integrity of the track, as deeper cracks can weaken the ties more severely than shallow ones, which is why it garners such consistent focus among all groups.

Table 26: Summary of the weights assigned by the respondent's organization

Factors	**weights**	**Academia /Research**	**Design/ Consultant**	**Class I**	**Short line**	**Railway service provider**
Tie crack	45	24.17	81.55	32.70	15.03	77.19
Spike	28	36.29	9.47	35.09	50.96	5.04
Tie Plate	27	39.54	8.98	32.21	34.01	17.77
Total	100					
Location of the crack	74	70.89	60.23	75.55	90.27	68.61
Size of the crack	26	29.11	39.77	24.45	9.73	31.39
Total	100					
Spike distance	31	36.22	17.81	28.33	40.90	26.89

Tie plate distance	18	27.63	22.15	16.89	8.46	15.48
In line with spike	22	12.36	33.66	24.15	14.41	22.61
Direction	13	14.34	16.40	17.29	12.14	16.92
Side	16	9.46	9.98	13.35	24.09	18.10
Total	100					
Depth	42	39.31	40.30	51.57	45.09	33.50
Width	29	27.39	25.73	29.71	27.15	33.96
Length	19	24.04	23.28	9.29	17.10	22.43
Presence of Ballast	10	9.26	10.70	9.43	10.65	10.11
Total	100					

The thresholds for scoring factors like spike height, crack depth, and width cannot be based solely on data from a single location, such as a case study. Railway tracks experience different levels of wear and tear, with traffic density being a key factor. High-traffic tracks face faster degradation and require different maintenance thresholds than low-traffic tracks. In Scenario Two (without outliers), thresholds for crack measurements are based on the mean and standard deviation of the case study data, excluding outliers beyond three standard deviations. This ensures that the thresholds reflect typical conditions without being skewed by extreme values. Scenario Three includes outliers, setting thresholds based on maximum values observed in the dataset to account for significant defects. Only crack depth, width, and length, which have outliers beyond Pavemetrics' ranges, experience score changes between these scenarios. If thresholds are based solely on data from low-traffic tracks or skewed datasets, the severity of damage on high-traffic tracks may be misrepresented. A reliable rating system requires data from diverse locations to ensure thresholds apply across different tracks. Outliers can distort analysis, pulling averages away from central values. This is clear in Scenario Three, where outliers caused lower mean values for

crack measurements than in Scenario Two. Outliers—extreme values—skew the data, making track conditions appear less severe overall. For example, outliers may represent large cracks that develop quickly on high-traffic tracks, while on low-traffic tracks, outliers may be smaller or slower-growing cracks. Removing outliers in Scenario Two gives a more accurate picture of the typical condition, ensuring thresholds reflect realistic conditions. Thresholds should also reflect how defects impact track performance, not just their frequency. A crack width may be standard, but the scoring threshold should consider its effect on structural integrity. On high-traffic tracks, even minor defects, like shallow cracks, may require immediate attention due to constant heavy loads, while the same defect on low-traffic tracks may not pose a significant risk. For instance, a 5 mm crack depth on a high-traffic track might need immediate repair, but intervention might be delayed until the crack reaches 10 mm on a low-traffic track. Similarly, ballast inside cracks may be more critical on high-traffic tracks, where drainage is crucial. Setting thresholds based on severity, not just frequency, ensures maintenance teams address areas that pose the greatest risk to track stability and safety.

Fastening components like spikes and tie plates are essential for maintaining track stability. They distribute the load from passing trains, reduce stress on cracks, and prevent further deterioration. Spikes limit rail movement, while tie plates spread pressure across the tie, delaying crack progression. In the proposed Tie and Rail Fastening System, spikes and tie plates are weighted at 28% and 27%, reflecting their crucial role in track integrity. However, further research is needed to assess their impact across different track conditions. Focusing solely on tie crack size and location can lead to an underestimated track condition. Cracks do not provide a complete picture of a tie's performance. A tie with cracks may still be structurally sound if its fastening components are in good condition, as they help prevent further damage. For instance, if tie cracks are given 100% weight, the results show most of the track in moderate to severe condition. However, this overlooks the possibility that the fastening components may still be maintaining track stability. For instance, if the Tie crack is given 100% weight, where the rating is based entirely on tie cracks (both location and size) and does not account for the condition of fastening components, the results show that most of the track is rated as being in moderate to severe condition. Specifically, only 4.23% of the track is rated with light defects, while 41.07% is rated moderate, 51.87% as severe, and 2.82% as very severe. This suggests that the track is in relatively poor condition based on

crack data alone. However, this overlooks the possibility that fastening components may still be performing well, significantly improving the track's overall stability and functionality. From a physical standpoint, while cracks in the ties are a concern, fastening components are key to maintaining the integrity of the track system. Without healthy fastening components, cracks could propagate more quickly, leading to rapid degradation of the ties and, ultimately, the track. However, when spikes and tie plates are in good condition, they mitigate the impact of the cracks, distributing the forces more effectively and reducing the rate at which the cracks worsen. For example, a tie with substantial cracks but well-functioning spikes and tie plates can continue to carry loads safely and maintain rail alignment. On the other hand, a tie with minimal cracks but deteriorated fastening components may pose a greater risk to the track's stability because the spikes or tie plates may fail to hold the rails in place properly, leading to potential derailment risks.

One of the limitations of this system is that it is mainly based on expert opinions gathered through surveys rather than physical studies or data-driven experimentation. While the Analytical Hierarchy Process (AHP) provides a structured way to prioritize factors, it does not account for interdependencies between those factors. For example, the interaction between crack size, location, and fastening conditions may be more complex than the weights suggest. However, these relationships still need to be captured in the AHP framework. Additionally, the weights assigned to different factors are derived from a small group of experts, meaning the results may not represent broader industry views or be applicable in different operational contexts. Another limitation lies in the thresholds used for scoring, which are based on the specific types of ties and spikes in the case study. These thresholds would need to be adjusted for different materials or track conditions. The validation of this model was conducted on a small case study, which does not account for factors like the dynamic growth of defects over time or changes in environmental conditions. Another important limitation of this system is the inherent assumption of independence between the factors considered in the AHP framework. AHP operates under the assumption that each factor, such as crack size, location, and fastening condition, is evaluated independently, without accounting for potential interdependencies between them. In reality, these factors can have a complex interplay; for instance, the condition of tie fastenings may influence the rate at which cracks propagate, and the location of the crack in relation to the fastening might affect the severity of the defect. The absence of a mechanism to model these interactions in the AHP framework may

lead to an oversimplified understanding of the real-world behavior of track defects, limiting the accuracy and predictive capability of the system. Furthermore, the model does not consider geometry defects or characteristics like gauge and alignment, which can influence the development of cracks and other defects. By incorporating these factors, the system might notice critical aspects of track degradation, potentially leading to more accurate predictions of defect evolution.

4.2 Condition Prediction Model

The classification and regression results reveal that the applied machine learning models—Random Forest, XGBoost, and CatBoost—perform well in predicting railway defects, each excelling in specific areas. Among these, CatBoost emerges as a consistent performer in classifying defect tags, which is important for automating the identification process. Defect tags were predicted using the amplitude from the regression model, in addition to direct classification. This approach achieved a 75% weighted F1 score. It shows the potential of using predicted features from regression tasks to improve the accuracy of defect classification, which can contribute to more efficient maintenance processes. Predicting defect tags accurately can significantly improve maintenance planning by enabling early detection of critical issues affecting track safety and performance. Automation in defect detection allows railway operators to proactively address track defects, reducing manual inspections and improving operational efficiency. Random Forest also demonstrates strength, particularly in predicting defect types, which is essential for categorizing the severity of various track conditions. Accurate prediction of defect types, such as dips or surface irregularities, provides valuable insight into the underlying causes of track degradation. This can help maintenance teams prioritize their actions, focusing on defects that pose higher risks and ensuring that the track remains in optimal condition for the safe passage of trains.

Table 27: Summary of the models

Target	Model	Metrics
Defect Tag	Cat Boost	0.95 F1 score
Defect type	Random Forest	0.63 F1 score

Defect amplitude	Random Forest	82 R^2
Defect length	Cat Boost	56 R^2

In regression tasks, predicting defect characteristics such as amplitude and length using operational data, including tonnage, traffic density, and the total number of trains and cars travelling through the track, enhances the ability to plan maintenance more effectively. Railway operators can schedule preventative maintenance before defects worsen by understanding how traffic load and usage impact defect growth, minimizing costly repairs and avoiding disruptions. For instance, predicting defect length can help identify areas where track wear is progressing faster, enabling targeted interventions that extend the life of the track infrastructure. Leveraging models to predict defect tags, types, amplitudes, and lengths based on traffic and operational data introduces a more data-driven maintenance strategy.

The proposed machine learning models show considerable performance improvements compared to prior studies. Notably, CatBoost excels in predicting defect tags with a 95% F1 score, significantly outperforming (Alemazkoor, Ruppert, and Meidani 2018), who achieved a 70% accuracy, and (Cárdenas-Gallo et al. 2017), with an 81% accuracy. This improvement highlights CatBoost's effectiveness in identifying defect tags, which ensures timely maintenance and prevents track deterioration. However, in predicting defect types, Random Forest's 63% F1 score falls below the 72% accuracy reported by (Sudhir et al. 2015), but they used defect amplitude to predict defect type, which could result in data leakage, as these features are inherently correlated. The proposed models avoid this issue by independently predicting amplitude, length, and tags, allowing for better control and reducing the risk of overfitting. These models' ability to predict various defect characteristics—such as amplitude, length, and tags—provides a comprehensive evaluation of track conditions. This holistic approach can significantly enhance maintenance planning, budgeting, and risk mitigation. By identifying severe defects, the models also improve the prioritization of repairs, reducing track downtime and improving the railway system's overall performance and safety.

Table 28: Models from the literature

Reference	Target	Metrics
(Alemazkoor, Ruppert, and Meidani 2018)	Defect Tag	70% accuracy
(Cárdenas-Gallo et al. 2017)	Defect Tag	81% accuracy
(Sudhir Kumar Sinha, Sumit Raut, and Harshad Khadilkar 2015)	Defect Type	72% accuracy

Amplitude, class, and speed have emerged as critical features for forecasting defect tags, highlighting the importance of physical defect characteristics and operational factors in the models' performance. Feature importance analysis consistently emphasizes the significance of variables like "Defect Amplitude" and "Freight Speed," reinforcing that these elements play a crucial role in the formation and severity of railway defects. These insights suggest that certain physical and operational factors are the primary drivers behind defect development and incorporating them into predictive models is essential for achieving high accuracy. However, despite the success in identifying major defect patterns, the models encounter difficulties in predicting less severe defects, such as dips and surface irregularities. This challenge indicates that some currently represented features may only partially capture these subtler defect patterns, suggesting the need to refine the feature set. By incorporating additional features or improving the representation of existing ones, the models may become more sensitive to these less obvious defects, enhancing overall predictive performance.

The prediction models for Defect Amplitude and Defect Length demonstrate good performance when evaluated through their mean, standard deviation, and RMSE values. For Defect Amplitude, the mean is -0.06, with a standard deviation of 1.23. The model achieves a low RMSE of 0.52, indicating that it effectively captures amplitude variations with minimal error. The lower variability in amplitude allows the model to generalize well and achieve strong predictive performance. For Defect Length, the mean is 12.13 with a significantly larger standard deviation of 18.03, reflecting more significant variability. The RMSE for defect length prediction is 10.97, which, although higher than for amplitude, remains reasonable given the increased complexity in

defect length prediction. The model still provides satisfactory accuracy for practical use in maintenance planning. One key limitation of the models is overfitting, particularly in defect length prediction. Despite applying cross-validation and regularization techniques, the models still overfit the training data, reducing their generalization ability to new datasets. Overfitting is more prominent for defect length due to the higher variability in the data. Although cross-validation and regularization were applied to reduce overfitting, they were unsuccessful, particularly for defect length prediction. This suggests that further refinement, such as incorporating additional or more diverse features, is needed to enhance the model's generalization ability and reduce overfitting. While the models perform well for both defect amplitude and length, predicting amplitude is easier due to lower variability. Improving generalization for defect length prediction remains challenging, particularly in addressing overfitting. Despite the promising results, several limitations must be considered when applying these machine learning models for defect prediction and classification. Overfitting remains a key challenge, particularly in models like Random Forest and Decision Trees, which tend to perform well on training data but show a significant drop in accuracy when tested on unseen or temporal data, limiting their ability to generalize effectively over time. Another critical issue is the imbalance in the dataset, especially with underrepresented defect types like "Dip" and "Surface." To address this, SMOTE (Synthetic et al.) has been applied to oversample the Dip defect type to balance the dataset for defect type prediction. Additionally, class weights have been adjusted to give more importance to these underrepresented classes. SMOTE was also used for defect tag prediction when using predicted amplitude to balance the dataset and improve model accuracy. Furthermore, Stratified K-fold cross-validation was implemented to ensure the folds maintain the class distribution across the validation process. However, even with these adjustments, the models need higher accuracy for the less frequent defects. The imbalance, coupled with overfitting, reduces the models' ability to consistently predict less common defect types and defect tags with high accuracy. It is also important to note that defect amplitude was not used for defect type prediction. Instead, features such as Class, Speed, Tonnage, and Traffic Density were the key predictors. While using amplitude could potentially increase accuracy, it also introduces the risk of data leakage, as defect amplitude is inherently related to defect type. This could cause the model to inadvertently learn relationships that would not be available in a real-world prediction scenario, which could artificially inflate its performance. While

SMOTE, class weight adjustments, and stratified folding have been applied to mitigate data imbalance, the models still face accuracy limitations, especially when predicting underrepresented defect types. Further refinement in these approaches, more advanced feature engineering, and improved generalization techniques will be essential to enhance model performance and reliability in real-world applications. The feature set may also lack key variables, particularly those capturing temporal and other physical and environmental properties that influence defect formation. Incorporating variables like seasonal changes, track material properties, and other relevant physical factors could significantly improve model performance. Finally, the quality of the dataset is crucial in shaping the models' predictions. The current dataset may not fully capture the complexities of defect progression but adding high-resolution and contextual data could substantially improve the models' accuracy and robustness. As machine learning advances and more data is gathered through automatic track inspections, the model's accuracy could improve. Additional factors such as rail wear, ballast type, Tie type, and weather conditions (e.g., temperature) should be considered to enhance the model's reliability in the future.

Chapter 5. Conclusion

This study developed a two-prong approach to managing the condition of rail assets, including condition assessment and condition prediction.

The Tie and Rail Fastening Rating system offers a straightforward method to assess track conditions by focusing on critical defect characteristics like crack size, location, spike height, and tie plate presence. With a weighted scoring system, the method helps to identify and prioritize the most severe issues that affect track safety. Cracks, especially their size and location, play the most significant role in determining the track's condition, showing the need to understand how close they are to essential track components. The transition from the tie crack size rating scale to the proposed Tie and rail fastening system rating has made track condition assessments more accurate and detailed. The proposed system highlights key factors like crack depth and spike location to the crack, ensuring any problem areas are identified and addressed effectively. Overall, the proposed rating system improves track safety by providing a more holistic view of the track's condition by including both crack details and the condition of fastening components like spikes and tie plates. This comprehensive approach ensures that cracks are evaluated, and their proximity to essential track components is also considered, giving a clearer picture of overall track health. By offering this detailed perspective, the system helps maintenance teams prioritize maintenance decisions, focusing on areas with the highest risk and ensuring more effective use of resources.

The condition prediction model in this study employed machine learning techniques like Random Forest, XGBoost and Cat Boost to predict various rail defect characteristics, including defect tags (yellow or red), defect type, length, and amplitude. The accuracy of these models is directly influenced by the data available and the features selected. The model performed exceptionally well in predicting defect tags, achieving an accuracy of over 94% when using all available data. This high accuracy makes it a reliable tool for automatically identifying defect tags. Additionally, it achieved a 75% accuracy when using predicted amplitude to forecast defect tags. While slightly less accurate, this provides valuable insights for planning maintenance and repair efforts. Key features, such as amplitude, class, and speed, proved essential for accurate defect tag predictions. Unlike previous models, which often struggled to predict multiple aspects of defects—such as

defect type or length—with high accuracy, this model successfully integrates operational data (such as class, speed, and tonnage) and physical characteristics (like defect amplitude). This combination results in a more comprehensive understanding of defect behaviour over time, enhancing decision-making and resource allocation for maintenance teams. By relying on predicted future conditions, teams can act proactively rather than simply reacting to existing defects. This predictive approach represents a significant improvement in maintenance planning, offering a more dynamic view of rail conditions and helping to prevent potential issues from becoming critical. In summary, this model enhances defect detection and provides predictive capabilities that forecast future track conditions, making it an invaluable tool for maintaining railway infrastructure both efficiently and proactively.

References

Abdusalomov, Akmalbek, Nodirbek Baratov, Alpamis Kutlimuratov, and Taeg Keun Whangbo. 2021. "An Improvement of the Fire Detection and Classification Method Using YOLOv3 for Surveillance Systems." Sensors 21 (19): 6519. https://doi.org/10.3390/s21196519.

Alakh. 2024. "One Hot Encoding vs. Label Encoding in Machine Learning." 2024. https://www.analyticsvidhya.com/blog/2020/03/one-hot-encoding-vs-label-encoding-using-scikit-learn/.

Alemazkoor, Negin, Conrad J Ruppert, and Hadi Meidani. 2018. "Survival Analysis at Multiple Scales for the Modeling of Track Geometry Deterioration." Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit 232 (3): 842–50. https://doi.org/10.1177/0954409717695650.

Amini, M., and R. Dziedzic. 2022. "Comparison of Machine Learning Classifiers for Predicting Water Main Failure." In Proceedings of the Canadian Society of Civil Engineering Annual Conference 2021, 250:501–12. Lecture Notes in Civil Engineering. Singapore: Springer Nature Singapore. https://doi.org/10.1007/978-981-19-1065-4_42.

Andersson, M. 2002. "'Strategic Planning of Track Maintenance. State of the Art.' TRITA-INFRA 02-035 (2002)." In . https://scholar.google.com/scholar_lookup?title=Strategic+planning+of+track+maintenance&author=M.+Anderson&publication_year=2002&pages=61-85.

Andrade, A. Ramos, and P. Fonseca Teixeira. 2011. "Uncertainty in Rail-Track Geometry Degradation: Lisbon-Oporto Line Case Study." Journal of Transportation Engineering 137 (3): 193–200.

Andrade, A.R., and P.F. Teixeira. 2015. "Statistical Modelling of Railway Track Geometry Degradation Using Hierarchical Bayesian Models." Reliability Engineering & System Safety 142 (October):169–83. https://doi.org/10.1016/j.ress.2015.05.009.

AREMA Manual for Railway Engineering. 2022a. "AREMA Manual for Railway Engineering, Chapter 4 - Rails, Volume 1."

———. 2022b. "AREMA Manual for Railway Engineering, Chapter 30 - Ties, Volume 1."

Audley, M, and Jd Andrews. 2013. "The Effects of Tamping on Railway Track Geometry Degradation." Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit 227 (4): 376–91. https://doi.org/10.1177/0954409713480439.

Bai, Lei, Rengkui Liu, Quanxin Sun, Futian Wang, and Peng Xu. 2015. "Markov-Based Model for the Prediction of Railway Track Irregularities." Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit 229 (2): 150–59. https://doi.org/10.1177/0954409713503460.

Bard, Jonathan F. 1992. "A COMPARISON OF THE ANALYTIC HIERARCHY PROCESS WITH MULTIATTRIBUTE UTILITY THEORY: A CASE STUDY." IIE Transactions 24 (5): 111–21. https://doi.org/10.1080/07408179208964251.

Bhatia, Angat Pal Singh, SangHyeok Han, and Osama Moselhi. 2022. "A Simulation-Based Statistical Method for Planning Modular Construction Manufacturing." Journal of Information Technology in Construction 27 (February):130–44. https://doi.org/10.36680/j.itcon.2022.007.

Bing, Alan J., and Arnold Gross. 1983. "Development of Railroad Track Degradation Models." Transportation Research Record, no. 939.

Black, Erin. 2022. "Why Freight Railroads Are so Successful in the U.S." CNBC. 2022. https://www.cnbc.com/2022/02/03/why-freight-railroads-are-so-successful-in-the-us.html.

Bogdan Sowinski. 2013. "Interrelation between Wavelengths of Track Geometry Irregularities and Rail Vehicle Dynamic Properties." https://repo.pw.edu.pl/info/article/WUT406392.

Breiman, Leo, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. 2017. Classification And Regression Trees. 1st ed. Routledge. https://doi.org/10.1201/9781315139470.

Caetano, Luis Filipe, and Paulo Fonseca Teixeira. 2015. "Optimisation Model to Schedule Railway Track Renewal Operations: A Life-Cycle Cost Approach." Structure and Infrastructure Engineering 11 (11): 1524–36. https://doi.org/10.1080/15732479.2014.982133.

Cannon, D. F., K.-O. Edel, S. L. Grassie, and K. Sawley. 2003. "Rail Defects: An Overview." Fatigue & Fracture of Engineering Materials & Structures 26 (10): 865–86. https://doi.org/10.1046/j.1460-2695.2003.00693.x.

Cárdenas-Gallo, Iván, Carlos A. Sarmiento, Gilberto A. Morales, Manuel A. Bolivar, and Raha Akhavan-Tabatabaei. 2017. "An Ensemble Classifier to Predict Track Geometry Degradation." Reliability Engineering & System Safety 161 (May):53–60. https://doi.org/10.1016/j.ress.2016.12.012.

Chai, T., and R. R. Draxler. 2014. "Root Mean Square Error (RMSE) or Mean Absolute Error (MAE)? – Arguments against Avoiding RMSE in the Literature." Geoscientific Model Development 7 (3): 1247–50. https://doi.org/10.5194/gmd-7-1247-2014.

Chandra, S. Satish, and M. M. Agarwal. 2013. Railway Engineering. Second edition. 1 online resource vols. Oxford Higher Education. New Delhi: Oxford University Press. http://app.knovel.com/hotlink/toc/id:kpREE00012/railway-engineering-2nd.

Chawla, N. V., K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. 2002. "SMOTE: Synthetic Minority Over-Sampling Technique." Journal of Artificial Intelligence Research 16 (June):321–57. https://doi.org/10.1613/jair.953.

Cooper, Lauren, Daria Kotys-Schwartz, and Derek Reamon. 2012. "Using Random Forests to Identify Factors of Student Motivation in a Project-Based Learning Course." In Volume 5: Education and Globalization; General Topics, 39–48. Houston, Texas, USA: American Society of Mechanical Engineers. https://doi.org/10.1115/IMECE2012-86088.

David Nettleton. 2014. Commercial Data Mining. Elsevier. https://doi.org/10.1016/C2013-0-00263-0.

Dell'Orco, Mauro, Michele Ottomanelli, Leonardo Caggiani, and Domenico Sassanelli. 2008. "New Decision Support System for Optimization of Rail Track Maintenance Planning Based on Adaptive Neurofuzzy Inference System." Transportation Research Record: Journal of the Transportation Research Board 2043 (1): 49–54. https://doi.org/10.3141/2043-06.

Dersch, Marcus, Tom Roadcap, J. Riley Edwards, Yu Qian, Jae-Yoon Kim, and Matheus Trizotto. 2019. "Investigation into the Effect of Lateral and Longitudinal Loads on Railroad Spike Stress Magnitude and Location Using Finite Element Analysis." Engineering Failure Analysis 104 (October):388–98. https://doi.org/10.1016/j.engfailanal.2019.06.009.

Dersch, Marcus S., Christian Khachaturian, and J. Riley Edwards. 2021. "Methods to Mitigate Railway Premium Fastening System Spike Fatigue Failures Using Finite Element Analysis." Engineering Failure Analysis 121 (March):105160. https://doi.org/10.1016/j.engfailanal.2020.105160.

Dorogush, Anna Veronika, Vasily Ershov, and Andrey Gulin. 2018. "CatBoost: Gradient Boosting with Categorical Features Support." arXiv. https://doi.org/10.48550/ARXIV.1810.11363.

Draper, Norman R., and Harry Smith. 1998. Applied Regression Analysis. 1st ed. Wiley Series in Probability and Statistics. Wiley. https://doi.org/10.1002/9781118625590.

Elkhoury, Najwa, Lalith Hitihamillage, Sara Moridpour, and Dilan Robert. 2018. "Degradation Prediction of Rail Tracks: A Review of the Existing Literature." The Open Transportation Journal 12 (1): 88–104. https://doi.org/10.2174/1874447801812010088.

El-Sibaie, Magdy, and Yu-Jiang Zhang. 2004. "Objective Track Quality Indices." Transportation Research Record: Journal of the Transportation Research Board 1863 (1): 81–87. https://doi.org/10.3141/1863-11.

Enblom, Roger. 2009. "Deterioration Mechanisms in the Wheel–Rail Interface with Focus on Wear Prediction: A Literature Review." Vehicle System Dynamics 47 (6): 661–700. https://doi.org/10.1080/00423110802331559.

Esveld, Coenraad. 2001a. Modern Railway Track. 2. ed. Zaltbommel: MRT-Productions.

———. 2001b. Modern Railway Track. Vol. 385. MRT-productions Zaltbommel.

Falamarzi, Amir, Sara Moridpour, and Majidreza Nazem. 2019. "A Review of Rail Track Degradation Prediction Models." Australian Journal of Civil Engineering 17 (2): 152–66. https://doi.org/10.1080/14488353.2019.1667710.

Falamarzi, Amir, Sara Moridpour, Majidreza Nazem, and Samira Cheraghi. 2018. "Development of Random Forests Regression Model to Predict Track Degradation Index: Melbourne Case Study." In Australian Transport Research Forum, 12.

Gao, Yin, Mike McHenry, and Brad Kerchof. 2018a. "Investigation of Broken Cut Spikes on Elastic Fastener Tie Plates Using an Integrated Simulation Method." In 2018 Joint Rail Conference, V001T01A015. Pittsburgh, Pennsylvania, USA: American Society of Mechanical Engineers. https://doi.org/10.1115/JRC2018-6185.

———. 2018b. "Investigation of Broken Cut Spikes on Elastic Fastener Tie Plates Using an Integrated Simulation Method." In 2018 Joint Rail Conference, V001T01A015. Pittsburgh,

Pennsylvania, USA: American Society of Mechanical Engineers. https://doi.org/10.1115/JRC2018-6185.

Georgetown Rail. 2022. "The Georgetown Rail 'Aurora' Tie Inspection System." https://loram.com/inspection-and-optimization/inspection-services/tie-inspection-services/aurora/.

Gofran J. Qasim. 2019. "Ideal Track." https://uomustansiriyah.edu.iq/media/lectures/5/5_2019_11_12!12_50_18_PM.pdf.

Guler, Hakan. 2013. "Decision Support System for Railway Track Maintenance and Renewal Management." Journal of Computing in Civil Engineering 27 (3): 292–306. https://doi.org/10.1061/(ASCE)CP.1943-5487.0000221.

———. 2014a. "Prediction of Railway Track Geometry Deterioration Using Artificial Neural Networks: A Case Study for Turkish State Railways." Structure and Infrastructure Engineering 10 (5): 614–26.

———. 2014b. "Prediction of Railway Track Geometry Deterioration Using Artificial Neural Networks: A Case Study for Turkish State Railways." Structure and Infrastructure Engineering 10 (5): 614–26. https://doi.org/10.1080/15732479.2012.757791.

Guler, Hakan, Stanislav Jovanovic, and Gungor Evren. 2011. "Modelling Railway Track Geometry Deterioration." Proceedings of the Institution of Civil Engineers - Transport 164 (2): 65–75. https://doi.org/10.1680/tran.2011.164.2.65.

Gustavsson, Emil. 2015. "Scheduling Tamping Operations on Railway Tracks Using Mixed Integer Linear Programming." EURO Journal on Transportation and Logistics 4 (1): 97–112.

Han, Lei, Yingying Liao, Haoyu Wang, and Hougui Zhang. 2024. "Analysis and Prediction of Railway Track Longitudinal Level Using Multiple Machine Learning Methods." Measurement Science and Technology 35 (2): 024001. https://doi.org/10.1088/1361-6501/ad060a.

Hay, William Walter. 1982. Railroad Engineering. 2nd ed. New York: Wiley.

He, Qing, Hongfei Li, Debarun Bhattacharjya, Dhaivat P Parikh, and Arun Hampapur. 2015a. "Track Geometry Defect Rectification Based on Track Deterioration Modelling and Derailment Risk Assessment." Journal of the Operational Research Society 66 (3): 392–404. https://doi.org/10.1057/jors.2014.7.

———. 2015b. "Track Geometry Defect Rectification Based on Track Deterioration Modelling and Derailment Risk Assessment." Journal of the Operational Research Society 66 (3): 392–404. https://doi.org/10.1057/jors.2014.7.

Herrero, Álvaro, Secil Bayraktar, and Alfredo Jiménez. 2020. "Machine Learning to Forecast the Success of Infrastructure Projects Worldwide." Cybernetics and Systems 51 (7): 714–31. https://doi.org/10.1080/01969722.2020.1798645.

Hosmer, David W., Stanley Lemeshow, and Rodney X. Sturdivant. 2013. Applied Logistic Regression. 1st ed. Wiley Series in Probability and Statistics. Wiley. https://doi.org/10.1002/9781118548387.

Hu, Can, and Xiang Liu. 2016. "Modeling Track Geometry Degradation Using Support Vector Machine Technique." In ASME/IEEE Joint Rail Conference, 49675:V001T01A011. American Society of Mechanical Engineers.

Hu, Qihang, Rui Gao, Jing Chen, and Zhiwen Yuan. 2023. "Ballast Deterioration Inspection and Quantification with 3D Form Method Based on Particle Inscribed Ellipsoid." Granular Matter 25 (3): 54. https://doi.org/10.1007/s10035-023-01348-5.

Indraratna, Buddhima, Li-jun Su, and Cholachat Rujikiatkamjorn. 2011a. "A New Parameter for Classification and Evaluation of Railway Ballast Fouling." Canadian Geotechnical Journal 48 (2): 322–26. https://doi.org/10.1139/T10-066.

———. 2011b. "A New Parameter for Classification and Evaluation of Railway Ballast Fouling." Canadian Geotechnical Journal 48 (2): 322–26. https://doi.org/10.1139/T10-066.

Ionescu, Daniela. 2023. "Ballast Degradation and Measurement of Ballast Fouling."

Jovanovic, Stanislav, Hakan Guler, and Bosko Coko. 2015. "Track Degradation Analysis in the Scope of Railway Infrastructure Maintenance Management Systems." Gradevinar 67 (3): 247–57.

Kabir, Golam, Rehan Sadiq, and Solomon Tesfamariam. 2014. "A Review of Multi-Criteria Decision-Making Methods for Infrastructure Management." Structure and Infrastructure Engineering 10 (9): 1176–1210. https://doi.org/10.1080/15732479.2013.795978.

Karimpour, Mostafa, Lalith Hitihamillage, Najwa Elkhoury, Sara Moridpour, and Reyhaneh Hesami. 2018. "Fuzzy Approach in Rail Track Degradation Prediction." Journal of Advanced Transportation 2018:1–7. https://doi.org/10.1155/2018/3096190.

King, Gary, and Langche Zeng. 2001. "Logistic Regression in Rare Events Data." Political Analysis 9 (2): 137–63. https://doi.org/10.1093/oxfordjournals.pan.a004868.

Kleinbaum, David G., and Mitchel Klein. 2010. Logistic Regression. Statistics for Biology and Health. New York, NY: Springer New York. https://doi.org/10.1007/978-1-4419-1742-3.

Kutner, M. H, C. J Nachtsheim, and J Neter. 2004. Applied Linear Regression Models. chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://users.stat.ufl.edu/~winner/sta4211/ALSM_5Ed_Kutner.pdf.

Lasisi, Ahmed, and Nii Attoh-Okine. 2018. "Principal Components Analysis and Track Quality Index: A Machine Learning Approach." Transportation Research Part C: Emerging Technologies 91:230–48.

Li, Dingqing. 2018. "25 Years of Heavy Axle Load Railway Subgrade Research at the Facility for Accelerated Service Testing (FAST)." Transportation Geotechnics 17:51–60.

Li, Haifeng, and Yude Xu. 2009. "Railway Track Integral Maintenance Index and Its Application." In International Conference on Transportation Engineering 2009, 2514–19. Southwest Jiaotong University, Chengdu, China: American Society of Civil Engineers. https://doi.org/10.1061/41039(345)415.

Li, Hongfei, Dhaivat Parikh, Qing He, Buyue Qian, Zhiguo Li, Dongping Fang, and Arun Hampapur. 2014. "Improving Rail Network Velocity: A Machine Learning Approach to

Predictive Maintenance." Transportation Research Part C: Emerging Technologies 45 (August):17–26. https://doi.org/10.1016/j.trc.2014.04.013.

Li, Hui, and Tianyuan Xiao. 2014. "Improved Generalized Energy Index Method for Comprehensive Evaluation and Prediction of Track Irregularity." Journal of Statistical Computation and Simulation 84 (6): 1213–31. https://doi.org/10.1080/00949655.2013.797420.

Li, Qing, Qiyuan Peng, Rengkui Liu, Ling Liu, and Lei Bai. 2019. "Track Grid Health Index for Grid-Based, Data-Driven Railway Track Health Evaluation." Advances in Mechanical Engineering 11 (11): 168781401988976. https://doi.org/10.1177/1687814019889768.

Liao, Yingying, Lei Han, Haoyu Wang, and Hougui Zhang. 2022. "Prediction Models for Railway Track Geometry Degradation Using Machine Learning Methods: A Review." Sensors 22 (19): 7275. https://doi.org/10.3390/s22197275.

Liu, Reng-Kui, Peng Xu, Zhuang-Zhi Sun, Ce Zou, and Quan-Xin Sun. 2015. "Establishment of Track Quality Index Standard Recommendations for Beijing Metro." Discrete Dynamics in Nature and Society 2015:1–9. https://doi.org/10.1155/2015/473830.

Liu, Rengkui, Peng Xu, and Futian Wang. 2010. "Research on a Short-Range Prediction Model for Track Irregularity over Small Track Lengths." Journal of Transportation Engineering 136 (12): 1085–91. https://doi.org/10.1061/(ASCE)TE.1943-5436.0000192.

Long, Tsang, Biao He, Ali Ghorbani, and Seyed Khatami. 2023. "Tree-Based Techniques for Predicting the Compression Index of Clayey Soils." Journal of Soft Computing in Civil Engineering 7 (3). https://doi.org/10.22115/scce.2023.377601.1579.

Ma, L., L.B. Shi, J. Guo, Q.Y. Liu, and W.J. Wang. 2018. "On the Wear and Damage Characteristics of Rail Material under Low Temperature Environment Condition." Wear 394–395 (January):149–58. https://doi.org/10.1016/j.wear.2017.10.011.

Madejski, Janusz. 2015. "Continuous Geometry Measurement for Diagnostics of Tracks and Switches." chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://www.researchgate.net/profile/Janusz-Madejski-2/publication/266460151_Continuous_geometry_measurement_for_diagnostics_of_tracks_and_switches/links/55b255c708aec0e5f4317b5a/Continuous-geometry-measurement-for-diagnostics-of-tracks-and-switches.pdf.

Marlow, David R., and Stewart Burn. 2008. "Effective Use of Condition Assessment within Asset Management." Journal AWWA 100 (1): 54–63. https://doi.org/10.1002/j.1551-8833.2008.tb08129.x.

Marquis, Brian P., Michelle Muhlanger, and David Y. Jeong. 2011. "Effect of Wheel/Rail Loads on Concrete Tie Stresses and Rail Rollover." In ASME 2011 Rail Transportation Division Fall Technical Conference, 143–50. Minneapolis, Minnesota, USA: ASMEDC. https://doi.org/10.1115/RTDF2011-67025.

McDowell, G. R., W. L. Lim, A. C. Collop, R. Armitage, and N. H. Thom. 2004. "Comparison of Ballast Index Tests for Railway Trackbeds." Proceedings of the Institution of Civil

Engineers - Geotechnical Engineering 157 (3): 151–61. https://doi.org/10.1680/geng.2004.157.3.151.

Md Saeed Hasan. 2015. "Deterioration Prediction of Concrete Bridge Components Using Artificial Intelligence and Stochastic Methods."

Montgomery, D. C, E. A Peck, and G. G. Vining. 2012. Introduction to Linear Regression Analysis. chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://statanaly.com/wp-content/uploads/2023/05/IntroductiontoLinearRegressionAnalysisbyDouglasC.MontgomeryElizabethA.PeckG_.GeoffreyViningz-lib.org_.pdf.

Moridpour, Sara, Ehsan Mazloumi, and Reyhaneh Hesami. 2017. "Application of Artificial Neural Networks in Predicting the Degradation of Tram Tracks Using Maintenance Data." In Applied Big Data Analytics in Operations Management, 30–54. IGI Global. https://web-s-ebscohost-com.lib-ezproxy.concordia.ca/ehost/ebookviewer/ebook/bmxlYmtfXzEzNjU5MzZfX0FO0?sid=af436e42-ba1e-4574-9616-e9639e330904@redis&vid=0&format=EB&lpid=lp_30&rid=0.

Mu, Enrique, and Milagros Pereyra-Rojas. 2017. "Understanding the Analytic Hierarchy Process." In Practical Decision Making, by Enrique Mu and Milagros Pereyra-Rojas, 7–22. SpringerBriefs in Operations Research. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-33861-3_2.

Mundrey, J. S. 2009. Railway Track Engineering. Tata McGraw-Hill Education, 2009. https://scholar.google.com/scholar_lookup?title=Railway+Track+Engineering&author=Mundrey,+J.&publication_year=2003#d=gs_cit&t=1717363429237&u=%2Fscholar%3Fq%3Dinfo%3AbQ5V8sDJSQ.

Offenbacher, Stefan, Johannes Neuhold, Peter Veit, and Matthias Landgraf. 2020. "Analyzing Major Track Quality Indices and Introducing a Universally Applicable TQI." Applied Sciences 10 (23): 8490. https://doi.org/10.3390/app10238490.

Palese, Joseph W., PE MCE, Donald R. Holfeld, and P. Eng. 1999. "Tie Planning Tools for the Track Inspector." Jan 1:4.

Peng, Fan, Yanfeng Ouyang, and Kamalesh Somani. 2013. "Optimal Routing and Scheduling of Periodic Inspections in Large-Scale Railroad Networks." Journal of Rail Transport Planning & Management 3 (4): 163–71. https://doi.org/10.1016/j.jrtpm.2014.02.003.

Powell, A, and P Gräbe. 2017. "Exploring the Relationship between Vertical and Lateral Forces, Speed and Superelevation in Railway Curves." Journal of the South African Institution of Civil Engineering 59 (3): 25–35. https://doi.org/10.17159/2309-8775/2017/v59n3a4.

Prokhorenkova, Liudmila, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. 2017. "CatBoost: Unbiased Boosting with Categorical Features." arXiv. https://doi.org/10.48550/ARXIV.1706.09516.

Puffert, Douglas J. 2000. "The Standardization of Track Gauge on North American Railways, 1830–1890." The Journal of Economic History 60 (4): 933–60.

Puzavac, Leposava, Zdenka Popović, and Luka Lazarević. 2012. "Influence of Track Stiffness on Track Behaviour under Vertical Load." Promet-Traffic&Transportation 24 (5): 405–12.

Quinlan, J. R. 1986. "Induction of Decision Trees." Machine Learning 1 (1): 81–106. https://doi.org/10.1007/BF00116251.

Quiroga, L., and E. Schnieder. 2010. "Modelling High Speed Railroad Geometry Ageing as a Discrete-Continuous Process." In Proceedings of the Stochastic Modeling Techniques and Data Analysis International Conference, SMTDA, Chania Crete Greece, 655–66. https://www.researchgate.net/profile/Teresa-Rivas-Moya-2/publication/317596376_Generalizability_Analysis_An_Example_Using_Unbalanced_Data/links/594276edaca272c2cac2a6b5/Generalizability-Analysis-An-Example-Using-Unbalanced-Data.pdf.

Rahimikelarijani, Behnam, Ahmad Mohassel, and Maryam Hamidi. 2020. "Railroad Track Geometric Degradation Analysis: A BNSF Case Study." Journal of Transportation Engineering, Part A: Systems 146 (2): 04019068. https://doi.org/10.1061/JTEPBS.0000303.

Railway Association of Canada. 2022. "RAC-Rail Trends-2022." chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://www.railcan.ca/wp-content/uploads/2022/12/RAC-Rail-Trends-2022-EN.pdf.

Railway Tie Association. 2024. "Railway Tie Association." https://www.rta.org/faq#:~:text=This%20translates%20to%20an%20average,each%20year%20in%20these%20countries.

"RAS 2015." n.d.

RAS 2015 problem. 2015. https://higherlogicdownload.s3.amazonaws.com/INFORMS/e52cec4c-eedb-4c3b-a379-8408d89f8fc9/UploadedImages/Data.20150707.zip).

Rokach, Lior, and Oded Maimon. 2005. "Decision Trees." In Data Mining and Knowledge Discovery Handbook, edited by Oded Maimon and Lior Rokach, 165–92. New York: Springer-Verlag. https://doi.org/10.1007/0-387-25465-X_9.

S Kaewunruen, AM Remennikov. 2005. "Integrated Field Measurements and Track Simulations for Condition Assessment of Railway Track." https://www.academia.edu/download/41216873/Integrated_field_measurements_and_track_20160113-22193-186910c.pdf20160115-19908-1einmu0.pdf.

Saaty, Thomas L., and Luis G. Vargas. 2012a. "How to Make a Decision." In Models, Methods, Concepts & Applications of the Analytic Hierarchy Process, by Thomas L. Saaty and Luis G. Vargas, 175:1–21. International Series in Operations Research & Management Science. Boston, MA: Springer US. https://doi.org/10.1007/978-1-4614-3597-6_1.

———. 2012b. "The Seven Pillars of the Analytic Hierarchy Process." In Models, Methods, Concepts & Applications of the Analytic Hierarchy Process, by Thomas L. Saaty and Luis G. Vargas, 175:23–40. International Series in Operations Research & Management Science. Boston, MA: Springer US. https://doi.org/10.1007/978-1-4614-3597-6_2.

Sadeghi, J. 2010a. "Development of Railway Track Geometry Indexes Based on Statistical Distribution of Geometry Data." Journal of Transportation Engineering 136 (8): 693–700. https://doi.org/10.1061/(ASCE)0733-947X(2010)136:8(693).

———. 2010b. "Development of Railway Track Geometry Indexes Based on Statistical Distribution of Geometry Data." Journal of Transportation Engineering 136 (8): 693–700. https://doi.org/10.1061/(ASCE)0733-947X(2010)136:8(693).

Sadeghi, J. M., and H. Askarinejad. 2011. "Development of Track Condition Assessment Model Based on Visual Inspection." Structure and Infrastructure Engineering 7 (12): 895–905. https://doi.org/10.1080/15732470903194676.

Sadeghi, Javad, and Hossein Askarinejad. 2010. "Development of Improved Railway Track Degradation Models." Structure and Infrastructure Engineering 6 (6): 675–88.

———. 2012. "Application of Neural Networks in Evaluation of Railway Track Quality Condition." Journal of Mechanical Science and Technology 26 (1): 113–22. https://doi.org/10.1007/s12206-011-1016-5.

Scanlan, Kirk M., Michael T. Hendry, and C. Derek Martin. 2016. "Evaluating the Equivalency Between Track Quality Indices and the Minimum Track Geometry Threshold Exceedances Along a Canadian Freight Railway." In 2016 Joint Rail Conference, V001T01A012. Columbia, South Carolina, USA: American Society of Mechanical Engineers. https://doi.org/10.1115/JRC2016-5748.

Setiawan, Dian M, and P. Rosyidi Sri Atmaja. 2016. "TRACK QUALITY INDEX AS TRACK QUALITY ASSESSMENT INDICATOR." chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://d1wqtxts1xzle7.cloudfront.net/57391669/Revisi_TrackQualityIndex_DSM_Fullpaper_SymposiumFSTPT19-libre.pdf?1537135875=&response-content-disposition=inline%3B+filename%3DTRACK_QUALITY_INDEX_AS_TRACK_QUALITY_ASS.pdf&Expires=1727226797&Signature=FnDu3~USsBHfvRnasYm7FfCxv-772Hlvl42B9iTHpDw0~MdPNeLNgLJ0-JQ1Y0-P9ELyQBZDi-JTX8zwjF-FtNhvQYntrB8-3pMLj~TtLLo046KkxnWitf1V48nmyQR1NAwrTMkk6VwCyndcHD~~ocZzW4P~3z-iqTwGAJS0A2z1hCbyfNxIau1ZGqyyWuf4pNyV3s7YBCRjUoMbWEyKyBf4zodSATSmju9z9tOK-boJF84XNBDMoMkVBrHNJYVy1tv2Z~sh7a7g0eE-8bnAMaOe450tg~bKufUhP5hc1MLw0DdBVxFyDagTOXdzwZr~NhSal0Llw4cLXcSk-roF5A__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA.

Shafahi, Y., and R. Hakhamaneshi. 2009. "Application of a Maintenance Management Model for Iranian Railways Based on the Markov Chain and Probabilistic Dynamic Programming." Scientia Iranica 16 (1). https://scientiairanica.sharif.edu/article_3182_130767cafcf651c287704fa787c6c3e0.pdf.

Shafahi, Y, P Masoudi, and R Hakhamaneshi. 2008. "Track Degradation Prediction Models, Using Markov Chain, Artificial Neural and Neuro-Fuzzy Network." Tehran, Iran: Sharif University of Technology.

Sharma, L. K., and T. N. Singh. 2018. "Regression-Based Models for the Prediction of Unconfined Compressive Strength of Artificially Structured Soil." Engineering with Computers 34 (1): 175–86. https://doi.org/10.1007/s00366-017-0528-8.

Sipahi, Seyhan, and Mehpare Timor. 2010. "The Analytic Hierarchy Process and Analytic Network Process: An Overview of Applications." Management Decision 48 (5): 775–808. https://doi.org/10.1108/00251741011043920.

Soares, C, ed. 2011. Advances in Safety, Reliability and Risk Management: ESREL 2011. CRC Press. https://doi.org/10.1201/b11433.

Soleimanmeigouni, I., A. Ahmadi, A. Nissen, and Xun Xiao. 2020. "Prediction of Railway Track Geometry Defects: A Case Study." Structure and Infrastructure Engineering 16:1001–1987. https://doi.org/10.1080/15732479.2019.1679193.

Solomon, Brian. 2001. Railway Maintenance Equipment. Osceola, Wis: MBI Pub. Co.

Sonti, Somnath S., Julio G. Davalos, Michael G. Zipfel, and Hota VS GangaRao. 1995. "A Review of Wood Crosstie Performance." Forest Products Journal 45 (9): 55.

Steven Loaiza. 2020. "Gini Impurity Measure – a Simple Explanation Using Python." 2020. https://towardsdatascience.com/gini-impurity-measure-dbd3878ead33#:~:text=Def%3A%20Gini%20Impurity%20tells%20us,lower%20the%20likelihood%20of%20misclassification.

Sudhir Kumar Sinha, Sumit Raut, and Harshad Khadilkar. 2015. "Track Geomerty Analytics." Https://Higherlogicdownload.S3.Amazonaws.Com/INFORMS/E52cec4c-Eedb-4c3b-A379-8408d89f8fc9/UploadedImages/Report_TCS_Explorers.Pdf. September 18, 2015. https://higherlogicdownload.s3.amazonaws.com/INFORMS/e52cec4c-eedb-4c3b-a379-8408d89f8fc9/UploadedImages/Report_TCS_Explorers.pdf.

The Transportation Safety Board of Canada. 2012. "RAILWAY INVESTIGATION REPORT R12E0008." https://www.bst-tsb.gc.ca/eng/rapports-reports/rail/2012/r12e0008/r12e0008.pdf.

Transport Canada. 2022. "RULES RESPECTING TRACK SAFETY." https://tc.canada.ca/en/rail-transportation/rules/2021-2022/rules-respecting-track-safety.

———. 2023. "Transport Canada Annual Report." https://tc.canada.ca/en/corporate-services/transparency/corporate-management-reporting/transportation-canada-annual-reports/2021/rail-network.

TSB. 2021. "Rail Transportation Safety Investigation R21V0118." https://www.tsb.gc.ca/eng/enquetes-investigations/rail/2021/R21V0118/R21V0118.html.

———. 2023. "Rail Transportation Occurrences in 2023." https://www.tsb.gc.ca/eng/stats/rail/2023/sser-ssro-2023.html.

Uzarski, D. R., Donald George Brown, Richard W. Harris, and Donald E. Plotkin. 1993. "Maintenance Management of US Army Railroad Networks–the RAILER System: Detailed Track Inspection Manual." https://apps.dtic.mil/sti/citations/tr/ADA274459.

Vaidya, Omkarprasad S., and Sushil Kumar. 2006. "Analytic Hierarchy Process: An Overview of Applications." European Journal of Operational Research 169 (1): 1–29. https://doi.org/10.1016/j.ejor.2004.04.028.

Vale, Cecília, and Simões M. Lurdes. 2013. "Stochastic Model for the Geometrical Rail Track Degradation Process in the Portuguese Railway Northern Line." Reliability Engineering & System Safety 116 (August):91–98. https://doi.org/10.1016/j.ress.2013.02.010.

Vidal, Ludovic-Alexandre, Franck Marle, and Jean-Claude Bocquet. 2011. "Using a Delphi Process and the Analytic Hierarchy Process (AHP) to Evaluate the Complexity of Projects." Expert Systems with Applications 38 (5): 5388–5405. https://doi.org/10.1016/j.eswa.2010.10.016.

Weston, P. F., C. S. Ling, C. J. Goodman, Clive Roberts, P. Li, and R. M. Goodall. 2007. "Monitoring Lateral Track Irregularity from In-Service Railway Vehicles." Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit 221 (1): 89–100.

Wong, Tzu-Tsung, and Po-Yang Yeh. 2020. "Reliable Accuracy Estimates from k -Fold Cross Validation." IEEE Transactions on Knowledge and Data Engineering 32 (8): 1586–94. https://doi.org/10.1109/TKDE.2019.2912815.

Xu, P, Q Sun, R Liu, and F Wang. 2011. "A Short-Range Prediction Model for Track Quality Index." Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit 225 (3): 277–85. https://doi.org/10.1177/2041301710392477.

Yan, Tzu-Hao, and Francesco Corman. 2020. "Assessing and Extending Track Quality Index for Novel Measurement Techniques in Railway Systems." Transportation Research Record: Journal of the Transportation Research Board 2674 (8): 24–36. https://doi.org/10.1177/0361198120923661.

Yu, Hailing, and David Jeong. 2012. "Railroad Tie Responses to Directly Applied Rail Seat Loading in Ballasted Tracks: A Computational Study." In 2012 Joint Rail Conference, 123–32. Philadelphia, Pennsylvania, USA: American Society of Mechanical Engineers. https://doi.org/10.1115/JRC2012-74149.

Zarembski, Allan M., Gregory T. Grissom, Todd L. Euston, and John J. Cronin. 2015. "Relationship between Missing Ballast and Development of Track Geometry Defects." Transportation Infrastructure Geotechnology 2:167–76.

Zerbst, U., R. Lundén, K.-O. Edel, and R.A. Smith. 2009. "Introduction to the Damage Tolerance Behaviour of Railway Rails – a Review." Engineering Fracture Mechanics 76 (17): 2563–2601. https://doi.org/10.1016/j.engfracmech.2009.09.003.

Zimmermann, H-J. 2010. "Fuzzy Set Theory." Wiley Interdisciplinary Reviews: Computational Statistics 2 (3): 317–32.

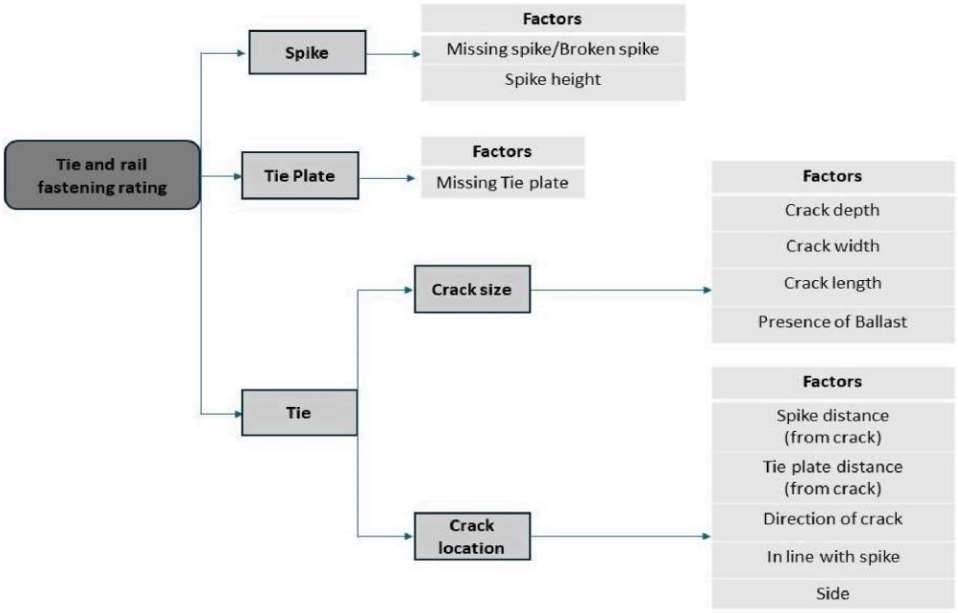
Appendix A. Questionnaire survey for Tie and rail fastening system

Tie and rail fastening rating survey

Maintaining railroad assets in good condition is crucial for the safe and reliable transportation of both cargo and passengers. The condition of ties and rail fastening components has traditionally been assessed separately, either by measuring individual defects or relying on subjective evaluations of sections. This research aims to develop a rating system that holistically evaluates the condition of ties, tie plates, and spikes across railroad sections. The following survey will take less than 15 minutes to answer and seeks to gather insights from individuals in the railroad industry regarding the severity of different types of defects. The resulting rating system will formalize these insights, supporting future automatic condition rating of railroad assets and aiding targeted maintenance planning and risk assessment.

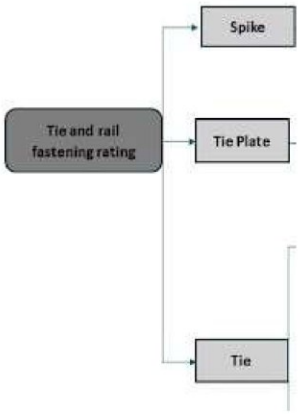
* Indicates required question

All the following questions will ask you to compare the severity of different types of defects and factors. The framework established for the rating system is shown below.



1. 1. How do you rate the severity of **spike defects** compared to **tie cracks**? *

   Spike defects are _____ than tie cracks.

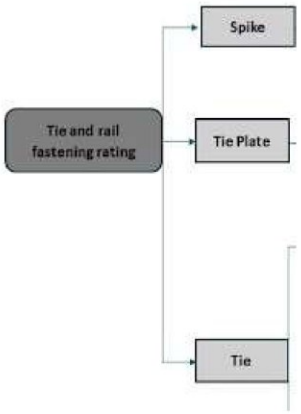


Check all that apply.

- [ ] a) significantly more severe
- [ ] b) moderately more severe
- [ ] c) equally severe
- [ ] d) moderately less severe
- [ ] e) significantly less severe

2. 2. How do you rate the severity of **tie plate defects** compared to **tie cracks**? *

Tie plate defects are _____ than tie cracks.



Check all that apply.

- [ ] a) significantly more severe
- [ ] b) moderately more severe
- [ ] c) equally severe
- [ ] d) moderately less severe
- [ ] e) significantly less severe

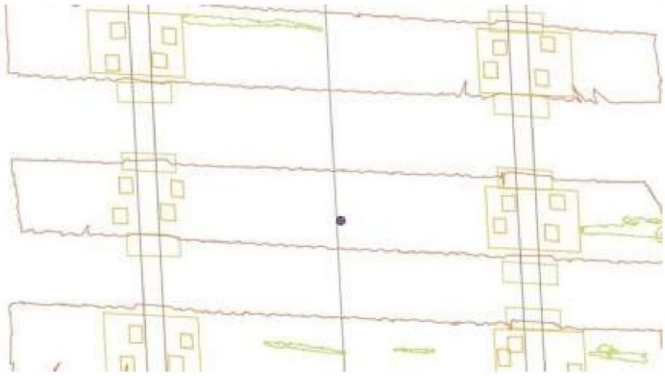
3. 3. When assessing the severity of **spike defects**, which of the following factors is more important? *

The occurrence of a **missing or broken spike** is _____ than the **height of the spike**.

Check all that apply.

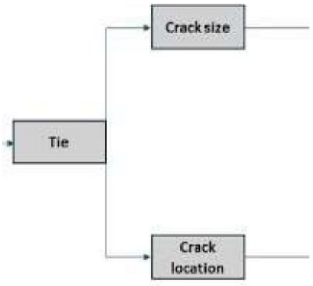
- [ ] a) significantly more important
- [ ] b) moderately more important
- [ ] c) equally important
- [ ] d) moderately less important
- [ ] e) significantly less important

Picture describing ties with crack nearest to spike, inline with spike, crack in field and gauge side and direction of the crack



4. 4. When assessing the severity of a tie crack, how important is its location compared to its size? *

The **location of the tie crack** is _____ than its **size**.

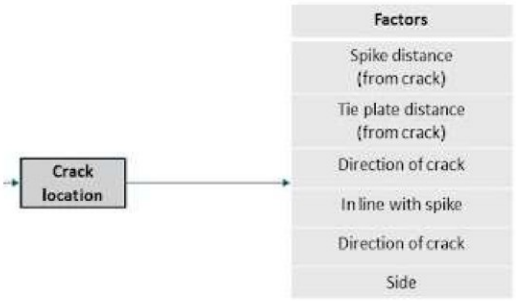


Check all that apply.

- [ ] a) significantly more important
- [ ] b) moderately more important
- [ ] c) equally important
- [ ] d) moderately less important
- [ ] e) significantly less important

5. 5. When assessing the severity of tie cracks in different locations, which of the following factors are more important? *

i)The distance from **the tie crack to the nearest spike** is _____ than the distance to **the tie plate**.



Check all that apply.

- [ ] a) significantly more important
- [ ] b) moderately more important
- [ ] c) equally important
- [ ] d) moderately less important
- [ ] e) significantly less important

6. ii) The distance from **the tie crack to the nearest spike** is _____ than **the direction of the crack** (along or across the tie) *

Check all that apply.

- [ ] a) significantly more important
- [ ] b) moderately more important
- [ ] c) equally important
- [ ] d) moderately less important
- [ ] e) significantly less important

7. iii) The distance from **the tie crack to the nearest spike** is _____ than whether **the crack is in line with the spike** *

Check all that apply.

- [ ] a) significantly more important
- [ ] b) moderately more important
- [ ] c) equally important
- [ ] d) moderately less important
- [ ] e) significantly less important

8. iv) The distance from **the tie crack to the nearest spike** is _____ than **the side of the crack** (field or gauge) *

Check all that apply.

- [ ] a) significantly more important
- [ ] b) moderately more important
- [ ] c) equally important
- [ ] d) moderately less important
- [ ] e) significantly less important

9. v) The distance **from the tie crack to the tie plate** is _____ than **the direction of the crack** (along or across the tie) *

Check all that apply.

- [ ] a) significantly more important
- [ ] b) moderately more important
- [ ] c) equally important
- [ ] d) moderately less important
- [ ] e) significantly less important

10. vi) The distance from **the tie crack to the tie plate** is _____ than whether **the crack is in line with the spike** *

Check all that apply.

- [ ] a) significantly more important
- [ ] b) moderately more important
- [ ] c) equally important
- [ ] d) moderately less important
- [ ] e) significantly less important

11. vii)The distance of **the crack from tie plate**_____ than the **side of the crack** (field or gauge) *

Check all that apply.

- [ ] a) significantly more important
- [ ] b) moderately more important
- [ ] c) equally important
- [ ] d) moderately less important
- [ ] e) significantly less important

12. viii)The **direction of the crack** _____ than whether the **crack is in line with a spike**. *

Check all that apply.

- [ ] a) significantly more important
- [ ] b) moderately more important
- [ ] c) equally important
- [ ] d) moderately less important
- [ ] e) significantly less important

13. ix)The **direction of the crack** _____ than the **side of the crack** (field or gauge) *

Check all that apply.

- [ ] a) significantly more important
- [ ] b) moderately more important
- [ ] c) equally important
- [ ] d) moderately less important
- [ ] e) significantly less important

14. x) Whether **the crack is in line with spike** _____ than **the side** (field or gauge) *

Check all that apply.

- [ ] a) significantly more important
- [ ] b) moderately more important
- [ ] c) equally important
- [ ] d) moderately less important
- [ ] e) significantly less important

15. 6. When assessing the severity of **tie cracks of different sizes**, which of the following factors are more important? *

i) The **depth of the crack** is _____ than **the width of the crack**

Check all that apply.

- [ ] a) significantly more important
- [ ] b) moderately more important
- [ ] c) equally important
- [ ] d) moderately less important
- [ ] e) significantly less important

16. ii) The **depth of the crack** is _____ than the **length of the crack** *

Check all that apply.

- [ ] a) significantly more important
- [ ] b) moderately more important
- [ ] c) equally important
- [ ] d) moderately less important
- [ ] e) significantly less important

17. iii) The **depth of the crack** is _____ than the **presence of ballast in the crack** *

Check all that apply.

- [ ] a) significantly more important
- [ ] b) moderately more important
- [ ] c) equally important
- [ ] d) moderately less important
- [ ] e) significantly less important

18. iv) The **width of the crack** is _____ than the **length of the crack** *

Check all that apply.

- [ ] a) significantly more important
- [ ] b) moderately more important
- [ ] c) equally important
- [ ] d) moderately less important
- [ ] e) significantly less important

19. v) The **width of the crack** is _____ than the **presence of ballast in the crack** *

Check all that apply.

- [ ] a) significantly more important
- [ ] b) moderately more important
- [ ] c) equally important
- [ ] d) moderately less important
- [ ] e) significantly less important

20. vi) The **length of the crack** is _____ than the **presence of ballast in the crack** *

Check all that apply.

- [ ] a) significantly more important
- [ ] b) moderately more important
- [ ] c) equally important
- [ ] d) moderately less important
- [ ] e) significantly less important

21. 7. Please mention the years of experience in the field *

Check all that apply.

- [ ] Less than 5 years
- [ ] 5-10 years
- [ ] 11-20 years
- [ ] More than 20 years

22. 8. Please select the location where you work *

Check all that apply.

- [ ] Western Canada
- [ ] Central Canada
- [ ] Eastern Canada
- [ ] Northern Canada
- [ ] International

23. 9. Please mention your area of expertise. If other (Please specify). *

Check all that apply.

- [ ] Maintenance
- [ ] Operations
- [ ] Engineering
- [ ] Safety Compliance
- [ ] Administration
- [ ] Other: _____

24. 10. Please select your role in making decisions. If Other (Please specify) *

Check all that apply.

- [ ] Directly involved
- [ ] Indirectly involved
- [ ] Not involved
- [ ] Other: _____

25. 11. Please indicate which organization you represent. If other (Please specify). *

Check all that apply.

- [ ] Short line
- [ ] Class I
- [ ] Government
- [ ] Academia/ Research
- [ ] Railway supplier or service provider
- [ ] Design/Consultant
- [ ] Other: _____

26. 12. Please share if you have any comments.

_____

_____

_____

_____

_____

27. 13. If you want the copy of the survey results, Please enter your email ID

_____

_____

_____

_____

_____

This content is neither created nor endorsed by Google.

Google Forms

Appendix B. Machine Learning codes for the Condition prediction model

Classification codes to predict defect tag using Logistic Regression, Random Forest XGBoost, and Cat boost

```
#logistic regression to predict defect tag (80% 20%)


import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline


# Load the dataset
data = pd.read_csv('Connect.csv')


# Define the features and the target variable (Defect tag)
features = ['Total car west', 'Total train east', 'Total train west',
'Total deflection',
            'Defect amplitude', 'Class', 'Freight speed', 'Passenger
speed']
X = data[features]
y = data['Defect tag']  # Target variable: Defect tag


# Split the data into training and testing sets (80% training, 20%
testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)


# Create a pipeline to scale the data and then apply logistic regression
pipe = Pipeline([
```

```
    ('scaler', StandardScaler()),  # Scaling the data
    ('logistic', LogisticRegression(class_weight='balanced'))  # Logistic regression
])

# Define the hyperparameters grid for tuning
param_grid = [
    {'logistic__penalty': ['l1'], 'logistic__C': [0.01, 0.1, 1.0, 10, 100], 'logistic__solver': ['saga'], 'logistic__max_iter': [200, 500]},
    {'logistic__penalty': ['l2'], 'logistic__C': [0.01, 0.1, 1.0, 10, 100], 'logistic__solver': ['lbfgs', 'liblinear', 'saga'], 'logistic__max_iter': [200, 500]},
    {'logistic__penalty': ['elasticnet'], 'logistic__C': [0.01, 0.1, 1.0, 10, 100], 'logistic__solver': ['saga'], 'logistic__l1_ratio': [0.5], 'logistic__max_iter': [200, 500]}
]

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=pipe, param_grid=param_grid, cv=5, verbose=1, n_jobs=-1)

# Fit the grid search to the training data
grid_search.fit(X_train, y_train)

# Get the best parameters from the grid search
best_params = grid_search.best_params_
print("Best Parameters:", best_params)

# Use the best estimator to predict on the test set
best_model = grid_search.best_estimator_

# Predictions on the test set
y_pred = best_model.predict(X_test)
```

```
# Calculate training accuracy
train_accuracy = best_model.score(X_train, y_train)

# Calculate test accuracy
test_accuracy = accuracy_score(y_test, y_pred)

# Generate and display a classification report
classification_rep = classification_report(y_test, y_pred)

# Print results
print(f"Training Accuracy: {train_accuracy:.2f}")
print(f"Test Accuracy: {test_accuracy:.2f}")
print("Classification Report:")
print(classification_rep)

# XGBoost code 80/20 split

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, StratifiedKFold,
RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.metrics import accuracy_score, classification_report
from xgboost import XGBClassifier
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Load your DataFrame
df = pd.read_csv('Connect.csv')
```

```
# Step 2: Basic Exploratory Data Analysis (EDA)

# Display basic information about the dataset
print("Dataset Info:")
df.info()  # Shows column names, non-null counts, and data types

# Display basic statistics of the dataset to understand distributions
print("\nDataset Description:")
print(df.describe())  # Provides mean, std, min, max, and quartile values for numerical columns

# Check for any missing values in the dataset
print("\nMissing values in each column:")
print(df.isnull().sum())  # Shows count of missing values in each column

# Check the distribution of the target variable to understand class imbalance
target = 'Defect tag'
print("\nTarget variable distribution:")
print(df[target].value_counts())  # Provides the count of each class in the target column

# Visualize the correlation matrix to understand relationships between features
features = ['Total car west', 'Total train east', 'Total train west', 'Total deflection', 'Defect amplitude', 'Class', 'Freight speed', 'Passenger speed']
plt.figure(figsize=(10, 8))
corr_matrix = df[features].corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Feature Correlation Matrix")
plt.show()
```

```
# Plot the distribution of each feature to visually understand data ranges
for feature in features:
    plt.figure(figsize=(8, 4))
    plt.hist(df[feature], bins=30, color='lightblue', edgecolor='black')
    plt.title(f"Distribution of {feature}")
    plt.xlabel(feature)
    plt.ylabel('Frequency')
    plt.show()

# Step 3: Define features and target variable
X = df[features].values
y = df[target].values

# Step 4: Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 5: Parameter grid for RandomizedSearchCV with regularization parameters
param_grid = {
    'xgbclassifier__max_depth': [3, 4],
    'xgbclassifier__learning_rate': [0.01, 0.1],
    'xgbclassifier__n_estimators': [100, 200],
    'xgbclassifier__subsample': [0.7, 0.8],
    'xgbclassifier__colsample_bytree': [0.8, 0.9],
    'xgbclassifier__reg_lambda': [1, 2, 5],  # L2 regularization
    'xgbclassifier__alpha': [0, 0.5, 1],     # L1 regularization
}

# Step 6: Create a pipeline with standard scaling and XGBoost classifier
```

```
pipeline = make_pipeline(
    StandardScaler(),  # Standardize the data
    XGBClassifier(use_label_encoder=False, eval_metric='logloss',
random_state=42)  # XGBoost Classifier
)


# Step 7: Stratified K-Fold for cross-validation with fewer splits for
faster computation
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)


# Step 8: Use RandomizedSearchCV for faster hyperparameter tuning
randomized_search = RandomizedSearchCV(pipeline,
param_distributions=param_grid, n_iter=10, cv=cv, n_jobs=-1,
scoring='accuracy', random_state=42)
randomized_search.fit(X_train, y_train)


# Step 9: Output the best parameters and cross-validation score
print("Best parameters found: ", randomized_search.best_params_)
print("Best cross-validation score:
{:.2f}".format(randomized_search.best_score_))


# Step 10: Predict on the testing data using the best model
y_pred = randomized_search.best_estimator_.predict(X_test)


# Step 11: Print accuracy and classification report for train and test
sets
train_accuracy = accuracy_score(y_train,
randomized_search.best_estimator_.predict(X_train))
test_accuracy = accuracy_score(y_test, y_pred)
print("Train Accuracy:", train_accuracy)
print("Test Accuracy:", test_accuracy)
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
# Step 12: Get the best XGBoost model from the pipeline
best_xgb_model =
randomized_search.best_estimator_.named_steps['xgbclassifier']

# Step 13: Get feature importances
feature_importances = best_xgb_model.feature_importances_

# Step 14: Plot feature importances to visualize the most important
factors for predictions
plt.figure(figsize=(10, 6))
plt.barh(features, feature_importances, color='skyblue')
plt.xlabel('Feature Importance')
plt.title('Feature Importance')
plt.show()

# Randomforest code 80/20 split

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Load the DataFrame
df = pd.read_csv('Connect.csv')

# Step 2: Basic Exploratory Data Analysis (EDA)
```

```
# Display basic information about the dataset
print("Dataset Info:")
df.info()  # Shows column names, non-null counts, and data types

# Display basic statistics of the dataset to understand distributions
print("\nDataset Description:")
print(df.describe())  # Provides mean, std, min, max, and quartile values for numerical columns

# Check for any missing values in the dataset
print("\nMissing values in each column:")
print(df.isnull().sum())  # Shows count of missing values in each column

# Check the distribution of the target variable to understand class imbalance
target = 'Defect tag'
print("\nTarget variable distribution:")
print(df[target].value_counts())  # Provides the count of each class in the target column

# Visualize the correlation matrix to understand relationships between features
features = ['Total car west', 'Total train east', 'Total train west', 'Total deflection', 'Freight speed', 'Class', 'Passenger speed', 'Defect amplitude']
plt.figure(figsize=(10, 8))
corr_matrix = df[features].corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Feature Correlation Matrix")
plt.show()

# Plot the distribution of each feature to visually understand data ranges
```

```
for feature in features:
    plt.figure(figsize=(8, 4))
    plt.hist(df[feature], bins=30, color='lightblue', edgecolor='black')
    plt.title(f"Distribution of {feature}")
    plt.xlabel(feature)
    plt.ylabel('Frequency')
    plt.show()


# Step 3: Define features and target variable
features = ['Total car west', 'Total train east', 'Total train west',
'Total deflection', 'Freight speed', 'Class', 'Passenger speed', 'Defect
amplitude']
target = 'Defect tag'


# Step 4: Prepare data
X = df[features].values
y = df[target].values


# Step 5: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,
random_state=42)


# Step 6: Create a pipeline with standard scaling and Random Forest
classifier
# Adjusting n_estimators, max_depth, and min_samples_split to manage
complexity and avoid overfitting
model = RandomForestClassifier(n_estimators=100, max_depth=10,
min_samples_split=4, random_state=42)
pipeline = make_pipeline(StandardScaler(), model)


# Step 7: Stratified K-Fold for cross-validation
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

```
# Step 8: Fit the pipeline on the training data
pipeline.fit(X_train, y_train)

# Step 9: Predict on the training data to calculate training accuracy
y_train_pred = pipeline.predict(X_train)
train_accuracy = accuracy_score(y_train, y_train_pred)

# Step 10: Predict on the testing data
y_pred = pipeline.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred)

# Step 11: Print training and test accuracy
print("Training Accuracy:", train_accuracy)
print("Test Accuracy:", test_accuracy)

# Step 12: Print classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Step 13: Get the feature importances from the RandomForest model
importances = model.feature_importances_

# Step 14: Sort the importances and features for plotting
indices = np.argsort(importances)
sorted_features = np.array(features)[indices]
sorted_importances = importances[indices]

# Step 15: Plot feature importances
plt.figure(figsize=(10, 6))
plt.barh(sorted_features, sorted_importances, color='skyblue')
plt.xlabel('Feature Importance')
```

```
plt.title('Feature Importance from RandomForest')
plt.show()

# Catboost code to predict the Defect tag(80% 20% split)
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, StratifiedKFold,
RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.metrics import accuracy_score, classification_report
from catboost import CatBoostClassifier
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Load your DataFrame
df = pd.read_csv('Connect.csv')

# Step 2: Basic Exploratory Data Analysis (EDA)

# Display basic information about the dataset
print("Dataset Info:")
df.info()  # Shows column names, non-null counts, and data types

# Display basic statistics of the dataset to understand distributions
print("\nDataset Description:")
print(df.describe())  # Provides mean, std, min, max, and quartile values
for numerical columns

# Check for any missing values in the dataset
print("\nMissing values in each column:")
print(df.isnull().sum())  # Shows count of missing values in each column
```

```
# Check the distribution of the target variable to understand class
imbalance
target = 'Defect tag'
print("\nTarget variable distribution:")
print(df[target].value_counts())  # Provides the count of each class in
the target column


# Visualize the correlation matrix to understand relationships between
features
features = ['Total car west', 'Total train east', 'Total train west',
'Total deflection', 'Defect amplitude', 'Class', 'Freight speed',
'Passenger speed']
plt.figure(figsize=(10, 8))
corr_matrix = df[features].corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Feature Correlation Matrix")
plt.show()

# Plot the distribution of each feature to visually understand data ranges
for feature in features:
    plt.figure(figsize=(8, 4))
    plt.hist(df[feature], bins=30, color='lightblue', edgecolor='black')
    plt.title(f"Distribution of {feature}")
    plt.xlabel(feature)
    plt.ylabel('Frequency')
    plt.show()

# Step 3: Define features and target variable
X = df[features].values
y = df[target].values
```

```
# Step 4: Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 5: Parameter grid for RandomizedSearchCV with CatBoost parameters
param_grid = {
    'catboostclassifier__depth': [4, 6, 8],
    'catboostclassifier__learning_rate': [0.01, 0.1, 0.2],
    'catboostclassifier__iterations': [100, 200, 300],
    'catboostclassifier__l2_leaf_reg': [1, 3, 5],  # L2 regularization term
    'catboostclassifier__border_count': [32, 64, 128],  # Number of splits for features
}

# Step 6: Create a pipeline with standard scaling and CatBoost classifier
pipeline = make_pipeline(
    StandardScaler(),  # Standardize the data
    CatBoostClassifier(verbose=0, random_state=42)  # CatBoost Classifier
)

# Step 7: Stratified K-Fold for cross-validation
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Step 8: Use RandomizedSearchCV for faster hyperparameter tuning
randomized_search = RandomizedSearchCV(pipeline, param_distributions=param_grid, n_iter=10, cv=cv, n_jobs=-1, scoring='accuracy', random_state=42)
randomized_search.fit(X_train, y_train)

# Step 9: Output the best parameters and cross-validation score
print("Best parameters found: ", randomized_search.best_params_)
```

```
print("Best cross-validation score:
{:.2f}".format(randomized_search.best_score_))


# Step 10: Predict on the testing data using the best model
y_pred = randomized_search.best_estimator_.predict(X_test)


# Step 11: Print accuracy and classification report for train and test
sets
train_accuracy = accuracy_score(y_train,
randomized_search.best_estimator_.predict(X_train))
test_accuracy = accuracy_score(y_test, y_pred)
print("Train Accuracy:", train_accuracy)
print("Test Accuracy:", test_accuracy)
print("Classification Report:")
print(classification_report(y_test, y_pred))


# Step 12: Get the best CatBoost model from the pipeline
best_cat_model =
randomized_search.best_estimator_.named_steps['catboostclassifier']


# Step 13: Get feature importances
feature_importances = best_cat_model.get_feature_importance()


# Step 14: Plot feature importances to visualize the most important
factors for predictions
plt.figure(figsize=(10, 6))
plt.barh(features, feature_importances, color='skyblue')
plt.xlabel('Feature Importance')
plt.title('Feature Importance (CatBoost)')
plt.show()


# XGBoost code to predict the Defect tag (2007 - 2012 training / 2013 test
)
```

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold, RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import learning_curve
from sklearn.pipeline import make_pipeline
from sklearn.metrics import accuracy_score, classification_report
from xgboost import XGBClassifier
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Load the DataFrames
train_df = pd.read_csv('Connecttrain.csv')
test_df = pd.read_csv('Connecttest.csv')

# Step 2: Basic Exploratory Data Analysis (EDA)

# Display basic information about the training and test datasets
print("Training Data Info:")
train_df.info()  # Shows column names, non-null counts, and data types for
the training set


print("\nTest Data Info:")
test_df.info()  # Shows column names, non-null counts, and data types for
the test set


# Display basic statistics of the training and test datasets to understand
distributions
print("\nTraining Data Description:")
print(train_df.describe())  # Provides statistics for numerical columns in
the training set
```

```
print("\nTest Data Description:")
print(test_df.describe())  # Provides statistics for numerical columns in the test set


# Check for any missing values in the training and test datasets
print("\nMissing values in Training Data:")
print(train_df.isnull().sum())  # Shows count of missing values in each column of the training set


print("\nMissing values in Test Data:")
print(test_df.isnull().sum())  # Shows count of missing values in each column of the test set


# Check the distribution of the target variable in both training and test sets to understand class imbalance
target = 'Defect tag'
print("\nTarget variable distribution in Training Data:")
print(train_df[target].value_counts())  # Count of each class in the training set target


print("\nTarget variable distribution in Test Data:")
print(test_df[target].value_counts())  # Count of each class in the test set target


# Visualize the correlation matrix for features in the training set
features = ['Total car west', 'Total train east', 'Total train west', 'Total deflection', 'Defect amplitude', 'Freight speed', 'Passenger speed']
plt.figure(figsize=(10, 8))
corr_matrix = train_df[features].corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Feature Correlation Matrix (Training Data)")
plt.show()
```

```
# Plot the distribution of each feature in the training set to visually
understand data ranges
for feature in features:
    plt.figure(figsize=(8, 4))
    plt.hist(train_df[feature], bins=30, color='lightblue',
edgecolor='black')
    plt.title(f"Distribution of {feature} (Training Data)")
    plt.xlabel(feature)
    plt.ylabel('Frequency')
    plt.show()


# Step 3: Define features and target variable
features = ['Total car west', 'Total train east', 'Total train west',
'Total deflection', 'Defect amplitude', 'Freight speed', 'Passenger
speed']
target = 'Defect tag'


# Step 4: Prepare data for training and testing
X_train = train_df[features].values
y_train = train_df[target].values
X_test = test_df[features].values
y_test = test_df[target].values


# Step 5: Parameter grid for RandomizedSearchCV with regularization
parameters
param_grid = {
    'xgbclassifier__max_depth': [3, 4],
    'xgbclassifier__learning_rate': [0.01, 0.1],
    'xgbclassifier__n_estimators': [100, 200],
    'xgbclassifier__subsample': [0.7, 0.8],
    'xgbclassifier__colsample_bytree': [0.8, 0.9],
    'xgbclassifier__reg_lambda': [1, 2, 5],  # L2 regularization
```

```
    'xgbclassifier__alpha': [0, 0.5, 1],     # L1 regularization
}

# Step 6: Create a pipeline with standard scaling and XGBoost classifier
pipeline = make_pipeline(
    StandardScaler(),
    XGBClassifier(use_label_encoder=False, eval_metric='logloss',
random_state=42)
)

# Step 7: Stratified K-Fold for cross-validation
cv = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)

# Step 8: Use RandomizedSearchCV for faster hyperparameter tuning
randomized_search = RandomizedSearchCV(pipeline,
param_distributions=param_grid, n_iter=10, cv=cv, n_jobs=-1,
scoring='accuracy', random_state=42)
randomized_search.fit(X_train, y_train)

# Step 9: Output the best parameters
print("Best parameters found: ", randomized_search.best_params_)
print("Best cross-validation score:
{:.2f}".format(randomized_search.best_score_))

# Step 10: Learning curve data using the best estimator
train_sizes, train_scores, test_scores = learning_curve(
    randomized_search.best_estimator_, X_train, y_train, cv=cv, n_jobs=-1,
train_sizes=np.linspace(0.1, 1.0, 5), scoring='accuracy')

# Step 11: Predict on the testing data using the best model
y_pred = randomized_search.best_estimator_.predict(X_test)

# Step 12: Print accuracy and classification report
```

```
train_accuracy = accuracy_score(y_train,
randomized_search.best_estimator_.predict(X_train))
test_accuracy = accuracy_score(y_test, y_pred)
print("Train Accuracy:", train_accuracy)
print("Test Accuracy:", test_accuracy)
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Step 13: Get the best XGBoost model from the pipeline
best_xgb_model =
randomized_search.best_estimator_.named_steps['xgbclassifier']

# Step 14: Get feature importances from the best XGBoost model
feature_importances = best_xgb_model.feature_importances_

# Step 15: Plot feature importances
plt.figure(figsize=(10, 6))
plt.barh(features, feature_importances, color='skyblue')
plt.xlabel('Feature Importance')
plt.title('Feature Importance')
plt.show()

# RandomForest code to predict the Defect tag (2007 - 2012 training / 2013
test)
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt
```

```
import seaborn as sns


# Step 1: Load the DataFrames
train_df = pd.read_csv('Connecttrain.csv')
test_df = pd.read_csv('Connecttest.csv')


# Step 2: Basic Exploratory Data Analysis (EDA)


# Display basic information about the training and test datasets
print("Training Data Info:")
train_df.info()  # Shows column names, non-null counts, and data types for
the training set


print("\nTest Data Info:")
test_df.info()  # Shows column names, non-null counts, and data types for
the test set


# Display basic statistics of the training and test datasets to understand
distributions
print("\nTraining Data Description:")
print(train_df.describe())  # Provides statistics for numerical columns in
the training set


print("\nTest Data Description:")
print(test_df.describe())  # Provides statistics for numerical columns in
the test set


# Check for any missing values in the training and test datasets
print("\nMissing values in Training Data:")
print(train_df.isnull().sum())  # Shows count of missing values in each
column of the training set


print("\nMissing values in Test Data:")
```

```
print(test_df.isnull().sum())  # Shows count of missing values in each column of the test set


# Check the distribution of the target variable in both training and test sets to understand class imbalance
target = 'Defect tag'
print("\nTarget variable distribution in Training Data:")
print(train_df[target].value_counts())  # Count of each class in the training set target


print("\nTarget variable distribution in Test Data:")
print(test_df[target].value_counts())  # Count of each class in the test set target


# Visualize the correlation matrix for features in the training set
features = ['Total car west', 'Total train east', 'Total train west', 'Total deflection', 'Defect amplitude', 'Freight speed', 'Passenger speed']
plt.figure(figsize=(10, 8))
corr_matrix = train_df[features].corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Feature Correlation Matrix (Training Data)")
plt.show()


# Plot the distribution of each feature in the training set to visually understand data ranges
for feature in features:
    plt.figure(figsize=(8, 4))
    plt.hist(train_df[feature], bins=30, color='lightblue', edgecolor='black')
    plt.title(f"Distribution of {feature} (Training Data)")
    plt.xlabel(feature)
    plt.ylabel('Frequency')
```

```
    plt.show()


# Step 3: Define features and target variable
features = ['Total car west', 'Total train east', 'Total train west',
'Total deflection', 'Defect amplitude', 'Freight speed', 'Passenger
speed']
target = 'Defect tag'


# Step 4: Prepare data for training and testing
X_train = train_df[features].values
y_train = train_df[target].values
X_test = test_df[features].values
y_test = test_df[target].values


# Step 5: Create a pipeline with standard scaling and Random Forest
classifier
# Adjusting n_estimators, max_depth, and min_samples_split to manage
complexity and avoid overfitting
model = RandomForestClassifier(n_estimators=100, max_depth=10,
min_samples_split=4, random_state=42)
pipeline = make_pipeline(StandardScaler(), model)


# Step 6: Stratified K-Fold for cross-validation
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)


# Step 7: Fit the pipeline on the training data
pipeline.fit(X_train, y_train)


# Step 8: Predict on the training data to calculate training accuracy
y_train_pred = pipeline.predict(X_train)
train_accuracy = accuracy_score(y_train, y_train_pred)


# Step 9: Predict on the testing data
```

```
y_pred = pipeline.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred)

# Step 10: Print training and test accuracy
print("Training Accuracy:", train_accuracy)
print("Test Accuracy:", test_accuracy)

# Step 11: Print classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Step 12: Get the feature importances from the RandomForest model
importances = model.feature_importances_

# Step 13: Sort the importances and features for plotting
indices = np.argsort(importances)
sorted_features = np.array(features)[indices]
sorted_importances = importances[indices]

# Step 14: Plot feature importances
plt.figure(figsize=(10, 6))
plt.barh(sorted_features, sorted_importances, color='skyblue')
plt.xlabel('Feature Importance')
plt.title('Feature Importance from RandomForest')
plt.show()

#catboost to predict the defect tag (2007 - 2012 training / 2013 test)

import pandas as pd
import numpy as np
```

```
from sklearn.model_selection import StratifiedKFold, learning_curve,
RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.metrics import accuracy_score, classification_report
from catboost import CatBoostClassifier
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Load the DataFrames
train_df = pd.read_csv('Connecttrain.csv')
test_df = pd.read_csv('Connecttest.csv')

# Step 2: Basic Exploratory Data Analysis (EDA)

# Display basic information about the training and test datasets
print("Training Data Info:")
train_df.info()

print("\nTest Data Info:")
test_df.info()

# Display basic statistics of the training and test datasets to understand
distributions
print("\nTraining Data Description:")
print(train_df.describe())

print("\nTest Data Description:")
print(test_df.describe())

# Check for any missing values in the training and test datasets
print("\nMissing values in Training Data:")
```

```
print(train_df.isnull().sum())


print("\nMissing values in Test Data:")
print(test_df.isnull().sum())


# Check the distribution of the target variable in both training and test
sets to understand class imbalance
target = 'Defect tag'
print("\nTarget variable distribution in Training Data:")
print(train_df[target].value_counts())


print("\nTarget variable distribution in Test Data:")
print(test_df[target].value_counts())


# Visualize the correlation matrix for features in the training set
features = ['Total car west', 'Total train east', 'Total train west',
'Total deflection', 'Defect amplitude', 'Freight speed', 'Passenger
speed']
plt.figure(figsize=(10, 8))
corr_matrix = train_df[features].corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Feature Correlation Matrix (Training Data)")
plt.show()


# Plot the distribution of each feature in the training set to visually
understand data ranges
for feature in features:
    plt.figure(figsize=(8, 4))
    plt.hist(train_df[feature], bins=30, color='lightblue',
edgecolor='black')
    plt.title(f"Distribution of {feature} (Training Data)")
    plt.xlabel(feature)
    plt.ylabel('Frequency')
```

```
    plt.show()


# Step 3: Define features and target variable
features = ['Total car west', 'Total train east', 'Total train west',
'Total deflection', 'Defect amplitude', 'Freight speed', 'Passenger
speed']
target = 'Defect tag'


# Step 4: Prepare data for training and testing
X_train = train_df[features].values
y_train = train_df[target].values
X_test = test_df[features].values
y_test = test_df[target].values


# Step 5: Parameter grid for RandomizedSearchCV with regularization
parameters
param_grid = {
    'catboostclassifier__depth': [4, 6, 8],
    'catboostclassifier__learning_rate': [0.01, 0.1, 0.2],
    'catboostclassifier__iterations': [100, 200, 300],
    'catboostclassifier__l2_leaf_reg': [1, 3, 5, 7],
    'catboostclassifier__border_count': [128, 160, 256],
    'catboostclassifier__bagging_temperature': [0.5, 1, 2],
}


# Step 6: Create a pipeline with standard scaling and CatBoost classifier
pipeline = make_pipeline(
    StandardScaler(),
    CatBoostClassifier(verbose=0, random_state=42)  # verbose=0 to
suppress output during fitting
)
```

```
# Step 7: Stratified K-Fold for cross-validation
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Step 8: Use RandomizedSearchCV for faster hyperparameter tuning
randomized_search = RandomizedSearchCV(pipeline,
param_distributions=param_grid, n_iter=10, cv=cv, n_jobs=-1,
scoring='accuracy', random_state=42)
randomized_search.fit(X_train, y_train)

# Step 9: Output the best parameters
print("Best parameters found: ", randomized_search.best_params_)
print("Best cross-validation score:
{:.2f}".format(randomized_search.best_score_))

# Step 10: Learning curve data using the best estimator
train_sizes, train_scores, test_scores = learning_curve(
    randomized_search.best_estimator_, X_train, y_train, cv=cv, n_jobs=-1,
train_sizes=np.linspace(0.1, 1.0, 5), scoring='accuracy')

# Step 11: Predict on the testing data using the best model
y_pred = randomized_search.best_estimator_.predict(X_test)

# Step 12: Print accuracy and classification report
train_accuracy = accuracy_score(y_train,
randomized_search.best_estimator_.predict(X_train))
test_accuracy = accuracy_score(y_test, y_pred)
print("Train Accuracy:", train_accuracy)
print("Test Accuracy:", test_accuracy)
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Step 13: Get the best CatBoost model from the pipeline
```

```
best_cat_model =
randomized_search.best_estimator_.named_steps['catboostclassifier']


# Step 14: Get feature importances from the best CatBoost model
feature_importances = best_cat_model.get_feature_importance()


# Step 15: Plot feature importances
plt.figure(figsize=(10, 6))
plt.barh(features, feature_importances, color='skyblue')
plt.xlabel('Feature Importance')
plt.title('Feature Importance (CatBoost)')
plt.show()


# XGBoost codes to predict the defect Tag using Predicted amplitude


import pandas as pd
import xgboost as xgb
from sklearn.model_selection import train_test_split, RandomizedSearchCV,
cross_val_score
from sklearn.metrics import roc_auc_score, classification_report
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt
import seaborn as sns


# Load the dataset
df = pd.read_csv('Pamp.csv')


# Step 2: Define the features (X) and target (Y)
X = df[['Total car east', 'Total car west', 'Total train east', 'Total
train west',
        'Total deflection', 'Class', 'Freight speed', 'Passenger speed',
'Prediction']]
```

```
Y = df['Defect tag']


# Step 3: Use SMOTE to oversample the minority class
smote = SMOTE(random_state=42)
X_resampled, Y_resampled = smote.fit_resample(X, Y)


# Step 4: Split the dataset into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X_resampled,
Y_resampled, test_size=0.2, random_state=42)


# Step 5: Define the XGBoost classifier with manually tuned
scale_pos_weight
xgb_model = xgb.XGBClassifier(scale_pos_weight=1.0)  # Adjust
`scale_pos_weight` if needed


# Step 6: Define the parameter grid for RandomizedSearchCV
param_dist = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.05, 0.1],
    'subsample': [0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0],
    'gamma': [0, 0.1, 0.2],
    'reg_alpha': [0, 0.1, 0.5],
    'reg_lambda': [1, 1.5, 2]
}


# Step 7: Use AUC-ROC as the scoring metric for RandomizedSearchCV
random_search = RandomizedSearchCV(xgb_model,
param_distributions=param_dist,
                                   n_iter=20, scoring='roc_auc', cv=5,
random_state=42, n_jobs=-1, verbose=2)
```

```
# Step 8: Fit the model
random_search.fit(X_train, Y_train)

# Step 9: Best parameters found by RandomizedSearchCV
best_params = random_search.best_params_
print(f'Best Parameters: {best_params}')

# Step 10: Use the best model found
best_model = random_search.best_estimator_

# Step 11: Make predictions on the training set
Y_train_pred = best_model.predict(X_train)

# Step 12: Make predictions on the test set
Y_test_pred = best_model.predict(X_test)

# Step 13: Calculate the AUC-ROC score
roc_auc_train = roc_auc_score(Y_train, Y_train_pred)
roc_auc_test = roc_auc_score(Y_test, Y_test_pred)
print(f'Training AUC-ROC Score: {roc_auc_train}')
print(f'Test AUC-ROC Score: {roc_auc_test}')

# Step 14: Cross-validation with 5 folds using the best model
cv_scores = cross_val_score(best_model, X_resampled, Y_resampled, cv=5,
scoring='roc_auc')
print(f'Cross-Validation Scores (AUC-ROC): {cv_scores}')
print(f'Mean Cross-Validation Score (AUC-ROC): {cv_scores.mean()}')

# Step 15: Classification report for the test set
report = classification_report(Y_test, Y_test_pred)
print('Classification Report:')
```

```
print(report)


# Random Forest codes to predict the defect Tag using Predicted amplitude


import pandas as pd
from sklearn.model_selection import train_test_split, RandomizedSearchCV,
cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
df = pd.read_csv('Pamp.csv')

# Step 1: Basic Exploratory Data Analysis (EDA)

# Display basic information about the dataset
print("Dataset Info:")
df.info()  # Shows column names, non-null counts, and data types

# Display basic statistics of the dataset to understand distributions
print("\nDataset Description:")
print(df.describe())  # Provides mean, std, min, max, and quartile values
for numerical columns

# Check for any missing values in the dataset
print("\nMissing values in each column:")
print(df.isnull().sum())  # Shows count of missing values in each column
```

```
# Check the distribution of the target variable to understand class
imbalance
target = 'Defect tag'
print("\nTarget variable distribution:")
print(df[target].value_counts())  # Provides the count of each class in
the target column


# Visualize the correlation matrix to understand relationships between
features
features = ['Total car east', 'Total car west', 'Total train east', 'Total
train west',
            'Total deflection', 'Class', 'Freight speed', 'Passenger
speed', 'Prediction']
plt.figure(figsize=(10, 8))
corr_matrix = df[features].corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Feature Correlation Matrix")
plt.show()


# Optional: Plot the distribution of each feature to visually understand
data ranges
for feature in features:
    plt.figure(figsize=(8, 4))
    plt.hist(df[feature], bins=30, color='lightblue', edgecolor='black')
    plt.title(f"Distribution of {feature}")
    plt.xlabel(feature)
    plt.ylabel('Frequency')
    plt.show()


# Step 2: Define the features (X) and target (Y)
X = df[['Total car east', 'Total car west', 'Total train east', 'Total
train west',
```

```
        'Total deflection', 'Class', 'Freight speed', 'Passenger
speed','Defect type','Prediction']]
Y = df['Defect tag']


# Step 3: Apply SMOTE to oversample the minority class
smote = SMOTE(random_state=42)
X_resampled, Y_resampled = smote.fit_resample(X, Y)


# Step 4: Split the resampled dataset into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X_resampled,
Y_resampled, test_size=0.2, random_state=42)


# Step 5: Set up a pipeline with scaling and Random Forest
pipeline = Pipeline([
    ('scaler', StandardScaler()),  # Feature scaling
    ('rf', RandomForestClassifier(random_state=42))  # Random forest
classifier
])


# Step 6: Define the parameter grid for RandomizedSearchCV
param_dist = {
    'rf__n_estimators': [50, 100, 200],
    'rf__max_depth': [None, 10, 20, 30],
    'rf__min_samples_split': [2, 5, 10],
    'rf__min_samples_leaf': [1, 2, 4],
    'rf__bootstrap': [True, False]
}


# Step 7: RandomizedSearchCV to find the best hyperparameters
random_search = RandomizedSearchCV(pipeline,
param_distributions=param_dist,
                                   n_iter=20, scoring='accuracy', cv=5,
random_state=42, n_jobs=-1, verbose=2)
```

```
# Step 8: Fit the model
random_search.fit(X_train, Y_train)

# Step 9: Best parameters found by RandomizedSearchCV
best_params = random_search.best_params_
print(f'Best Parameters: {best_params}')

# Step 10: Use the best model found
best_model = random_search.best_estimator_

# Step 11: Make predictions on the training set
Y_train_pred = best_model.predict(X_train)

# Step 12: Make predictions on the test set
Y_test_pred = best_model.predict(X_test)

# Step 13: Calculate test accuracy
test_accuracy = accuracy_score(Y_test, Y_test_pred)
print(f'Test Accuracy: {test_accuracy}')

# Step 14: Cross-validation with 5 folds using the best model
cv_scores = cross_val_score(best_model, X_resampled, Y_resampled, cv=5,
scoring='accuracy')
print(f'Cross-Validation Scores: {cv_scores}')
print(f'Mean Cross-Validation Score: {cv_scores.mean()}')

# Step 15: Classification report for the test set
report = classification_report(Y_test, Y_test_pred)
print('Classification Report:')
print(report)
```

```
# CatBoost codes to predict the defect Tag using Predicted amplitude


import pandas as pd

from catboost import CatBoostClassifier

from sklearn.model_selection import train_test_split, RandomizedSearchCV,
StratifiedKFold, cross_val_score

from sklearn.metrics import roc_auc_score, classification_report,
accuracy_score

from imblearn.over_sampling import SMOTE


# Load the dataset

df = pd.read_csv('Pamp.csv')


# Step 2: Define the features (X) and target (Y)

X = df[['Total car east', 'Total car west', 'Total train east', 'Total
train west',

        'Total deflection', 'Class', 'Freight speed', 'Passenger speed',
'Prediction']]

Y = df['Defect tag']


# Step 3: Use SMOTE to oversample the minority class

smote = SMOTE(random_state=42)

X_resampled, Y_resampled = smote.fit_resample(X, Y)


# Step 4: Split the dataset into training and testing sets

X_train, X_test, Y_train, Y_test = train_test_split(X_resampled,
Y_resampled, test_size=0.2, random_state=42)


# Step 5: Define the CatBoost model

catboost_model = CatBoostClassifier(random_state=42, silent=True)
```

```
# Step 6: Define the parameter grid for hyperparameter tuning, including class weights
catboost_param_grid = {
    'iterations': [500, 700, 1000],
    'depth': [6, 8],
    'learning_rate': [0.01, 0.03],
    'l2_leaf_reg': [9, 11, 13],
    'border_count': [128, 254],
    'bagging_temperature': [0.5, 1.0],
    'class_weights': [{0: 1, 1: 1.2}, {0: 1, 1: 1.1}]  # Further lowering class weight for class 1
}

# Step 7: Hyperparameter tuning using RandomizedSearchCV with stratified k-fold cross-validation
skf = StratifiedKFold(n_splits=5)
catboost_random_search = RandomizedSearchCV(
    estimator=catboost_model,
    param_distributions=catboost_param_grid,
    n_iter=50,
    cv=skf,
    verbose=2,
    random_state=42,
    scoring='roc_auc'
)

# Step 8: Fit the CatBoost model with hyperparameter tuning
catboost_random_search.fit(X_train, Y_train)
print(f"Best CatBoost Parameters: {catboost_random_search.best_params_}")

# Step 9: Use the best model for predictions
catboost_best_model = catboost_random_search.best_estimator_
```

```
# Step 10: Make predictions on the test set
Y_test_pred = catboost_best_model.predict(X_test)
Y_train_pred = catboost_best_model.predict(X_train)

# Step 11: Calculate and print the AUC-ROC score and classification report
roc_auc_test = roc_auc_score(Y_test, Y_test_pred)
print(f'Test AUC-ROC Score: {roc_auc_test}')

# Classification report
report = classification_report(Y_test, Y_test_pred)
print('Classification Report:')
print(report)

# Step 12: Calculate and print training and test accuracy
train_accuracy = accuracy_score(Y_train, Y_train_pred)
test_accuracy = accuracy_score(Y_test, Y_test_pred)
print(f'Training Accuracy: {train_accuracy}')
print(f'Test Accuracy: {test_accuracy}')
```

Classification model codes to predict the defect type using Logistic Regression, XGBoost, Random Forest and Cat Boost

```
#logistic Regression to predict defect type (80% / 20%)
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score,
make_scorer, f1_score, precision_score, recall_score
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
```

```
# Load the dataset
file_path = 'Connect.csv'  # Make sure this file path is correct for your dataset
data = pd.read_csv(file_path)


# Define the features and the target variable (Defect type)
features = ['Line segment number', 'Track standard number', 'Milepost', 'Total train east',
            'Total train west', 'Total deflection', 'Class', 'Freight speed', 'Passenger speed']
X = data[features]
y = data['Defect type']  # Target variable: Defect type


# Split the data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Create a pipeline to scale the data and then apply logistic regression
pipe = Pipeline([
    ('scaler', StandardScaler()),  # Scaling the data
    ('logistic', LogisticRegression(class_weight='balanced'))  # Logistic regression
])


# Extend the hyperparameters grid for wider tuning
param_grid = [
    {'logistic__penalty': ['l1', 'l2'],
     'logistic__C': [0.001, 0.01, 0.1, 1.0, 10, 100],
     'logistic__solver': ['liblinear', 'saga'],
     'logistic__max_iter': [200, 500, 1000]},
    {'logistic__penalty': ['elasticnet'],
     'logistic__C': [0.001, 0.01, 0.1, 1.0, 10, 100],
```

```
    'logistic__solver': ['saga'],
    'logistic__l1_ratio': [0.5, 0.7, 0.9],
    'logistic__max_iter': [200, 500, 1000]}
]

# Custom scorers to evaluate accuracy, F1-macro, precision, and recall
scorers = {
    'accuracy': 'accuracy',
    'f1_macro': make_scorer(f1_score, average='macro'),
    'precision_macro': make_scorer(precision_score, average='macro'),
    'recall_macro': make_scorer(recall_score, average='macro')
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=pipe, param_grid=param_grid, cv=5,
scoring=scorers, refit='f1_macro', verbose=1, n_jobs=-1)

# Fit the grid search to the training data
grid_search.fit(X_train, y_train)

# Get the best parameters from the grid search
best_params = grid_search.best_params_
print("Best Parameters:", best_params)

# Use the best estimator to predict on the test set
best_model = grid_search.best_estimator_

# Predictions on the test set
y_pred = best_model.predict(X_test)

# Calculate training accuracy
```

```
train_accuracy = best_model.score(X_train, y_train)

# Calculate test accuracy
test_accuracy = accuracy_score(y_test, y_pred)

# Generate and display a classification report
classification_rep = classification_report(y_test, y_pred)

# Print results
print(f"Training Accuracy: {train_accuracy:.2f}")
print(f"Test Accuracy: {test_accuracy:.2f}")
print("Classification Report:")
print(classification_rep)

# Cross-validation scores
print("Cross-validation scores:")
for scorer in scorers:
    print(f"{scorer}:
{grid_search.cv_results_[f'mean_test_{scorer}'].max()}")

#XGBoost code to predict defect type (80% 20% split)
import pandas as pd
from sklearn.model_selection import train_test_split, RandomizedSearchCV,
StratifiedKFold
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
from xgboost import XGBClassifier
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline
import matplotlib.pyplot as plt
import seaborn as sns
```

```
import numpy as np

# Load the dataset
df = pd.read_csv('Connect.csv')

# Step 1: Exploratory Data Analysis (EDA)
# Check the basic statistics of the dataset
print("Dataset Statistics:\n", df.describe())

# Check for missing values
print("Missing Values:\n", df.isnull().sum())

# Plot distribution of target variable ('Defect type')
plt.figure(figsize=(8, 6))
sns.countplot(x='Defect type', data=df)
plt.title('Distribution of Defect Type')
plt.show()

# Step 2: Correlation Matrix
# Calculate and display the correlation matrix for the features
features = ['Line segment number', 'Track standard number', 'Milepost',
'Total train east',
            'Total train west', 'Total deflection', 'Class', 'Freight
speed', 'Passenger speed']

plt.figure(figsize=(10, 8))
corr_matrix = df[features].corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Feature Correlation Matrix")
plt.show()

# Step 3: Feature Engineering
```

```
# Create a new feature as an interaction between two features ('Total car east' and 'Total car west')
df['Total cars'] = df['Total car east'] + df['Total car west']


# Step 4: Define Features and Target Variable
features = ['Line segment number', 'Track standard number', 'Milepost', 'Total cars', 'Total train east',
            'Total train west', 'Total deflection', 'Class', 'Freight speed', 'Passenger speed']
target = 'Defect type'


# Prepare the feature matrix (X) and target vector (y)
X = df[features].values
y = df[target].values


# Step 5: Split Data into Training and Testing Sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=0, stratify=y)


# Step 6: Create a Pipeline with SMOTE, StandardScaler, and XGBClassifier
pipeline = Pipeline([
    ('smote', SMOTE(random_state=42)),  # SMOTE for oversampling the minority class
    ('scaler', StandardScaler()),  # Standardize the features
    ('classifier', XGBClassifier(random_state=42, use_label_encoder=False, eval_metric='mlogloss'))  # XGBoost Classifier
])


# Step 7: Define the Parameter Grid for Hyperparameter Tuning
param_grid = {
    'classifier__n_estimators': [100, 200],
    'classifier__max_depth': [3, 5, 7],  # Smaller depth for gradient boosting
```

```
    'classifier__learning_rate': [0.01, 0.1, 0.2],
    'classifier__subsample': [0.8, 1.0],
    'classifier__colsample_bytree': [0.8, 1.0]
}

# Step 8: Use RandomizedSearchCV with StratifiedKFold for Cross-Validation
stratified_kfold = StratifiedKFold(n_splits=5, shuffle=True,
random_state=42)  # 5-fold stratified cross-validation
random_search = RandomizedSearchCV(pipeline,
param_distributions=param_grid, n_iter=20, cv=stratified_kfold,
                                   scoring='accuracy', n_jobs=-1,
verbose=2, random_state=42)

# Step 9: Fit the Randomized Search to the Training Data
random_search.fit(X_train, y_train)

# Step 10: Get the Best Parameters from the Randomized Search
best_params = random_search.best_params_
print("Best parameters found: ", best_params)

# Step 11: Get the Best Model from RandomizedSearchCV
best_model = random_search.best_estimator_

# Step 12: Make Predictions on Both Training and Test Sets
y_train_pred = best_model.predict(X_train)
y_test_pred = best_model.predict(X_test)

# Step 13: Calculate and Print Training Accuracy
training_accuracy = accuracy_score(y_train, y_train_pred)
print(f"Training Accuracy: {training_accuracy:.4f}")

# Step 14: Calculate and Print Test Accuracy
```

```
test_accuracy = accuracy_score(y_test, y_test_pred)
print(f"Test Accuracy: {test_accuracy:.4f}")

# Step 15: Print the Classification Report for the Test Set
print("Classification Report:")
print(classification_report(y_test, y_test_pred))

# Step 16: Plot the Confusion Matrix for the Test Set
conf_matrix = confusion_matrix(y_test, y_test_pred)
plt.figure(figsize=(6, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=np.unique(y), yticklabels=np.unique(y))
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# Step 17: Extract and Plot Feature Importances from the Best Model
importances = best_model.named_steps['classifier'].feature_importances_
indices = np.argsort(importances)[::-1]

# Step 18: Plot the Feature Importances
plt.figure(figsize=(10, 6))
plt.title('Feature Importances')
sns.barplot(y=np.array(features)[indices], x=importances[indices],
orient='h')
plt.xlabel('Relative Importance')
plt.show()

#Random Forest code to predict defect type (80% 20% split)
```

```
from imblearn.over_sampling import SMOTE
from imblearn.ensemble import BalancedRandomForestClassifier
from sklearn.model_selection import train_test_split, StratifiedKFold,
cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import RFE
from sklearn.preprocessing import StandardScaler
from imblearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np


# Step 1: Define features and target variable
# Assuming df is your DataFrame containing the data
features = ['Line segment number', 'Track standard number', 'Milepost',
'Total car east', 'Total car west', 'Total train east',
            'Total train west', 'Total deflection', 'Class', 'Freight
speed', 'Passenger speed']
target = 'Defect type'


# Step 2: Feature Engineering
# Creating a new feature 'Total cars' as the sum of 'Total car east' and
'Total car west'
df['Total cars'] = df['Total car east'] + df['Total car west']


# Update features list to include the new feature 'Total cars'
features = ['Line segment number', 'Track standard number', 'Milepost',
'Total cars', 'Total train east',
            'Total train west', 'Total deflection', 'Class', 'Freight
speed', 'Passenger speed']
```

```
# Step 3: Prepare the data
X = df[features].values  # Feature matrix
y = df[target].values    # Target variable


# Step 4: Perform feature selection using Recursive Feature Elimination
(RFE)
# Selecting the top 8 features using RandomForestClassifier as the
estimator
rfe_selector = RFE(estimator=RandomForestClassifier(),
n_features_to_select=8, step=1)
X_selected = rfe_selector.fit_transform(X, y)  # Fit and transform the
feature matrix
selected_features = [features[i] for i in range(len(features)) if
rfe_selector.support_[i]]  # Keep only selected features


# Step 5: Split the data into training and testing sets (80% training, 20%
testing)
X_train, X_test, y_train, y_test = train_test_split(X_selected, y,
test_size=0.20, random_state=0, stratify=y)


# Step 6: Apply SMOTE (Synthetic Minority Over-sampling Technique) to
balance class 0
# Increase the number of samples in class 0 to twice the mean of all class
samples
smote = SMOTE(sampling_strategy={0: int(np.mean(np.bincount(y)) * 2)},
random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)


# Step 7: Define the BalancedRandomForestClassifier model
brf = BalancedRandomForestClassifier(
    random_state=42,
    n_estimators=100,         # Number of trees
    max_depth=15,             # Maximum depth of the tree
```

```
    min_samples_split=5,      # Minimum samples required to split an
internal node
    min_samples_leaf=2,       # Minimum samples required to be at a leaf
node
    sampling_strategy='all',  # Sampling strategy for balancing classes
    replacement=True,         # Adopt future behavior: allow replacement
in sampling
    bootstrap=False           # Disable bootstrap sampling
)


# Step 8: Create a pipeline with StandardScaler (to standardize features)
and BalancedRandomForestClassifier
pipeline = Pipeline([
    ('scaler', StandardScaler()),  # Step to standardize features
    ('classifier', brf)            # Balanced Random Forest Classifier
])


# Step 9: Implement StratifiedKFold for stratified cross-validation (cv=5)
stratified_kfold = StratifiedKFold(n_splits=5, shuffle=True,
random_state=42)


# Step 10: Perform cross-validation with 5 stratified folds
cross_val_scores = cross_val_score(pipeline, X_train_smote, y_train_smote,
cv=stratified_kfold, scoring='accuracy')


# Step 11: Output the cross-validation results
print(f"Cross-Validation Scores (Accuracy): {cross_val_scores}")
print(f"Mean Cross-Validation Score (Accuracy):
{cross_val_scores.mean():.4f}")


# Step 12: Fit the pipeline to the SMOTE-enhanced training data
pipeline.fit(X_train_smote, y_train_smote)
```

```
# Step 13: Make predictions on both the training and test sets
y_train_pred = pipeline.predict(X_train)  # Predictions on the training set
y_test_pred = pipeline.predict(X_test)    # Predictions on the test set

# Step 14: Calculate and Print Training Accuracy
training_accuracy = accuracy_score(y_train, y_train_pred)
print(f"Training Accuracy: {training_accuracy:.4f}")

# Step 15: Calculate and print the test accuracy
test_accuracy = accuracy_score(y_test, y_test_pred)
print("Test Accuracy:", test_accuracy)

# Step 16: Print the classification report for the test set
print("Classification Report:")
print(classification_report(y_test, y_test_pred))

# Step 17: Extract and plot feature importances from the BalancedRandomForestClassifier
importances = pipeline.named_steps['classifier'].feature_importances_  # Get feature importances
indices = np.argsort(importances)[::-1]  # Sort feature importances in descending order

# Step 18: Plot the feature importances
plt.figure(figsize=(10, 6))
plt.title('Feature Importances')
sns.barplot(y=np.array(selected_features)[indices], x=importances[indices], orient='h')  # Plot using the selected features
plt.xlabel('Relative Importance')
plt.show()

#catBoost code to predict Defect Type (80/ 20%)
```

```
import pandas as pd
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
from catboost import CatBoostClassifier
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Load the dataset
df = pd.read_csv('Connect.csv')

# Step 1: Exploratory Data Analysis (EDA)
# Check the basic statistics of the dataset
print("Dataset Statistics:\n", df.describe())

# Check for missing values
print("Missing Values:\n", df.isnull().sum())

# Plot distribution of target variable ('Defect type')
plt.figure(figsize=(8, 6))
sns.countplot(x='Defect type', data=df)
plt.title('Distribution of Defect Type')
plt.show()

# Step 2: Correlation Matrix
features = ['Line segment number', 'Track standard number', 'Milepost',
'Total train east',
```

```
            'Total train west', 'Total deflection', 'Class', 'Freight
speed', 'Passenger speed']


plt.figure(figsize=(10, 8))
corr_matrix = df[features].corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Feature Correlation Matrix")
plt.show()


# Step 3: Feature Engineering
# Create a new feature as an interaction between two features ('Total car
east' and 'Total car west')
df['Total cars'] = df['Total car east'] + df['Total car west']


# Step 4: Define Features and Target Variable
features = ['Line segment number', 'Track standard number', 'Milepost',
'Total cars', 'Total train east',
            'Total train west', 'Total deflection', 'Class', 'Freight
speed', 'Passenger speed']
target = 'Defect type'


# Prepare the feature matrix (X) and target vector (y)
X = df[features].values
y = df[target].values


# Step 5: Split Data into Training and Testing Sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,
random_state=0, stratify=y)


# Step 6: Create a Pipeline with StandardScaler, SMOTE, and
CatBoostClassifier
# Adding class_weights to handle class imbalance
class_weights = [1, 1.5, 2]
```

```
pipeline = Pipeline([
    ('smote', SMOTE(random_state=42)),  # SMOTE for oversampling the minority class
    ('scaler', StandardScaler()),  # Standardize the features
    ('classifier', CatBoostClassifier(verbose=0, random_state=42, class_weights=class_weights))  # CatBoost Classifier
])

# Step 7: Implement StratifiedKFold with 5 folds
stratified_kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# To store the scores for each fold
fold_accuracies = []

# Perform stratified k-fold cross-validation
for train_idx, val_idx in stratified_kfold.split(X_train, y_train):
    # Get training and validation sets for this fold
    X_train_fold, X_val_fold = X_train[train_idx], X_train[val_idx]
    y_train_fold, y_val_fold = y_train[train_idx], y_train[val_idx]

    # Fit the pipeline to the fold training data
    pipeline.fit(X_train_fold, y_train_fold)

    # Make predictions on the validation set
    y_val_pred = pipeline.predict(X_val_fold)

    # Calculate accuracy for the current fold
    fold_accuracy = accuracy_score(y_val_fold, y_val_pred)
    fold_accuracies.append(fold_accuracy)
```

```
    print(f"Fold Accuracy: {fold_accuracy:.4f}")

# Step 8: Average accuracy across all 5 folds
average_accuracy = np.mean(fold_accuracies)
print(f"Average Stratified K-Fold Accuracy: {average_accuracy:.4f}")

# Step 9: Fit the model to the entire training set after cross-validation
pipeline.fit(X_train, y_train)

# Step 10: Make Predictions on the Test Set
y_test_pred = pipeline.predict(X_test)

# Step 11: Calculate and Print Test Accuracy
test_accuracy = accuracy_score(y_test, y_test_pred)
print(f"Test Accuracy: {test_accuracy:.4f}")

# Step 12: Print the Classification Report for the Test Set
print("Classification Report:")
print(classification_report(y_test, y_test_pred))

# Step 13: Plot the Confusion Matrix for the Test Set
conf_matrix = confusion_matrix(y_test, y_test_pred)
plt.figure(figsize=(6, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=np.unique(y_test), yticklabels=np.unique(y_test))
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# Step 14: Extract and Plot Feature Importances from the Best Model
```

```
importances = pipeline.named_steps['classifier'].get_feature_importance()
indices = np.argsort(importances)[::-1]

# Step 15: Plot the Feature Importances
plt.figure(figsize=(10, 6))
plt.title('Feature Importances')
sns.barplot(y=np.array(features)[indices], x=importances[indices],
orient='h')
plt.xlabel('Relative Importance')
plt.show()



# XGBoost code to predict Defect Type (2007 to 2012 - Training Data / 2013
- Test data)

import pandas as pd
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV, StratifiedKFold
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Step 1: Load the train and test datasets
train_df = pd.read_csv('Connecttrain.csv')
test_df = pd.read_csv('Connecttest.csv')

# Step 2: Define features and target variable
```

```
features = ['Line segment number', 'Track standard number', 'Milepost',
'Total car east', 'Total car west', 'Total train east',
            'Total train west', 'Total deflection', 'Class', 'Freight
speed', 'Passenger speed']
target = 'Defect type'


# Step 3: Feature Engineering - Create 'Total cars' feature
train_df['Total cars'] = train_df['Total car east'] + train_df['Total car
west']
test_df['Total cars'] = test_df['Total car east'] + test_df['Total car
west']


# Update the feature list to include the newly created 'Total cars'
feature
features = ['Line segment number', 'Track standard number', 'Milepost',
'Total cars', 'Total train east',
            'Total train west', 'Total deflection', 'Class', 'Freight
speed', 'Passenger speed']


# Step 4: Prepare the data
X_train = train_df[features].values  # Feature matrix for training
y_train = train_df[target].values    # Target variable for training
X_test = test_df[features].values    # Feature matrix for testing
y_test = test_df[target].values      # Target variable for testing


# Step 5: Create a Pipeline with SMOTE and XGBClassifier
pipeline = Pipeline([
    ('smote', SMOTE(random_state=42)),  # Apply SMOTE to handle class
imbalance
    ('classifier', XGBClassifier(use_label_encoder=False,
eval_metric='mlogloss', random_state=42))  # XGBoost Classifier
])
```

```
# Step 6: Define the Parameter Grid for Hyperparameter Tuning, focusing on
scale_pos_weight
param_grid = {
    'classifier__n_estimators': [100, 200],
    'classifier__max_depth': [3, 5, 7, 9],
    'classifier__learning_rate': [0.01, 0.1, 0.2],
    'classifier__subsample': [0.8, 1.0],
    'classifier__colsample_bytree': [0.6, 0.8],
    'classifier__reg_alpha': [0, 0.1, 0.5],  # L1 regularization
    'classifier__reg_lambda': [1.0, 1.5, 2.0],  # L2 regularization
    'classifier__scale_pos_weight': [3, 5, 7, 10]  # Further tuning
scale_pos_weight
}


# Step 7: Use Stratified K-Fold Cross-Validation with 5 folds
stratified_kfold = StratifiedKFold(n_splits=5, shuffle=True,
random_state=42)


# Step 8: Use RandomizedSearchCV for Hyperparameter Tuning with Stratified
K-Fold
random_search = RandomizedSearchCV(pipeline,
param_distributions=param_grid, n_iter=30, cv=stratified_kfold,
                                   scoring='accuracy', n_jobs=-1,
verbose=2, random_state=42)


# Step 9: Fit the Randomized Search to the Training Data
random_search.fit(X_train, y_train)


# Step 10: Get the Best Parameters from the Randomized Search
best_params = random_search.best_params_
print("Best parameters found: ", best_params)


# Step 11: Get the Best Model from RandomizedSearchCV
```

```
best_model = random_search.best_estimator_

# Step 12: Make Predictions on Both Training and Test Sets
y_train_pred = best_model.predict(X_train)
y_test_pred = best_model.predict(X_test)

# Step 13: Calculate and Print Training Accuracy
training_accuracy = accuracy_score(y_train, y_train_pred)
print(f"Training Accuracy: {training_accuracy:.4f}")

# Step 14: Calculate and Print Test Accuracy
test_accuracy = accuracy_score(y_test, y_test_pred)
print(f"Test Accuracy: {test_accuracy:.4f}")

# Step 15: Print the Classification Report for the Test Set
print("Classification Report:")
print(classification_report(y_test, y_test_pred))

# Step 16: Plot the Confusion Matrix for the Test Set
conf_matrix = confusion_matrix(y_test, y_test_pred)
plt.figure(figsize=(6, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=np.unique(y_test), yticklabels=np.unique(y_test))
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# Random forest code to predict Defect Type (2007 to 2012 - Training Data
/ 2013 - Test data)
import pandas as pd
```

```
from imblearn.over_sampling import SMOTE
from imblearn.ensemble import BalancedRandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV, StratifiedKFold
from sklearn.feature_selection import RFE
from sklearn.preprocessing import StandardScaler
from imblearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np


# Step 1: Load the training and test datasets
train_df = pd.read_csv('Connecttrain.csv')
test_df = pd.read_csv('Connecttest.csv')


# Step 2: Define features and target variable
features = ['Line segment number', 'Track standard number', 'Milepost',
'Total car east', 'Total car west', 'Total train east',
            'Total train west', 'Total deflection', 'Class', 'Freight
speed', 'Passenger speed']
target = 'Defect type'


# Step 3: Feature Engineering - Create 'Total cars' feature
train_df['Total cars'] = train_df['Total car east'] + train_df['Total car
west']
test_df['Total cars'] = test_df['Total car east'] + test_df['Total car
west']


# Update features list to include 'Total cars'
features = ['Line segment number', 'Track standard number', 'Milepost',
'Total cars', 'Total train east',
```

```
            'Total train west', 'Total deflection', 'Class', 'Freight
speed', 'Passenger speed']


# Step 4: Prepare the data
X_train = train_df[features].values
y_train = train_df[target].values
X_test = test_df[features].values
y_test = test_df[target].values


# Step 5: Feature Selection using RFE
rfe_selector = RFE(estimator=BalancedRandomForestClassifier(),
n_features_to_select=8, step=1)
X_train_selected = rfe_selector.fit_transform(X_train, y_train)
X_test_selected = rfe_selector.transform(X_test)


# Step 6: Adjust SMOTE sampling strategy (Oversample class 0 more
aggressively)
smote = SMOTE(sampling_strategy={0: int(np.mean(np.bincount(y_train)) *
3)}, random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train_selected,
y_train)


# Step 7: Define the BalancedRandomForestClassifier with reduced
complexity and fixed warnings
brf = BalancedRandomForestClassifier(
    random_state=42,
    n_estimators=100,         # Number of trees in the forest
    max_depth=10,             # Reduced depth to prevent overfitting
    min_samples_split=10,     # Force trees to split on more samples
    min_samples_leaf=5,       # Minimum number of samples required to be
at a leaf node
    sampling_strategy='all',  # Explicitly set the future sampling
strategy
```

```
    replacement=True,          # Replacement set to True to avoid future
warning
    bootstrap=False            # Set to False to silence the future warning
)


# Step 8: Define the parameter grid for RandomizedSearchCV (Simplified)
param_grid = {
    'n_estimators': [100, 150],
    'max_depth': [5, 10],
    'min_samples_split': [10, 15],
    'min_samples_leaf': [5, 10]
}


# Step 9: Use StratifiedKFold with 5 folds
stratified_kfold = StratifiedKFold(n_splits=5, shuffle=True,
random_state=42)


# Step 10: RandomizedSearchCV for hyperparameter tuning with Stratified K-
Fold
random_search = RandomizedSearchCV(estimator=brf,
param_distributions=param_grid, n_iter=10, cv=stratified_kfold, verbose=2,
random_state=42, n_jobs=-1)


# Step 11: Create a pipeline with StandardScaler and RandomizedSearchCV
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', random_search)
])


# Step 12: Fit the pipeline to the SMOTE-enhanced training data
pipeline.fit(X_train_smote, y_train_smote)


# Step 13: Make predictions on both the training and test sets
```

```
y_train_pred = pipeline.predict(X_train_selected)
y_test_pred = pipeline.predict(X_test_selected)

# Step 14: Calculate and print training accuracy
training_accuracy = accuracy_score(y_train, y_train_pred)
print(f"Training Accuracy: {training_accuracy:.4f}")

# Step 15: Calculate and print test accuracy
test_accuracy = accuracy_score(y_test, y_test_pred)
print(f"Test Accuracy: {test_accuracy:.4f}")

# Step 16: Print the classification report for the test set
print("Classification Report:")
print(classification_report(y_test, y_test_pred))

# Step 17: Plot the confusion matrix for the test set
conf_matrix = confusion_matrix(y_test, y_test_pred)
plt.figure(figsize=(6, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=np.unique(y_test), yticklabels=np.unique(y_test))
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

#Catboost code to predict Defect Type (2007 to 2012 - Training Data / 2013
- Test data)

import pandas as pd
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
from catboost import CatBoostClassifier
```

```
from sklearn.model_selection import RandomizedSearchCV
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Step 1: Load the train and test datasets
train_df = pd.read_csv('Connecttrain.csv')
test_df = pd.read_csv('Connecttest.csv')

# Step 2: Define features and target variable
features = ['Line segment number', 'Track standard number', 'Milepost',
'Total car east', 'Total car west', 'Total train east',
            'Total train west', 'Total deflection', 'Class', 'Freight
speed', 'Passenger speed']
target = 'Defect type'

# Step 3: Feature Engineering - Create 'Total cars' feature
train_df['Total cars'] = train_df['Total car east'] + train_df['Total car
west']
test_df['Total cars'] = test_df['Total car east'] + test_df['Total car
west']

# Update the feature list to include the newly created 'Total cars'
feature
features = ['Line segment number', 'Track standard number', 'Milepost',
'Total cars', 'Total train east',
            'Total train west', 'Total deflection', 'Class', 'Freight
speed', 'Passenger speed']

# Step 4: Prepare the data
X_train = train_df[features].values  # Feature matrix for training
```

```
y_train = train_df[target].values    # Target variable for training
X_test = test_df[features].values    # Feature matrix for testing
y_test = test_df[target].values      # Target variable for testing

# Step 5: Create a Pipeline with SMOTE and CatBoostClassifier
# Adjust class weights to balance class prediction across classes
class_weights = [2, 1.5, 1.2]  # Adjust weights for class 0, 1, and 2
pipeline = Pipeline([
    ('smote', SMOTE(random_state=42)),  # Apply SMOTE to handle class imbalance
    ('classifier', CatBoostClassifier(verbose=0, random_state=42, class_weights=class_weights))  # CatBoost Classifier with class weights
])

# Step 6: Define the Parameter Grid for Hyperparameter Tuning
param_grid = {
    'classifier__iterations': [300, 500, 700],
    'classifier__depth': [6, 8, 10],
    'classifier__learning_rate': [0.005, 0.01, 0.05],
    'classifier__l2_leaf_reg': [5, 7, 10],  # Increasing L2 regularization to avoid overfitting
    'classifier__bagging_temperature': [1, 1.5, 2]  # Overfitting control
}

# Step 7: Use RandomizedSearchCV for Hyperparameter Tuning
random_search = RandomizedSearchCV(pipeline, param_distributions=param_grid, n_iter=30, cv=3, scoring='accuracy', n_jobs=-1, verbose=2, random_state=42)

# Step 8: Fit the Randomized Search to the Training Data
random_search.fit(X_train, y_train)

# Step 9: Get the Best Parameters from the Randomized Search
```

```
best_params = random_search.best_params_
print("Best parameters found: ", best_params)

# Step 10: Get the Best Model from RandomizedSearchCV
best_model = random_search.best_estimator_

# Step 11: Make Predictions on Both Training and Test Sets
y_train_pred = best_model.predict(X_train)
y_test_pred = best_model.predict(X_test)

# Step 12: Calculate and Print Training Accuracy
training_accuracy = accuracy_score(y_train, y_train_pred)
print(f"Training Accuracy: {training_accuracy:.4f}")

# Step 13: Calculate and Print Test Accuracy
test_accuracy = accuracy_score(y_test, y_test_pred)
print(f"Test Accuracy: {test_accuracy:.4f}")

# Step 14: Print the Classification Report for the Test Set
print("Classification Report:")
print(classification_report(y_test, y_test_pred))

# Step 15: Plot the Confusion Matrix for the Test Set
conf_matrix = confusion_matrix(y_test, y_test_pred)
plt.figure(figsize=(6, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=np.unique(y_test), yticklabels=np.unique(y_test))
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

Regression model to predict the defect amplitude using Multiple Linear Regression, Decision Trees, XGBoost, Random Forest and Cat Boost

```
#Multiple linear regression (80/20 split)


from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


# Step 1: Load the data
df = pd.read_csv('Connect.csv')


# Step 2: Extract features and target
X = df[['Line segment number', 'Track standard number', 'Milepost', 'Total
car east', 'Total car west',
        'Total train east', 'Total train west', 'Total deflection',
'Class', 'Freight speed',
        'Passenger speed', 'Defect type']].values


# Step 3:target variable 'Defect amplitude'
y = df['Defect amplitude'].values


# Step 4: Split the dataset 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 5: Initialize Linear Regression model
linear_model = LinearRegression()
```

```
# Step 6: Train the model on the training data
linear_model.fit(X_train, y_train)

# Step 7: Predict on the test data
y_test_pred_linear = linear_model.predict(X_test)

# Step 8: Predict on the training data to calculate training accuracy
y_train_pred_linear = linear_model.predict(X_train)

# Step 9: Calculate and print the R-squared score for the test and
training sets
r2_test_linear = r2_score(y_test, y_test_pred_linear)
r2_train_linear = r2_score(y_train, y_train_pred_linear)

print(f"R-squared (Test) for Multiple Linear Regression:
{r2_test_linear}")
print(f"R-squared (Training) for Multiple Linear Regression:
{r2_train_linear}")

# Step 10: Print the coefficients and intercept of the model
print("Coefficients of the model: ", linear_model.coef_)
print("Intercept of the model: ", linear_model.intercept_)

# Step 11: Calculate RMSE for both training and test sets
rmse_train_linear = np.sqrt(mean_squared_error(y_train,
y_train_pred_linear))
rmse_test_linear = np.sqrt(mean_squared_error(y_test, y_test_pred_linear))

print(f"RMSE (Training) for Linear Regression: {rmse_train_linear}")
print(f"RMSE (Test) for Linear Regression: {rmse_test_linear}")

# Step 12: Plot Actual vs Predicted for Training set
```

```
plt.figure(figsize=(8, 6))
plt.scatter(y_train, y_train_pred_linear, alpha=0.6, color='blue',
label='Predicted')
plt.plot([min(y_train), max(y_train)], [min(y_train), max(y_train)],
color='red', lw=2, label='Perfect Fit')  # Perfect fit line
plt.title('Actual vs Predicted (Training Set)')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()

# Step 13: Plot Actual vs Predicted for Test set
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_test_pred_linear, alpha=0.6, color='green',
label='Predicted')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color='red', lw=2, label='Perfect Fit')  # Perfect fit line
plt.title('Actual vs Predicted (Test Set)')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()

#decision tree to predict the defect amplitude (80/20 split)

from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import r2_score, mean_squared_error
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
# Step 1: Load the data
df = pd.read_csv('Connect.csv')


# Step 2: Extract features and target
X = df[['Line segment number', 'Track standard number', 'Milepost', 'Total
car east', 'Total car west',
        'Total train east', 'Total train west', 'Total deflection',
'Class', 'Freight speed',
        'Passenger speed', 'Defect type']].values


# Step 3: target variable 'Defect amplitude'
y = df['Defect amplitude'].values


# Step 4: Split the dataset 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)


# Refined parameter grid to reduce overfitting
param_grid = {
    'max_depth': [10, 15, 20, None],  # Limiting depth to avoid
overfitting
    'min_samples_split': [5, 10, 15],  # Requiring more samples to split a
node
    'min_samples_leaf': [5, 10, 15],  # Requiring more samples at each
leaf node
    'max_features': [None, 'sqrt', 'log2']  # Limiting the number of
features for each split
}


# Step 6: Initialize Decision Tree Regressor
tree = DecisionTreeRegressor(random_state=42)


# Step 7: Perform hyperparameter tuning with GridSearchCV
```

```
tree_search = GridSearchCV(tree, param_grid, cv=5, scoring='r2',
verbose=1, n_jobs=-1)
tree_search.fit(X_train, y_train)

# Step 8: Output the best parameters and the cross-validation R² score
best_tree_params = tree_search.best_params_
best_tree_score = tree_search.best_score_
print(f"Best parameters for Decision Tree: {best_tree_params}")
print(f"Best cross-validation R² score for Decision Tree:
{best_tree_score}")

# Step 9: Train the model with the best parameters
best_tree_model = tree_search.best_estimator_

# Step 10: Predict on the test data
y_test_pred_tree = best_tree_model.predict(X_test)

# Step 11: Predict on the training data to calculate training accuracy
y_train_pred_tree = best_tree_model.predict(X_train)

# Step 12: Calculate and print the R-squared score for the test and
training sets
r2_test_tree = r2_score(y_test, y_test_pred_tree)
r2_train_tree = r2_score(y_train, y_train_pred_tree)

print(f"R-squared (Test) for Decision Tree: {r2_test_tree}")
print(f"R-squared (Training) for Decision Tree: {r2_train_tree}")

# Step 13: Calculate RMSE for both training and test sets
rmse_train_tree = np.sqrt(mean_squared_error(y_train, y_train_pred_tree))
rmse_test_tree = np.sqrt(mean_squared_error(y_test, y_test_pred_tree))
```

```
print(f"RMSE (Training) for Decision Tree: {rmse_train_tree}")
print(f"RMSE (Test) for Decision Tree: {rmse_test_tree}")


# Step 14: Plot Actual vs Predicted for Training set
plt.figure(figsize=(8, 6))
plt.scatter(y_train, y_train_pred_tree, alpha=0.6, color='blue',
label='Predicted')
plt.plot([min(y_train), max(y_train)], [min(y_train), max(y_train)],
color='red', lw=2, label='Perfect Fit')  # Perfect fit line
plt.title('Actual vs Predicted (Training Set)')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()


# Step 15: Plot Actual vs Predicted for Test set
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_test_pred_tree, alpha=0.6, color='green',
label='Predicted')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color='red', lw=2, label='Perfect Fit')  # Perfect fit line
plt.title('Actual vs Predicted (Test Set)')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()


# XGBoost (80/20 split)
from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split, RandomizedSearchCV,
KFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
```

```
from sklearn.metrics import r2_score, mean_squared_error
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Load the data
df = pd.read_csv('Connect.csv')

# Step 2: Exploratory Data Analysis (EDA)
print("Dataset Statistics:\n", df.describe())
print("Missing Values:\n", df.isnull().sum())

# Plot the distribution of the target variable 'Defect amplitude'
plt.figure(figsize=(8, 6))
sns.histplot(df['Defect amplitude'], kde=True, bins=30)
plt.title('Distribution of Defect Amplitude')
plt.xlabel('Defect Amplitude')
plt.ylabel('Frequency')
plt.show()

# Step 3: Correlation Matrix
features = ['Line segment number','Track standard number','Milepost',
'Total car east', 'Total car west',
        'Total train east', 'Total train west', 'Total deflection',
'Class', 'Freight speed',
        'Passenger speed','Defect type']

plt.figure(figsize=(10, 8))
corr_matrix = df[features].corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Feature Correlation Matrix")
```

```
plt.show()


# Step 4: Extract features and target
X = df[['Milepost', 'Total car east', 'Total car west',
        'Total train east', 'Total train west', 'Total deflection',
'Class', 'Freight speed',
        'Passenger speed','Defect type']].values
y = df['Defect amplitude'].values


# Step 5: 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 6: Define a pipeline that first scales the data then applies
XGBoost Regressor
pipeline = Pipeline([
    ('scaler', StandardScaler()),                # Standardize the
features
    ('xgb', XGBRegressor(objective='reg:squarederror',
random_state=42))  # XGBoost model
])


# Step 7: Set up a parameter distribution with higher regularization
param_distributions = {
    'xgb__n_estimators': [1000, 1500],                  # Increase number
of boosting rounds for fine-tuning
    'xgb__max_depth': [5, 7],                    # Limit tree depth to
avoid overfitting
    'xgb__learning_rate': [0.01, 0.02],          # Lower learning rates
for smaller updates
    'xgb__subsample': [0.7],                     # Fixed subsample ratio
to reduce variance
    'xgb__colsample_bytree': [0.7],              # Fixed feature sampling
to reduce overfitting
```

```
    'xgb__min_child_weight': [3, 5],              # Increase to avoid
overfitting small splits
    'xgb__gamma': [0.1, 0.2],                     # Minimum loss reduction
to make a split
    'xgb__reg_alpha': [0.5, 1],                   # Higher L1
regularization
    'xgb__reg_lambda': [2, 3]                     # Higher L2
regularization
}

# Step 8: Use K-Fold for cross-validation
kf = KFold(n_splits=5)

# Step 9: Set up RandomizedSearchCV with the updated parameter space
random_search = RandomizedSearchCV(estimator=pipeline,
param_distributions=param_distributions,
                                   n_iter=10, cv=kf, scoring='r2',
n_jobs=-1, verbose=2, random_state=42)

# Step 10: Fit the random search to the data
random_search.fit(X_train, y_train)

# Step 11: Best hyperparameters
best_params = random_search.best_params_
print("Best Hyperparameters: ", best_params)

# Step 12: Train the model with the best hyperparameters
best_model = random_search.best_estimator_

# Step 13: Predict on the train and test sets
y_train_pred = best_model.predict(X_train)
y_test_pred = best_model.predict(X_test)
```

```
# Step 14: Calculate R^2 and RMSE scores
r2_train = r2_score(y_train, y_train_pred)
r2_test = r2_score(y_test, y_test_pred)

rmse_train = np.sqrt(mean_squared_error(y_train, y_train_pred))
rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))

# Step 15: Output the R^2 and RMSE scores
print("Training R^2 Score: ", r2_train)
print("Testing R^2 Score: ", r2_test)
print("Training RMSE: ", rmse_train)
print("Testing RMSE: ", rmse_test)

# Step 16: Predict on the entire dataset
y_pred = best_model.predict(X)

# Step 17: Add the predictions as a new column to the original DataFrame
df['Prediction'] = y_pred

# Step 18: Save the updated DataFrame to a new CSV file
df.to_csv('Connect_with_predictions.csv', index=False)

# Step 19: Output location
print("Results saved to 'Connect_with_predictions.csv'")

# Step 20: Plot the feature importance
xgb_model = best_model.named_steps['xgb']  # Extract the XGB model from
the pipeline
feature_importances = xgb_model.feature_importances_

# Step 21: Define feature names
```

```
feature_names = ['Milepost', 'Total car east', 'Total car west', 'Total
train east',
                 'Total train west', 'Total deflection', 'Class', 'Freight
speed',
                 'Passenger speed', 'Defect type']


# Step 22: Create a DataFrame for plotting
importances_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': feature_importances
}).sort_values(by='Importance', ascending=False)


# Step 23: Plot the feature importances
plt.figure(figsize=(10, 6))
plt.barh(importances_df['Feature'], importances_df['Importance'])
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Feature Importances from XGBoost Model')
plt.gca().invert_yaxis()  # Invert y-axis to display the most important
feature at the top
plt.show()


# Step 24: Plot Predicted vs Observed (Training set)
plt.figure(figsize=(8, 6))
plt.scatter(y_train, y_train_pred, alpha=0.6, color='b')
plt.plot([min(y_train), max(y_train)], [min(y_train), max(y_train)],
color='red', lw=2)  # Line of perfect fit
plt.title('Predicted vs Observed (Training Set)')
plt.xlabel('Observed')
plt.ylabel('Predicted')
plt.show()
```

```
# Step 25: Plot Predicted vs Observed (Test set)
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_test_pred, alpha=0.6, color='b')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color='red', lw=2)  # Line of perfect fit
plt.title('Predicted vs Observed (Test Set)')
plt.xlabel('Observed')
plt.ylabel('Predicted')
plt.show()

#Random Forest(80/20 split)

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split, RandomizedSearchCV,
KFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import r2_score, mean_squared_error

# Step 1: Load the data
df = pd.read_csv('Connect.csv')

# Step 2: Exploratory Data Analysis (EDA)
# Check basic statistics of the dataset
print("Dataset Statistics:\n", df.describe())

# Check for missing values
print("Missing Values:\n", df.isnull().sum())
```

```
# Plot the distribution of the target variable 'Defect amplitude'
plt.figure(figsize=(8, 6))
sns.histplot(df['Defect amplitude'], kde=True, bins=30)
plt.title('Distribution of Defect Amplitude')
plt.xlabel('Defect Amplitude')
plt.ylabel('Frequency')
plt.show()

# Step 3: Correlation Matrix
# Calculate correlation between numerical features

features = ['Milepost', 'Total car east', 'Total car west',
        'Total train east', 'Total train west', 'Total deflection',
'Class', 'Freight speed',
        'Passenger speed','Defect type']

plt.figure(figsize=(10, 8))
corr_matrix = df[features].corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Feature Correlation Matrix")
plt.show()

# Step 4: Extract features and target
X = df[['Line segment number', 'Milepost', 'Track standard number',
        'Total car east', 'Total car west',
        'Total train east', 'Total train west', 'Total deflection',
'Class', 'Freight speed',
        'Passenger speed','Defect type']].values
y = df['Defect amplitude'].values

# Step 5: 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 6: Define a pipeline that first scales the data then applies
RandomForestRegressor
pipeline = Pipeline([
    ('scaler', StandardScaler()),            # Standardize the features
    ('rf', RandomForestRegressor(random_state=42))  # RandomForest model
])


# Step 7: Set up a more aggressive parameter distribution to sample from
for regularization
param_distributions = {
    'rf__n_estimators': [600, 800, 1000],            # Further increase
the number of trees
    'rf__max_depth': [15, 20, 25],                  # Experiment with
different tree depths
    'rf__min_samples_split': [10, 20, 30],          # Require more samples
to split
    'rf__min_samples_leaf': [2, 4, 6],              # Allow smaller leaves
    'rf__max_features': ['sqrt', 0.2, 0.3]          # Experiment with
different feature subsets
}


# Step 8: Use K-Fold for cross-validation
kf = KFold(n_splits=5)


# Step 9: Set up RandomizedSearchCV with the pipeline and K-Fold
random_search = RandomizedSearchCV(estimator=pipeline,
param_distributions=param_distributions,
                                   n_iter=50, cv=kf, scoring='r2',
n_jobs=-1, verbose=2, random_state=42)


# Step 10: Fit the random search to the data
```

```
random_search.fit(X_train, y_train)

# Step 11: Best hyperparameters
best_params = random_search.best_params_
print("Best Hyperparameters: ", best_params)

# Step 12: Train the model with the best hyperparameters
best_model = random_search.best_estimator_

# Step 13: Predict on the train and test sets
y_train_pred = best_model.predict(X_train)
y_test_pred = best_model.predict(X_test)

# Step 14: Calculate R^2 and RMSE scores
r2_train = r2_score(y_train, y_train_pred)
r2_test = r2_score(y_test, y_test_pred)

# RMSE for training and test sets
rmse_train = np.sqrt(mean_squared_error(y_train, y_train_pred))
rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))

# Step 15: Output the R^2 and RMSE scores
print("Training R^2 Score: ", r2_train)
print("Testing R^2 Score: ", r2_test)
print("Training RMSE: ", rmse_train)
print("Testing RMSE: ", rmse_test)

# Step 16: Predict on the entire dataset
y_pred = best_model.predict(X)

# Step 17: Add the predictions as a new column to the original DataFrame
```

```
df['Prediction'] = y_pred


# Step 18: Save the updated DataFrame to a new CSV file
df.to_csv('Connect_with_predictions.csv', index=False)


# Step 19: Output location
print("Results saved to 'Connect_with_predictions.csv'")


# Step 20: Plot feature importance
rf_model = best_model.named_steps['rf']  # Extract the Random Forest model from the pipeline
feature_importances = rf_model.feature_importances_


# Step 21: Define feature names
feature_names = ['Line segment number', 'Milepost', 'Track standard number', 'Total car east',
                 'Total car west', 'Total train east', 'Total train west', 'Total deflection', 'Class', 'Freight speed', 'Passenger speed', 'Defect type']


# Step 22: Create a DataFrame for plotting feature importances
importances_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': feature_importances
}).sort_values(by='Importance', ascending=False)


# Step 23: Plot the feature importances
plt.figure(figsize=(10, 6))
plt.barh(importances_df['Feature'], importances_df['Importance'])
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Feature Importances from Random Forest Model')
```

```
plt.gca().invert_yaxis()  # Invert y-axis to display the most important
feature at the top
plt.show()


# Step 24: Plot Predicted vs Observed (Training set)
plt.figure(figsize=(8, 6))
plt.scatter(y_train, y_train_pred, alpha=0.6, color='b')
plt.plot([min(y_train), max(y_train)], [min(y_train), max(y_train)],
color='red', lw=2)  # Line of perfect fit
plt.title('Predicted vs Observed (Training Set)')
plt.xlabel('Observed')
plt.ylabel('Predicted')
plt.show()


# Step 25: Plot Predicted vs Observed (Test set)
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_test_pred, alpha=0.6, color='b')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color='red', lw=2)  # Line of perfect fit
plt.title('Predicted vs Observed (Test Set)')
plt.xlabel('Observed')
plt.ylabel('Predicted')
plt.show()


#catboost (80/20% split)


from sklearn.model_selection import RandomizedSearchCV
from catboost import CatBoostRegressor
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
```

```
import matplotlib.pyplot as plt


# Step 1: Load the data
df = pd.read_csv('Connect.csv')


# Step 2: Extract features and target
X = df[['Line segment number','Track standard number','Milepost', 'Total
car east', 'Total car west',
        'Total train east', 'Total train west', 'Total deflection',
'Class', 'Freight speed',
        'Passenger speed', 'Defect type']]


# Convert the target variable 'Defect amplitude' to its absolute value
y = df['Defect amplitude'].values


# Step 3: Split the dataset 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 4: Define a parameter grid for tuning
param_grid = {
    'iterations': [250, 300, 350],
    'depth': [10, 12, 14],
    'learning_rate': [0.05, 0.08, 0.1],
    'l2_leaf_reg': [3, 4, 5],
    'border_count': [160, 180],
    'bagging_temperature': [1.0, 1.2, 1.5]
}


# Step 5: Initialize CatBoost Regressor
catboost_reg = CatBoostRegressor(verbose=0, random_state=42)
```

```
# Step 6: Initialize RandomizedSearchCV for CatBoost tuning
random_search_catboost = RandomizedSearchCV(
    catboost_reg,
    param_distributions=param_grid,
    n_iter=20,
    scoring='r2',
    cv=5,
    verbose=1,
    n_jobs=-1,
    random_state=42
)

# Step 7: Fit RandomizedSearchCV
random_search_catboost.fit(X_train, y_train)

# Step 8: Output the best parameters and cross-validation R² score
best_params = random_search_catboost.best_params_
best_score = random_search_catboost.best_score_

print(f"Best parameters for tuned CatBoost: {best_params}")
print(f"Best cross-validation R² score: {best_score}")

# Step 9: Train the model with the best parameters
best_model_catboost = random_search_catboost.best_estimator_

# Step 10: Predict on the test data
y_test_pred = best_model_catboost.predict(X_test)

# Step 11: Predict on the training data
y_train_pred = best_model_catboost.predict(X_train)
```

```
# Step 12: Calculate R-squared and RMSE for training and test sets
r2_train = r2_score(y_train, y_train_pred)
r2_test = r2_score(y_test, y_test_pred)

rmse_train = np.sqrt(mean_squared_error(y_train, y_train_pred))
rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))

# Step 13: Output R-squared and RMSE scores
print(f"R-squared (Training): {r2_train}")
print(f"R-squared (Test): {r2_test}")
print(f"RMSE (Training): {rmse_train}")
print(f"RMSE (Test): {rmse_test}")

# Step 14: Plot Actual vs Predicted for Training set
plt.figure(figsize=(8, 6))
plt.scatter(y_train, y_train_pred, alpha=0.6, color='b',
label='Predicted')
plt.plot([min(y_train), max(y_train)], [min(y_train), max(y_train)],
color='red', lw=2, label='Perfect Fit')  # Perfect fit line
plt.title('Actual vs Predicted (Training Set)')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()

# Step 15: Plot Actual vs Predicted for Test set
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_test_pred, alpha=0.6, color='b', label='Predicted')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color='red', lw=2, label='Perfect Fit')  # Perfect fit line
plt.title('Actual vs Predicted (Test Set)')
plt.xlabel('Actual Values')
```

```
plt.ylabel('Predicted Values')
plt.legend()
plt.show()

#XGBoost (2007 - 2012 training data, 2013 Test data)
from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split, RandomizedSearchCV,
KFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import r2_score, mean_squared_error
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Load the training and test datasets
train_df = pd.read_csv('Connecttrain.csv')
test_df = pd.read_csv('Connecttest.csv')

# Step 2: Exploratory Data Analysis (EDA)
# Check basic statistics of the training dataset
print("Training Data Statistics:\n", train_df.describe())

# Check for missing values in training and test datasets
print("Missing Values in Training Data:\n", train_df.isnull().sum())
print("Missing Values in Test Data:\n", test_df.isnull().sum())

# Plot the distribution of the target variable 'Defect amplitude' in
training data
plt.figure(figsize=(8, 6))
sns.histplot(train_df['Defect amplitude'], kde=True, bins=30)
```

```
plt.title('Distribution of Defect Amplitude in Training Data')
plt.xlabel('Defect Amplitude')
plt.ylabel('Frequency')
plt.show()

# Step 3: Define the feature and target columns
features = ['Milepost', 'Total car east', 'Total car west', 'Total train east',
            'Total train west', 'Total deflection', 'Class', 'Freight speed',
            'Passenger speed', 'Defect type']
target = 'Defect amplitude'

# Step 4: Extract features and target
X_train = train_df[features].values
y_train = train_df[target].values
X_test = test_df[features].values
y_test = test_df[target].values

# Step 5: Define a pipeline that first scales the data then applies XGBoost Regressor
pipeline = Pipeline([
    ('scaler', StandardScaler()),                # Standardize the features
    ('xgb', XGBRegressor(objective='reg:squarederror', random_state=42))  # XGBoost model
])

# Step 6: Set up a parameter distribution with increased regularization and early stopping
param_distributions = {
    'xgb__n_estimators': [3000],                 # Increase number of boosting rounds for fine-tuning
```

```
    'xgb__max_depth': [4, 6],                        # Further limit tree
depth
    'xgb__learning_rate': [0.005, 0.01],             # Lower learning rates
for smaller updates
    'xgb__subsample': [0.7],                         # Fixed subsample ratio
to reduce variance
    'xgb__colsample_bytree': [0.7],                  # Fixed feature sampling
to reduce overfitting
    'xgb__min_child_weight': [6, 8],                 # Increase to avoid
overfitting small splits
    'xgb__gamma': [0.1, 0.2],                        # Minimum loss reduction
to make a split
    'xgb__reg_alpha': [1, 2],                        # Higher L1
regularization
    'xgb__reg_lambda': [4, 5]                        # Higher L2
regularization
}


# Step 7: Use K-Fold for cross-validation
kf = KFold(n_splits=5)


# Step 8: Set up RandomizedSearchCV with the updated parameter space
random_search = RandomizedSearchCV(estimator=pipeline,
param_distributions=param_distributions,
                                   n_iter=10, cv=kf, scoring='r2',
n_jobs=-1, verbose=2, random_state=42)


# Step 9: Fit the random search to the data
random_search.fit(X_train, y_train)


# Step 10: Best hyperparameters
best_params = random_search.best_params_
print("Best Hyperparameters: ", best_params)
```

```
# Step 11: Extract the best model and set up early stopping
best_xgb_model = random_search.best_estimator_.named_steps['xgb']
best_xgb_model.set_params(early_stopping_rounds=100)

# Step 12: Train the model with the best hyperparameters and early
stopping
best_xgb_model.fit(X_train, y_train, eval_set=[(X_test, y_test)],
verbose=False)

# Step 13: Predict on the train and test sets
y_train_pred = best_xgb_model.predict(X_train)
y_test_pred = best_xgb_model.predict(X_test)

# Step 14: Calculate R^2 scores and RMSE for training and test sets
r2_train = r2_score(y_train, y_train_pred)
r2_test = r2_score(y_test, y_test_pred)

rmse_train = np.sqrt(mean_squared_error(y_train, y_train_pred))
rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))

# Step 15: Output the R^2 and RMSE scores
print("Training R^2 Score: ", r2_train)
print("Testing R^2 Score: ", r2_test)
print("Training RMSE: ", rmse_train)
print("Testing RMSE: ", rmse_test)

# Step 16: Plot Actual vs Predicted for Training set
plt.figure(figsize=(8, 6))
plt.scatter(y_train, y_train_pred, alpha=0.6, color='b',
label='Predicted')
plt.plot([min(y_train), max(y_train)], [min(y_train), max(y_train)],
color='red', lw=2, label='Perfect Fit')  # Perfect fit line
```

```
plt.title('Actual vs Predicted (Training Set)')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()

# Step 17: Plot Actual vs Predicted for Test set
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_test_pred, alpha=0.6, color='b', label='Predicted')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color='red', lw=2, label='Perfect Fit')  # Perfect fit line
plt.title('Actual vs Predicted (Test Set)')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()

#Random Forest (2007 - 2012 training data, 2013 Test data)

from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV, KFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import r2_score, mean_squared_error
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Define the feature and target columns
features = ['Line segment number', 'Milepost', 'Track standard number',
```

```
            'Total car east', 'Total car west', 'Total train east',
            'Total train west', 'Total deflection', 'Track code',
            'Class', 'Freight speed', 'Passenger speed', 'Defect type']
target = 'Defect amplitude'

# Step 2: Load the train and test data
train_df = pd.read_csv('Connecttrain.csv')
test_df = pd.read_csv('Connecttest.csv')

# Step 3: Exploratory Data Analysis (EDA)
# Check basic statistics of the training dataset
print("Training Data Statistics:\n", train_df.describe())

# Check for missing values in training and test datasets
print("Missing Values in Training Data:\n", train_df.isnull().sum())
print("Missing Values in Test Data:\n", test_df.isnull().sum())

# Plot the distribution of the target variable 'Defect amplitude' in the
training data
plt.figure(figsize=(8, 6))
sns.histplot(train_df['Defect amplitude'], kde=True, bins=30)
plt.title('Distribution of Defect Amplitude in Training Data')
plt.xlabel('Defect Amplitude')
plt.ylabel('Frequency')
plt.show()

# Step 5: Extract features and target
X_train = train_df[features].values
y_train = train_df[target].values
X_test = test_df[features].values
y_test = test_df[target].values
```

```
# Step 6: Define a pipeline that first scales the data then applies
RandomForestRegressor
pipeline = Pipeline([
    ('scaler', StandardScaler()),            # Standardize the features
    ('rf', RandomForestRegressor(random_state=42))  # RandomForest model
])

# Step 7: Set up a more aggressive parameter distribution to sample from
for regularization
param_distributions = {
    'rf__n_estimators': [700, 800, 1000],          # Increase the number
of trees
    'rf__max_depth': [8, 12, 16],                  # Limit the tree depth
more aggressively
    'rf__min_samples_split': [20, 25, 30],         # Require more samples
to split
    'rf__min_samples_leaf': [5, 7, 9],             # Require more samples
at leaf nodes
    'rf__max_features': [0.1, 0.2, 'sqrt']         # Limit features for
each split
}

# Step 8: Use K-Fold for cross-validation
kf = KFold(n_splits=5)

# Step 9: Set up RandomizedSearchCV with the pipeline and K-Fold
random_search = RandomizedSearchCV(estimator=pipeline,
param_distributions=param_distributions,
                                   n_iter=30, cv=kf, scoring='r2',
n_jobs=-1, verbose=2, random_state=42)

# Step 10: Fit the random search to the data
random_search.fit(X_train, y_train)
```

```
# Step 11: Best hyperparameters
best_params = random_search.best_params_
print("Best Hyperparameters: ", best_params)

# Step 12: Train the model with the best hyperparameters
best_model = random_search.best_estimator_

# Step 13: Predict on the train and test sets
y_train_pred = best_model.predict(X_train)
y_test_pred = best_model.predict(X_test)

# Step 14: Calculate R^2 scores and RMSE for both training and test sets
r2_train = r2_score(y_train, y_train_pred)
r2_test = r2_score(y_test, y_test_pred)

rmse_train = np.sqrt(mean_squared_error(y_train, y_train_pred))
rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))

# Step 15: Output the R^2 and RMSE scores
print("Training R^2 Score: ", r2_train)
print("Testing R^2 Score: ", r2_test)
print("Training RMSE: ", rmse_train)
print("Testing RMSE: ", rmse_test)

# Step 16: Plot Actual vs Predicted for Training set
plt.figure(figsize=(8, 6))
plt.scatter(y_train, y_train_pred, alpha=0.6, color='b',
label='Predicted')
plt.plot([min(y_train), max(y_train)], [min(y_train), max(y_train)],
color='red', lw=2, label='Perfect Fit')  # Perfect fit line
plt.title('Actual vs Predicted (Training Set)')
```

```
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()

# Step 17: Plot Actual vs Predicted for Test set
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_test_pred, alpha=0.6, color='b', label='Predicted')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color='red', lw=2, label='Perfect Fit')  # Perfect fit line
plt.title('Actual vs Predicted (Test Set)')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()

#cat boost  (2007 - 2012 training data, 2013 Test data)

from catboost import CatBoostRegressor
from sklearn.model_selection import train_test_split, RandomizedSearchCV,
KFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import r2_score, mean_squared_error
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Load the training and test datasets
train_df = pd.read_csv('Connecttrain.csv')
test_df = pd.read_csv('Connecttest.csv')
```

```
# Step 2: Exploratory Data Analysis (EDA)
# Check basic statistics of the training dataset
print("Training Data Statistics:\n", train_df.describe())

# Check for missing values in training and test datasets
print("Missing Values in Training Data:\n", train_df.isnull().sum())
print("Missing Values in Test Data:\n", test_df.isnull().sum())

# Plot the distribution of the target variable 'Defect amplitude' in
training data
plt.figure(figsize=(8, 6))
sns.histplot(train_df['Defect amplitude'], kde=True, bins=30)
plt.title('Distribution of Defect Amplitude in Training Data')
plt.xlabel('Defect Amplitude')
plt.ylabel('Frequency')
plt.show()

# Step 3: Define the feature and target columns
features = ['Milepost', 'Total car east', 'Total car west', 'Total train
east',
            'Total train west', 'Total deflection', 'Class', 'Freight
speed',
            'Passenger speed', 'Defect type']
target = 'Defect amplitude'

# Step 4: Extract features and target
X_train = train_df[features].values
y_train = train_df[target].values
X_test = test_df[features].values
y_test = test_df[target].values
```

```
# Step 5: Define a pipeline that first scales the data then applies CatBoost Regressor
pipeline = Pipeline([
    ('scaler', StandardScaler()),  # Standardize the features
    ('catboost', CatBoostRegressor(verbose=0, random_state=42))  # CatBoost model
])

# Step 6: Set up a refined parameter distribution with lower learning rate and more iterations
param_distributions = {
    'catboost__iterations': [1500, 2000],          # Higher number of boosting rounds
    'catboost__depth': [6, 7],                     # Keep depth at moderate levels
    'catboost__learning_rate': [0.01, 0.02],       # Lower learning rates for gradual updates
    'catboost__subsample': [0.7],                  # Subsampling to reduce variance
    'catboost__l2_leaf_reg': [5, 7],               # L2 regularization for generalization
    'catboost__bagging_temperature': [2, 3]        # Add randomness through bagging
}

# Step 7: Use K-Fold for cross-validation
kf = KFold(n_splits=5)

# Step 8: Set up RandomizedSearchCV with the parameter space
random_search = RandomizedSearchCV(estimator=pipeline, param_distributions=param_distributions,
                                   n_iter=10, cv=kf, scoring='r2', n_jobs=-1, verbose=2, random_state=42)
```

```
# Step 9: Fit the random search to the data
random_search.fit(X_train, y_train)

# Step 10: Best hyperparameters
best_params = random_search.best_params_
print("Best Hyperparameters: ", best_params)

# Step 11: Extract the best model from the pipeline
best_catboost_model =
random_search.best_estimator_.named_steps['catboost']

# Step 12: Train the model with early stopping using validation data
best_catboost_model.fit(X_train, y_train, eval_set=(X_test, y_test),
early_stopping_rounds=100, verbose=False)

# Step 13: Predict on the train and test sets
y_train_pred = best_catboost_model.predict(X_train)
y_test_pred = best_catboost_model.predict(X_test)

# Step 14: Calculate R^2 scores and RMSE for both training and test sets
r2_train = r2_score(y_train, y_train_pred)
r2_test = r2_score(y_test, y_test_pred)

rmse_train = np.sqrt(mean_squared_error(y_train, y_train_pred))
rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))

# Step 15: Output the R^2 and RMSE scores
print("Training R^2 Score: ", r2_train)
print("Testing R^2 Score: ", r2_test)
print("Training RMSE: ", rmse_train)
print("Testing RMSE: ", rmse_test)
```

```
# Step 16: Plot Actual vs Predicted for Training set
plt.figure(figsize=(8, 6))
plt.scatter(y_train, y_train_pred, alpha=0.6, color='b',
label='Predicted')
plt.plot([min(y_train), max(y_train)], [min(y_train), max(y_train)],
color='red', lw=2, label='Perfect Fit')  # Perfect fit line
plt.title('Actual vs Predicted (Training Set)')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()

# Step 17: Plot Actual vs Predicted for Test set
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_test_pred, alpha=0.6, color='b', label='Predicted')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color='red', lw=2, label='Perfect Fit')  # Perfect fit line
plt.title('Actual vs Predicted (Test Set)')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()
```

Regression Model to predict the defect length using Multiple Linear Regression, Decision Trees, XGBoost, Random Forest and Cat Boost

```
# Multiple linear regression to predict the defect length (80/20 Split)

import pandas as pd
import numpy as np
```

```
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Step 1: Load the dataset
df = pd.read_csv('Connect.csv')

# Step 2: Exploratory Data Analysis (EDA)
# Check basic statistics of the dataset
print("Dataset Statistics:\n", df.describe())

# Check for missing values
print("Missing Values:\n", df.isnull().sum())

# Step 3: Define features and target
features = ['Line segment number', 'Track standard number', 'Milepost',
'Total car east', 'Total car west',
            'Total train east', 'Total train west', 'Defect amplitude',
'Total deflection', 'Class',
            'Freight speed', 'Passenger speed', 'Defect type']
target = 'Defect length'

# Step 4: Prepare the data
X = df[features].values  # Feature matrix
y = df[target].values    # Target variable

# Step 5: 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 6: Initialize the Linear Regression model
```

```
model = LinearRegression()

# Step 7: Train the model on the training data
model.fit(X_train, y_train)

# Step 8: Predict on both training and test sets
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

# Step 9: Evaluate the model
# Calculate R² score and RMSE for both training and test sets
train_r2 = r2_score(y_train, y_train_pred)
test_r2 = r2_score(y_test, y_test_pred)

train_rmse = np.sqrt(mean_squared_error(y_train, y_train_pred))
test_rmse = np.sqrt(mean_squared_error(y_test, y_test_pred))

# Step 10: Output the results
print(f'R² Score on Training Set: {train_r2}')
print(f'R² Score on Test Set: {test_r2}')
print(f'RMSE on Training Set: {train_rmse}')
print(f'RMSE on Test Set: {test_rmse}')

# Step 11: Plot Actual vs Predicted for Training set
plt.figure(figsize=(8, 6))
plt.scatter(y_train, y_train_pred, alpha=0.6, color='blue',
label='Predicted')
plt.plot([min(y_train), max(y_train)], [min(y_train), max(y_train)],
color='red', lw=2, label='Perfect Fit')  # Perfect fit line
plt.title('Actual vs Predicted (Training Set)')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
```

```
plt.legend()
plt.show()

# Step 12: Plot Actual vs Predicted for Test set
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_test_pred, alpha=0.6, color='green',
label='Predicted')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color='red', lw=2, label='Perfect Fit')  # Perfect fit line
plt.title('Actual vs Predicted (Test Set)')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()

# Decision trees regression to predict the defect length(80% 20% split)

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import r2_score, mean_squared_error

# Step 1: Load the dataset
df = pd.read_csv('Connect.csv')

# Step 2: Exploratory Data Analysis (EDA)
# Check basic statistics of the dataset
print("Dataset Statistics:\n", df.describe())

# Check for missing values
```

```
print("Missing Values:\n", df.isnull().sum())


# Step 3: Define features and target
features = ['Line segment number', 'Track standard number', 'Milepost',
'Total car east', 'Total car west',
            'Total train east', 'Total train west', 'Defect amplitude',
'Total deflection', 'Class',
            'Freight speed', 'Passenger speed', 'Defect type']
target = 'Defect length'


# Step 4: Prepare data
X = df[features].values
y = df[target].values


# Step 5: Split the dataset 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 6: Define the parameter grid for hyperparameter tuning
param_grid = {
    'max_depth': [5, 10, 15, 20, None],  # Depth of the tree
    'min_samples_split': [2, 5, 10],  # Minimum samples required to split
a node
    'min_samples_leaf': [1, 2, 5, 10],  # Minimum samples required in a
leaf node
    'max_features': [None, 'sqrt', 'log2']  # Number of features to
consider for best split
}


# Step 7: Initialize Decision Tree Regressor
tree = DecisionTreeRegressor(random_state=42)


# Step 8: Perform hyperparameter tuning with GridSearchCV
```

```
tree_search = GridSearchCV(tree, param_grid, cv=5, scoring='r2',
verbose=1, n_jobs=-1)
tree_search.fit(X_train, y_train)

# Step 9: Output the best parameters and the cross-validation R² score
best_tree_params = tree_search.best_params_
best_tree_score = tree_search.best_score_
print(f"Best parameters for Decision Tree: {best_tree_params}")
print(f"Best cross-validation R² score for Decision Tree:
{best_tree_score}")

# Step 10: Train the model with the best parameters
best_tree_model = tree_search.best_estimator_

# Step 11: Predict on the test and training data
y_test_pred_tree = best_tree_model.predict(X_test)
y_train_pred_tree = best_tree_model.predict(X_train)

# Step 12: Calculate R² and RMSE scores for the training and test sets
r2_test_tree = r2_score(y_test, y_test_pred_tree)
r2_train_tree = r2_score(y_train, y_train_pred_tree)
rmse_test_tree = np.sqrt(mean_squared_error(y_test, y_test_pred_tree))
rmse_train_tree = np.sqrt(mean_squared_error(y_train, y_train_pred_tree))

print(f"R-squared (Test) for Decision Tree: {r2_test_tree}")
print(f"R-squared (Training) for Decision Tree: {r2_train_tree}")
print(f"RMSE (Test) for Decision Tree: {rmse_test_tree}")
print(f"RMSE (Training) for Decision Tree: {rmse_train_tree}")

# Step 13: Plot Actual vs Predicted for Training set
plt.figure(figsize=(8, 6))
```

```
plt.scatter(y_train, y_train_pred_tree, alpha=0.6, color='blue',
label='Predicted')
plt.plot([min(y_train), max(y_train)], [min(y_train), max(y_train)],
color='red', lw=2, label='Perfect Fit')  # Perfect fit line
plt.title('Actual vs Predicted (Training Set)')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()


# Step 14: Plot Actual vs Predicted for Test set
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_test_pred_tree, alpha=0.6, color='green',
label='Predicted')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color='red', lw=2, label='Perfect Fit')  # Perfect fit line
plt.title('Actual vs Predicted (Test Set)')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()


#XGBoost to predict defect length (80/20 Split)


import pandas as pd
import numpy as np
from sklearn.model_selection import KFold, train_test_split,
RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import r2_score, mean_squared_error
import xgboost as xgb
from scipy.stats import uniform, randint
```

```
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Load the dataset
df = pd.read_csv('Connect.csv')

# Step 2: Exploratory Data Analysis (EDA)
# Check basic statistics of the dataset
print("Dataset Statistics:\n", df.describe())

# Check for missing values
print("Missing Values:\n", df.isnull().sum())

# Plot the distribution of the target variable 'Defect length'
plt.figure(figsize=(8, 6))
sns.histplot(df['Defect length'], kde=True, bins=30)
plt.title('Distribution of Defect Length')
plt.xlabel('Defect Length')
plt.ylabel('Frequency')
plt.show()

# Step 3: Correlation Matrix
# Calculate correlation between numerical features
features = ['Line segment number', 'Track standard number', 'Milepost',
'Total car east', 'Total car west',
            'Total train east', 'Total train west', 'Defect amplitude',
'Total deflection', 'Class',
            'Freight speed', 'Passenger speed', 'Defect type']

plt.figure(figsize=(10, 8))
corr_matrix = df[features].corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
```

```
plt.title("Feature Correlation Matrix")
plt.show()

# Step 4: Define features and target
target = 'Defect length'

# Step 5: Prepare data
X = df[features].values
y = df[target].values

# Step 6: Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Step 7: Initialize the scaler and the model
scaler = StandardScaler()
model = xgb.XGBRegressor(objective='reg:squarederror', random_state=42)

# Step 8: Create a pipeline that first scales the data then fits the model
pipeline = Pipeline([
    ('scaler', scaler),
    ('regressor', model)
])

# Step 9: Set up cross-validation
kfold = KFold(n_splits=10, shuffle=True, random_state=42)

# Step 10: Define the parameter distributions to sample from
param_dist = {
    'regressor__n_estimators': randint(100, 300),
    'regressor__max_depth': randint(3, 10),
```

```
    'regressor__learning_rate': uniform(0.01, 0.2),
    'regressor__subsample': uniform(0.6, 0.4),
    'regressor__colsample_bytree': uniform(0.6, 0.4),
    'regressor__min_child_weight': randint(1, 10),
    'regressor__gamma': uniform(0, 0.5),
    'regressor__reg_alpha': uniform(0, 1),  # L1 regularization term
    'regressor__reg_lambda': uniform(0, 1)  # L2 regularization term
}

# Step 11: Set up RandomizedSearchCV
random_search = RandomizedSearchCV(pipeline, param_dist, n_iter=50,
cv=kfold, scoring='r2', n_jobs=-1, random_state=42)

# Step 12: Fit the RandomizedSearchCV to find the best model
random_search.fit(X_train, y_train)

# Step 13: Get the best model from the random search
best_model = random_search.best_estimator_

# Step 14: Predict on the training and test sets using the best model
y_train_pred = best_model.predict(X_train)
y_test_pred = best_model.predict(X_test)

# Step 15: Calculate R² scores and RMSE for both training and test sets
r2_train = r2_score(y_train, y_train_pred)
r2_test = r2_score(y_test, y_test_pred)

rmse_train = np.sqrt(mean_squared_error(y_train, y_train_pred))
rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))

# Step 16: Output the results
```

```
print(f'Best Parameters: {random_search.best_params_}')
print(f'R² Score on Training Set: {r2_train}')
print(f'R² Score on Test Set: {r2_test}')
print(f'RMSE on Training Set: {rmse_train}')
print(f'RMSE on Test Set: {rmse_test}')

# Step 17: Plot feature importance
xgb_model = best_model.named_steps['regressor']  # Extract the XGBoost model from the pipeline
feature_importances = xgb_model.feature_importances_

# Step 18: Define feature names
feature_names = features

# Step 19: Create a DataFrame for plotting
importances_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': feature_importances
}).sort_values(by='Importance', ascending=False)

# Step 20: Plot the feature importances
plt.figure(figsize=(10, 6))
plt.barh(importances_df['Feature'], importances_df['Importance'])
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Feature Importances from XGBoost Model')
plt.gca().invert_yaxis()  # Invert y-axis to display the most important feature at the top
plt.show()

# Step 21: Plot Actual vs Predicted for Training set
plt.figure(figsize=(8, 6))
```

```
plt.scatter(y_train, y_train_pred, alpha=0.6, color='b',
label='Predicted')
plt.plot([min(y_train), max(y_train)], [min(y_train), max(y_train)],
color='red', lw=2, label='Perfect Fit')  # Perfect fit line
plt.title('Actual vs Predicted (Training Set)')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()

# Step 22: Plot Actual vs Predicted for Test set
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_test_pred, alpha=0.6, color='b', label='Predicted')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color='red', lw=2, label='Perfect Fit')  # Perfect fit line
plt.title('Actual vs Predicted (Test Set)')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()

#Random Forest to predict defect length (80/20 Split)

import pandas as pd
import numpy as np
from sklearn.model_selection import KFold, train_test_split,
RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import r2_score, mean_squared_error
from scipy.stats import randint
from sklearn.ensemble import RandomForestRegressor
```

```
import matplotlib.pyplot as plt
import seaborn as sns  # Importing seaborn for correlation matrix

# Step 1: Load the dataset
df = pd.read_csv('Connect.csv')

# Step 2: Exploratory Data Analysis (EDA)
# Check basic statistics of the dataset
print("Dataset Statistics:\n", df.describe())

# Check for missing values
print("Missing Values:\n", df.isnull().sum())

# Plot the distribution of the target variable 'Defect length'
plt.figure(figsize=(8, 6))
sns.histplot(df['Defect length'], kde=True, bins=30)
plt.title('Distribution of Defect Length')
plt.xlabel('Defect Length')
plt.ylabel('Frequency')
plt.show()

# Step 3: Correlation Matrix
features = ['Line segment number', 'Track standard number', 'Milepost',
'Total car east', 'Total car west',
            'Total train east', 'Total train west', 'Defect
amplitude','Total deflection', 'Class',
            'Freight speed', 'Passenger speed', 'Defect type']

plt.figure(figsize=(10, 8))
corr_matrix = df[features].corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Feature Correlation Matrix")
```

```
plt.show()


# Step 4: Define features and target
features = ['Milepost', 'Total car east', 'Total car west', 'Total train
east', 'Total train west',
            'Defect amplitude', 'Total deflection', 'Class', 'Freight
speed', 'Passenger speed', 'Defect type']
target = 'Defect length'


# Step 5: Prepare data
X = df[features].values
y = df[target].values


# Step 6: Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)


# Step 7: Initialize the scaler and the model
scaler = StandardScaler()
model = RandomForestRegressor(random_state=42)


# Step 8: Create a pipeline that first scales the data then fits the model
pipeline = Pipeline([
    ('scaler', scaler),
    ('regressor', model)
])


# Step 9: Set up cross-validation
kfold = KFold(n_splits=10, shuffle=True, random_state=42)


# Step 10: Define the parameter distributions to sample from
param_dist = {
```

```
    'regressor__n_estimators': randint(50, 200),
    'regressor__max_depth': [10, 15, 20],
    'regressor__min_samples_split': randint(5, 15),
    'regressor__min_samples_leaf': randint(2, 10),
    'regressor__max_features': ['sqrt', 'log2', 0.5]
}

# Step 11: Set up RandomizedSearchCV
random_search = RandomizedSearchCV(pipeline, param_dist, n_iter=50,
cv=kfold, scoring='r2', n_jobs=-1, random_state=42)

# Step 12: Fit the RandomizedSearchCV to find the best model
random_search.fit(X_train, y_train)

# Step 13: Get the best model from the random search
best_model = random_search.best_estimator_

# Step 14: Predict on the training and test sets using the best model
y_train_pred = best_model.predict(X_train)
y_test_pred = best_model.predict(X_test)

# Step 15: Calculate R² scores and RMSE for both training and test sets
r2_train = r2_score(y_train, y_train_pred)
r2_test = r2_score(y_test, y_test_pred)

rmse_train = np.sqrt(mean_squared_error(y_train, y_train_pred))
rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))

# Step 16: Output the results
print(f'Best Parameters: {random_search.best_params_}')
print(f'R² Score on Training Set: {r2_train}')
```

```
print(f'R² Score on Test Set: {r2_test}')
print(f'RMSE on Training Set: {rmse_train}')
print(f'RMSE on Test Set: {rmse_test}')

# Step 17: Extract feature importances
feature_importances =
best_model.named_steps['regressor'].feature_importances_

# Step 18: Plot feature importances
plt.figure(figsize=(10, 6))
plt.barh(features, feature_importances, color='skyblue')
plt.xlabel('Feature Importance')
plt.title('Feature Importances in RandomForestRegressor')
plt.gca().invert_yaxis()  # To have the most important at the top
plt.show()

# Step 19: Plot Actual vs Predicted for Training set
plt.figure(figsize=(8, 6))
plt.scatter(y_train, y_train_pred, alpha=0.6, color='b',
label='Predicted')
plt.plot([min(y_train), max(y_train)], [min(y_train), max(y_train)],
color='red', lw=2, label='Perfect Fit')  # Perfect fit line
plt.title('Actual vs Predicted (Training Set)')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()

# Step 20: Plot Actual vs Predicted for Test set
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_test_pred, alpha=0.6, color='b', label='Predicted')
```

```
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color='red', lw=2, label='Perfect Fit')  # Perfect fit line
plt.title('Actual vs Predicted (Test Set)')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()

# Cat boost to predict defect length (80/20 Split)

from catboost import CatBoostRegressor
from sklearn.model_selection import KFold, RandomizedSearchCV
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Assuming your DataFrame (df) is already loaded
# Example: df = pd.read_csv('your_file.csv')

# Step 1: Define features and target
features = ['Line segment number', 'Track standard number', 'Milepost',
'Total car east', 'Total car west',
            'Total train east', 'Total train west', 'Defect amplitude',
'Total deflection', 'Class',
            'Freight speed', 'Passenger speed', 'Defect type']
target = 'Defect length'

# Step 2: Prepare data
X = df[features].values
```

```
y = df[target].values


# Step 3: Split the dataset 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 4: Define a pipeline that first scales the data then applies
CatBoost Regressor
pipeline = Pipeline([
    ('scaler', StandardScaler()),                # Standardize the
features
    ('catboost', CatBoostRegressor(verbose=0, random_state=42))  #
CatBoost model
])


# Step 5: Define the refined parameter grid for RandomizedSearchCV
param_distributions = {
    'catboost__iterations': [100, 150, 200],  # Lower iterations to
prevent overfitting
    'catboost__depth': [6, 8, 10],  # Shallower trees for reduced
complexity
    'catboost__learning_rate': [0.05, 0.1],  # Smaller learning rates for
more controlled updates
    'catboost__l2_leaf_reg': [7, 9, 11],  # Stronger regularization to
prevent overfitting
    'catboost__bagging_temperature': [1, 2, 3]  # More randomness in
bagging
}


# Step 6: Set up KFold cross-validation with 10 folds
kfold = KFold(n_splits=10, shuffle=True, random_state=42)


# Step 7: Perform hyperparameter tuning with RandomizedSearchCV
catboost_search = RandomizedSearchCV(
```

```
    estimator=pipeline,
    param_distributions=param_distributions,
    n_iter=20,  # Number of random parameter combinations to try
    scoring='r2',
    cv=kfold,  # 10-fold cross-validation
    verbose=1,
    n_jobs=-1,
    random_state=42
)

# Step 8: Fit RandomizedSearchCV to the training data
catboost_search.fit(X_train, y_train)

# Step 9: Output the best parameters and the cross-validation R² score
best_catboost_params = catboost_search.best_params_
best_catboost_score = catboost_search.best_score_
print(f"Best parameters for CatBoost: {best_catboost_params}")
print(f"Best cross-validation R² score for CatBoost:
{best_catboost_score}")

# Step 10: Get the best model from RandomizedSearchCV
best_catboost_model = catboost_search.best_estimator_

# Step 11: Predict on the test data
y_test_pred_catboost = best_catboost_model.predict(X_test)

# Step 12: Predict on the training data
y_train_pred_catboost = best_catboost_model.predict(X_train)

# Step 13: Calculate and print the R-squared score for the test and
training sets
r2_test_catboost = r2_score(y_test, y_test_pred_catboost)
```

```
r2_train_catboost = r2_score(y_train, y_train_pred_catboost)


print(f"R-squared (Test) for CatBoost: {r2_test_catboost}")
print(f"R-squared (Training) for CatBoost: {r2_train_catboost}")


# Step 14: Calculate RMSE for both training and test sets
rmse_train_catboost = np.sqrt(mean_squared_error(y_train,
y_train_pred_catboost))
rmse_test_catboost = np.sqrt(mean_squared_error(y_test,
y_test_pred_catboost))


print(f"RMSE (Training) for CatBoost: {rmse_train_catboost}")
print(f"RMSE (Test) for CatBoost: {rmse_test_catboost}")


# Step 15: Plot Actual vs Predicted for Training set
plt.figure(figsize=(8, 6))
plt.scatter(y_train, y_train_pred_catboost, alpha=0.6, color='b',
label='Predicted')
plt.plot([min(y_train), max(y_train)], [min(y_train), max(y_train)],
color='red', lw=2, label='Perfect Fit')  # Perfect fit line
plt.title('Actual vs Predicted (Training Set)')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()


# Step 16: Plot Actual vs Predicted for Test set
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_test_pred_catboost, alpha=0.6, color='b',
label='Predicted')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color='red', lw=2, label='Perfect Fit')  # Perfect fit line
plt.title('Actual vs Predicted (Test Set)')
```

```
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()

#XGBoost to predict repeated defects

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import KFold, RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import r2_score, mean_squared_error
import xgboost as xgb
from scipy.stats import uniform, randint

# Step 1: Define features and target
features = ['Line segment number', 'Track standard number', 'Milepost',
'Total car east', 'Total car west',
            'Total train east', 'Total train west', 'Defect amplitude',
'Total deflection', 'Class',
            'Freight speed', 'Passenger speed', 'Defect type']
target = 'Defect length'

# Step 2: Load the train and test data
train_df = pd.read_csv('Defecttrain.csv')
test_df = pd.read_csv('Defecttest.csv')

# Step 3: Exploratory Data Analysis (EDA)
# Check basic statistics of the training dataset
```

```
print("Training Data Statistics:\n", train_df.describe())

# Check for missing values in the training and test datasets
print("Missing Values in Training Data:\n", train_df.isnull().sum())
print("Missing Values in Test Data:\n", test_df.isnull().sum())

# Plot the distribution of the target variable 'Defect length' in the
training data
plt.figure(figsize=(8, 6))
sns.histplot(train_df['Defect length'], kde=True, bins=30)
plt.title('Distribution of Defect Length in Training Data')
plt.xlabel('Defect Length')
plt.ylabel('Frequency')
plt.show()

# Step 5: Extract features and target
X_train = train_df[features].values
y_train = train_df[target].values
X_test = test_df[features].values
y_test = test_df[target].values

# Step 6: Initialize the scaler and the model
scaler = StandardScaler()
model = xgb.XGBRegressor(objective='reg:squarederror', random_state=42)

# Step 7: Create a pipeline that first scales the data then fits the model
pipeline = Pipeline([
    ('scaler', scaler),
    ('regressor', model)
])
```

```
# Step 8: Set up cross-validation
kfold = KFold(n_splits=10, shuffle=True, random_state=42)

# Step 9: Define the parameter distributions to sample from
param_dist = {
    'regressor__n_estimators': randint(100, 300),
    'regressor__max_depth': randint(3, 10),
    'regressor__learning_rate': uniform(0.01, 0.2),
    'regressor__subsample': uniform(0.6, 0.4),
    'regressor__colsample_bytree': uniform(0.6, 0.4),
    'regressor__min_child_weight': randint(1, 10),
    'regressor__gamma': uniform(0, 0.5),
    'regressor__reg_alpha': uniform(0, 1),  # L1 regularization term
    'regressor__reg_lambda': uniform(0, 1)  # L2 regularization term
}

# Step 10: Set up RandomizedSearchCV
random_search = RandomizedSearchCV(pipeline, param_dist, n_iter=50,
cv=kfold, scoring='r2', n_jobs=-1, random_state=42)

# Step 11: Fit the RandomizedSearchCV to find the best model
random_search.fit(X_train, y_train)

# Step 12: Get the best model from the random search
best_model = random_search.best_estimator_

# Step 13: Predict on the training and test sets using the best model
y_train_pred = best_model.predict(X_train)
y_test_pred = best_model.predict(X_test)

# Step 14: Calculate R² scores
```

```
r2_train = r2_score(y_train, y_train_pred)
r2_test = r2_score(y_test, y_test_pred)

# Step 15: Calculate RMSE for both training and test sets
rmse_train = np.sqrt(mean_squared_error(y_train, y_train_pred))
rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))

# Step 16: Output the results
print(f'Best Parameters: {random_search.best_params_}')
print(f'R² Score on Training Set: {r2_train}')
print(f'R² Score on Test Set: {r2_test}')
print(f'RMSE on Training Set: {rmse_train}')
print(f'RMSE on Test Set: {rmse_test}')

# Step 17: Plot Actual vs Predicted for Training set
plt.figure(figsize=(8, 6))
plt.scatter(y_train, y_train_pred, alpha=0.6, color='blue',
label='Predicted')
plt.plot([min(y_train), max(y_train)], [min(y_train), max(y_train)],
color='red', lw=2, label='Perfect Fit')  # Perfect fit line
plt.title('Actual vs Predicted (Training Set)')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()

# Step 18: Plot Actual vs Predicted for Test set
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_test_pred, alpha=0.6, color='green',
label='Predicted')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color='red', lw=2, label='Perfect Fit')  # Perfect fit line
```

```
plt.title('Actual vs Predicted (Test Set)')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()

#Random Forest (Repeated defect number)

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import KFold, RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import r2_score, mean_squared_error
from scipy.stats import randint
import numpy as np

# Step 1: Define features and target
features = ['Line segment number', 'Track standard number', 'Milepost',
'Total car east', 'Total car west',
            'Total train east', 'Total train west', 'Defect
amplitude','Total deflection', 'Class',
            'Freight speed', 'Passenger speed', 'Defect type']
target = 'Defect length'

# Step 2: Load the train and test data
train_df = pd.read_csv('Defecttrain.csv')
test_df = pd.read_csv('Defecttest.csv')

# Step 3: Exploratory Data Analysis (EDA)
```

```
# Check basic statistics of the training dataset
print("Training Data Statistics:\n", train_df.describe())

# Check for missing values in the training and test datasets
print("Missing Values in Training Data:\n", train_df.isnull().sum())
print("Missing Values in Test Data:\n", test_df.isnull().sum())

# Plot the distribution of the target variable 'Defect length' in the
training data
plt.figure(figsize=(8, 6))
sns.histplot(train_df['Defect length'], kde=True, bins=30)
plt.title('Distribution of Defect Length in Training Data')
plt.xlabel('Defect Length')
plt.ylabel('Frequency')
plt.show()

# Step 5: Extract features and target
X_train = train_df[features].values
y_train = train_df[target].values
X_test = test_df[features].values
y_test = test_df[target].values

# Step 6: Initialize the scaler and the model
scaler = StandardScaler()
model = RandomForestRegressor(random_state=42)

# Step 7: Create a pipeline that first scales the data then fits the model
pipeline = Pipeline([
    ('scaler', scaler),
    ('regressor', model)
])
```

```
# Step 8: Set up cross-validation
kfold = KFold(n_splits=10, shuffle=True, random_state=42)

# Step 9: Define the parameter distributions to sample from
param_dist = {
    'regressor__n_estimators': randint(50, 200),
    'regressor__max_depth': [10, 15, 20],
    'regressor__min_samples_split': randint(5, 15),
    'regressor__min_samples_leaf': randint(2, 10),
    'regressor__max_features': ['sqrt', 'log2', 0.5]
}

# Step 10: Set up RandomizedSearchCV
random_search = RandomizedSearchCV(pipeline, param_dist, n_iter=50,
cv=kfold, scoring='r2', n_jobs=-1, random_state=42)

# Step 11: Fit the RandomizedSearchCV to find the best model
random_search.fit(X_train, y_train)

# Step 12: Get the best model from the random search
best_model = random_search.best_estimator_

# Step 13: Predict on the training and test sets using the best model
y_train_pred = best_model.predict(X_train)
y_test_pred = best_model.predict(X_test)

# Step 14: Calculate R² scores
r2_train = r2_score(y_train, y_train_pred)
r2_test = r2_score(y_test, y_test_pred)
```

```
# Step 15: Calculate RMSE for both training and test sets
rmse_train = np.sqrt(mean_squared_error(y_train, y_train_pred))
rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))

# Step 16: Output the results
print(f'Best Parameters: {random_search.best_params_}')
print(f'R² Score on Training Set: {r2_train}')
print(f'R² Score on Test Set: {r2_test}')
print(f'RMSE on Training Set: {rmse_train}')
print(f'RMSE on Test Set: {rmse_test}')

# Step 17: Plot Actual vs Predicted for Training set
plt.figure(figsize=(8, 6))
plt.scatter(y_train, y_train_pred, alpha=0.6, color='blue',
label='Predicted')
plt.plot([min(y_train), max(y_train)], [min(y_train), max(y_train)],
color='red', lw=2, label='Perfect Fit')  # Perfect fit line
plt.title('Actual vs Predicted (Training Set)')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()

# Step 18: Plot Actual vs Predicted for Test set
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_test_pred, alpha=0.6, color='green',
label='Predicted')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color='red', lw=2, label='Perfect Fit')  # Perfect fit line
plt.title('Actual vs Predicted (Test Set)')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
```

```
plt.legend()
plt.show()

# Cat Boost to predict the repeated defect length

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import KFold, RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import r2_score, mean_squared_error
from catboost import CatBoostRegressor
from scipy.stats import uniform, randint

# Step 1: Define features and target
features = ['Line segment number', 'Track standard number', 'Milepost',
'Total car east', 'Total car west',
            'Total train east', 'Total train west', 'Defect amplitude',
'Total deflection', 'Class',
            'Freight speed', 'Passenger speed', 'Defect type']
target = 'Defect length'

# Step 2: Load the train and test data
train_df = pd.read_csv('Defecttrain.csv')
test_df = pd.read_csv('Defecttest.csv')

# Step 3: Exploratory Data Analysis (EDA)
# Check basic statistics of the training dataset
print("Training Data Statistics:\n", train_df.describe())
```

```
# Check for missing values in the training and test datasets
print("Missing Values in Training Data:\n", train_df.isnull().sum())
print("Missing Values in Test Data:\n", test_df.isnull().sum())

# Plot the distribution of the target variable 'Defect length' in the
training data
plt.figure(figsize=(8, 6))
sns.histplot(train_df['Defect length'], kde=True, bins=30)
plt.title('Distribution of Defect Length in Training Data')
plt.xlabel('Defect Length')
plt.ylabel('Frequency')
plt.show()

# Step 5: Extract features and target
X_train = train_df[features].values
y_train = train_df[target].values
X_test = test_df[features].values
y_test = test_df[target].values

# Step 6: Initialize the scaler and the model
scaler = StandardScaler()
model = CatBoostRegressor(verbose=0, random_state=42)

# Step 7: Create a pipeline that first scales the data then fits the model
pipeline = Pipeline([
    ('scaler', scaler),
    ('regressor', model)
])

# Step 8: Set up cross-validation
kfold = KFold(n_splits=10, shuffle=True, random_state=42)
```

```
# Step 9: Define the refined parameter distributions for CatBoost
param_dist = {
    'regressor__iterations': randint(500, 1000),     # Increase iterations for gradual learning
    'regressor__depth': randint(6, 7),               # Reduce depth to prevent overfitting
    'regressor__learning_rate': uniform(0.03, 0.05), # Lower learning rate for smaller updates
    'regressor__l2_leaf_reg': uniform(10, 15),       # Increase L2 regularization
    'regressor__subsample': uniform(0.6, 0.4),       # Subsample ratio of the training instances
    'regressor__bagging_temperature': uniform(0, 1), # Control randomness in bagging
}

# Step 10: Set up RandomizedSearchCV for CatBoost
random_search = RandomizedSearchCV(
    pipeline, param_dist, n_iter=50, cv=kfold, scoring='r2', n_jobs=-1, random_state=42, verbose=2
)

# Step 11: Fit the RandomizedSearchCV to find the best model
random_search.fit(X_train, y_train)

# Step 12: Get the best model from the random search
best_model = random_search.best_estimator_

# Step 13: Predict on the training and test sets using the best model
y_train_pred = best_model.predict(X_train)
y_test_pred = best_model.predict(X_test)
```

```
# Step 14: Calculate R² scores
r2_train = r2_score(y_train, y_train_pred)
r2_test = r2_score(y_test, y_test_pred)

# Step 15: Calculate RMSE for both training and test sets
rmse_train = np.sqrt(mean_squared_error(y_train, y_train_pred))
rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))

# Step 16: Output the results
print(f'Best Parameters: {random_search.best_params_}')
print(f'R² Score on Training Set: {r2_train}')
print(f'R² Score on Test Set: {r2_test}')
print(f'RMSE on Training Set: {rmse_train}')
print(f'RMSE on Test Set: {rmse_test}')

# Step 17: Plot Actual vs Predicted for Training set
plt.figure(figsize=(8, 6))
plt.scatter(y_train, y_train_pred, alpha=0.6, color='blue',
label='Predicted')
plt.plot([min(y_train), max(y_train)], [min(y_train), max(y_train)],
color='red', lw=2, label='Perfect Fit')  # Perfect fit line
plt.title('Actual vs Predicted (Training Set)')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()

# Step 18: Plot Actual vs Predicted for Test set
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_test_pred, alpha=0.6, color='green',
label='Predicted')
```

```
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color='red', lw=2, label='Perfect Fit')  # Perfect fit line
plt.title('Actual vs Predicted (Test Set)')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()

#XGBoost (Increase in defect length)

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import KFold, train_test_split,
RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import r2_score, mean_squared_error
import xgboost as xgb
from scipy.stats import uniform, randint

# Step 1: Load the dataset
df = pd.read_csv('Defect length increase.csv')

# Step 2: Exploratory Data Analysis (EDA)
# Check basic statistics of the dataset
print("Dataset Statistics:\n", df.describe())

# Check for missing values
print("Missing Values:\n", df.isnull().sum())
```

```
# Plot the distribution of the target variable 'Defect length'
plt.figure(figsize=(8, 6))
sns.histplot(df['Defect length'], kde=True, bins=30)
plt.title('Distribution of Defect Length')
plt.xlabel('Defect Length')
plt.ylabel('Frequency')
plt.show()

# Step 4: Define features and target
features = ['Defect amplitude', 'Previous defect length', 'Time gap',
'Defect type']
target = 'Defect length'

# Step 5: Prepare data
X = df[features].values
y = df[target].values

# Step 6: Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Step 7: Initialize the scaler and the model
scaler = StandardScaler()
model = xgb.XGBRegressor(objective='reg:squarederror', random_state=42)

# Step 8: Create a pipeline that first scales the data then fits the model
pipeline = Pipeline([
    ('scaler', scaler),
    ('regressor', model)
])

# Step 9: Set up cross-validation
```

```
kfold = KFold(n_splits=10, shuffle=True, random_state=42)

# Step 10: Define the parameter distributions to sample from
param_dist = {
    'regressor__n_estimators': randint(100, 500),
    'regressor__max_depth': randint(3, 15),
    'regressor__learning_rate': uniform(0.01, 0.2),
    'regressor__subsample': uniform(0.6, 0.4),
    'regressor__colsample_bytree': uniform(0.6, 0.4),
    'regressor__min_child_weight': randint(1, 10),
    'regressor__gamma': uniform(0, 0.5),
    'regressor__reg_alpha': uniform(0, 1),
    'regressor__reg_lambda': uniform(0, 1)
}

# Step 11: Set up RandomizedSearchCV
random_search = RandomizedSearchCV(pipeline, param_dist, n_iter=100,
cv=kfold, scoring='r2', n_jobs=-1, random_state=42)

# Step 12: Fit the RandomizedSearchCV to find the best model
random_search.fit(X_train, y_train)

# Step 13: Get the best model from the random search
best_model = random_search.best_estimator_

# Step 14: Predict on the training and test sets using the best model
y_train_pred = best_model.predict(X_train)
y_test_pred = best_model.predict(X_test)

# Step 15: Calculate R² scores
r2_train = r2_score(y_train, y_train_pred)
```

```
r2_test = r2_score(y_test, y_test_pred)

# Step 16: Calculate RMSE for both training and test sets
rmse_train = np.sqrt(mean_squared_error(y_train, y_train_pred))
rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))

# Step 17: Output the results
print(f'Best Parameters for XGBoost: {random_search.best_params_}')
print(f'R² Score on Training Set: {r2_train}')
print(f'R² Score on Test Set: {r2_test}')
print(f'RMSE on Training Set: {rmse_train}')
print(f'RMSE on Test Set: {rmse_test}')

# Step 18: Plot Actual vs Predicted for Training set
plt.figure(figsize=(8, 6))
plt.scatter(y_train, y_train_pred, alpha=0.6, color='blue',
label='Predicted')
plt.plot([min(y_train), max(y_train)], [min(y_train), max(y_train)],
color='red', lw=2, label='Perfect Fit')  # Perfect fit line
plt.title('Actual vs Predicted (Training Set)')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()

# Step 19: Plot Actual vs Predicted for Test set
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_test_pred, alpha=0.6, color='green',
label='Predicted')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color='red', lw=2, label='Perfect Fit')  # Perfect fit line
plt.title('Actual vs Predicted (Test Set)')
```

```
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()

#Random Forest (Increase in defect length)

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import KFold, train_test_split,
RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import r2_score, mean_squared_error
from scipy.stats import randint
import numpy as np

# Step 1: Load the new dataset
df = pd.read_csv('Defect length increase.csv')

# Step 2: Exploratory Data Analysis (EDA)
# Check basic statistics of the dataset
print("Dataset Statistics:\n", df.describe())

# Check for missing values
print("Missing Values:\n", df.isnull().sum())

# Plot the distribution of the target variable 'Defect length'
plt.figure(figsize=(8, 6))
```

```
sns.histplot(df['Defect length'], kde=True, bins=30)
plt.title('Distribution of Defect Length')
plt.xlabel('Defect Length')
plt.ylabel('Frequency')
plt.show()

# Step 4: Define features and target
features = ['Defect amplitude', 'Previous defect length', 'Time gap',
'Defect type']
target = 'Defect length'

# Step 5: Prepare data
X = df[features].values
y = df[target].values

# Step 6: Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Step 7: Initialize the scaler and the model
scaler = StandardScaler()
model = RandomForestRegressor(random_state=42)

# Step 8: Create a pipeline that first scales the data then fits the model
pipeline = Pipeline([
    ('scaler', scaler),
    ('regressor', model)
])

# Step 9: Set up cross-validation
kfold = KFold(n_splits=10, shuffle=True, random_state=42)
```

```
# Step 10: Define the parameter distributions to sample from
param_dist = {
    'regressor__n_estimators': randint(50, 200),
    'regressor__max_depth': [10, 15, 20],
    'regressor__min_samples_split': randint(5, 15),
    'regressor__min_samples_leaf': randint(2, 10),
    'regressor__max_features': ['sqrt', 'log2', 0.5]
}

# Step 11: Set up RandomizedSearchCV
random_search = RandomizedSearchCV(pipeline, param_dist, n_iter=50,
cv=kfold, scoring='r2', n_jobs=-1, random_state=42)

# Step 12: Fit the RandomizedSearchCV to find the best model
random_search.fit(X_train, y_train)

# Step 13: Get the best model from the random search
best_model = random_search.best_estimator_

# Step 14: Predict on the training and test sets using the best model
y_train_pred = best_model.predict(X_train)
y_test_pred = best_model.predict(X_test)

# Step 15: Calculate R² scores
r2_train = r2_score(y_train, y_train_pred)
r2_test = r2_score(y_test, y_test_pred)

# Step 16: Calculate RMSE for both training and test sets
rmse_train = np.sqrt(mean_squared_error(y_train, y_train_pred))
rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))
```

```
# Step 17: Output the results
print(f'Best Parameters: {random_search.best_params_}')
print(f'R² Score on Training Set: {r2_train}')
print(f'R² Score on Test Set: {r2_test}')
print(f'RMSE on Training Set: {rmse_train}')
print(f'RMSE on Test Set: {rmse_test}')

# Step 18: Plot Actual vs Predicted for Training set
plt.figure(figsize=(8, 6))
plt.scatter(y_train, y_train_pred, alpha=0.6, color='blue',
label='Predicted')
plt.plot([min(y_train), max(y_train)], [min(y_train), max(y_train)],
color='red', lw=2, label='Perfect Fit')  # Perfect fit line
plt.title('Actual vs Predicted (Training Set)')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()

# Step 19: Plot Actual vs Predicted for Test set
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_test_pred, alpha=0.6, color='green',
label='Predicted')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color='red', lw=2, label='Perfect Fit')  # Perfect fit line
plt.title('Actual vs Predicted (Test Set)')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()

#CatBoost to predict increase in defect length
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import KFold, train_test_split,
RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import r2_score, mean_squared_error
from catboost import CatBoostRegressor
from scipy.stats import uniform, randint

# Step 1: Load the dataset
df = pd.read_csv('Defect length increase.csv')

# Step 2: Exploratory Data Analysis (EDA)
print("Dataset Statistics:\n", df.describe())
print("Missing Values:\n", df.isnull().sum())

# Plot the distribution of the target variable 'Defect length'
plt.figure(figsize=(8, 6))
sns.histplot(df['Defect length'], kde=True, bins=30)
plt.title('Distribution of Defect Length')
plt.xlabel('Defect Length')
plt.ylabel('Frequency')
plt.show()

# Step 4: Define features and target
features = ['Defect amplitude', 'Previous defect length', 'Time gap',
'Defect type']
target = 'Defect length'
```

```
# Step 5: Prepare data
X = df[features].values
y = df[target].values

# Step 6: Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Step 7: Initialize the scaler and the model
scaler = StandardScaler()
model = CatBoostRegressor(objective='RMSE', random_state=42, verbose=0,
early_stopping_rounds=100)

# Step 8: Create a pipeline that first scales the data then fits the model
pipeline = Pipeline([
    ('scaler', scaler),
    ('regressor', model)
])

# Step 9: Set up cross-validation
kfold = KFold(n_splits=10, shuffle=True, random_state=42)

# Step 10: Define the parameter distributions to sample from
param_dist = {
    'regressor__depth': randint(3, 6),  # Reduce depth to prevent
overfitting
    'regressor__learning_rate': uniform(0.01, 0.03),  # Keep learning rate
low
    'regressor__iterations': randint(300, 600),  # Decrease iterations to
avoid overfitting
    'regressor__l2_leaf_reg': uniform(10, 20),  # Increase L2
regularization
```

```
    'regressor__bagging_temperature': uniform(0.3, 1),  # Regularization
    'regressor__subsample': uniform(0.5, 0.5),  # Subsample in (0.5, 1] to
avoid errors
    'regressor__random_strength': uniform(5, 10)  # Increase randomness
}

# Step 11: Set up RandomizedSearchCV
random_search = RandomizedSearchCV(pipeline, param_dist, n_iter=100,
cv=kfold, scoring='r2', n_jobs=-1, random_state=42, error_score='raise')

# Step 12: Fit the RandomizedSearchCV to find the best model
random_search.fit(X_train, y_train)

# Step 13: Get the best model from the random search
best_model = random_search.best_estimator_

# Step 14: Predict on the training and test sets using the best model
y_train_pred = best_model.predict(X_train)
y_test_pred = best_model.predict(X_test)

# Step 15: Calculate R² scores
r2_train = r2_score(y_train, y_train_pred)
r2_test = r2_score(y_test, y_test_pred)

# Step 16: Calculate RMSE for both training and test sets
rmse_train = np.sqrt(mean_squared_error(y_train, y_train_pred))
rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))

# Step 17: Output the results
print(f'Best Parameters for CatBoost: {random_search.best_params_}')
print(f'R² Score on Training Set: {r2_train}')
print(f'R² Score on Test Set: {r2_test}')
```

```
print(f'RMSE on Training Set: {rmse_train}')
print(f'RMSE on Test Set: {rmse_test}')


# Step 18: Plot Actual vs Predicted for Training set
plt.figure(figsize=(8, 6))
plt.scatter(y_train, y_train_pred, alpha=0.6, color='blue',
label='Predicted')
plt.plot([min(y_train), max(y_train)], [min(y_train), max(y_train)],
color='red', lw=2, label='Perfect Fit')  # Perfect fit line
plt.title('Actual vs Predicted (Training Set)')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()


# Step 19: Plot Actual vs Predicted for Test set
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_test_pred, alpha=0.6, color='green',
label='Predicted')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color='red', lw=2, label='Perfect Fit')  # Perfect fit line
plt.title('Actual vs Predicted (Test Set)')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()
```