

https://dspace.org/

Modelling of MF2 Platinum Concentrators with Blended Feeds

Mneno, Dumile Peabo

2025
University of the Witwatersrand, Johannesburg
https://hdl.handle.net/10539/47578
Mneno, Dumile Peabo. (2025).Modelling of MF2 Platinum Concentrators with Blended Feeds [Master`s dissertation, University of the Witwatersrand, Johannesburg]. WIReDSpace. https://hdl.handle.net/10539/47578

Downloaded from a DSpace repository. https://dspace.org/



UNIVERSITY OF THE
WITWATERSRAND,
JOHANNESBURG

MODELLING OF MF2 PLATINUM CONCENTRATORS WITH BLENDED FEEDS

RESEARCH DISSERTATION

M.Sc. Full Research

Prepared by

Dumile Peabo Mneno

1726263

Submitted to

School of Chemical and Metallurgical Engineering, Faculty of Engineering and the Built Environment, University of the Witwatersrand, Johannesburg, South Africa

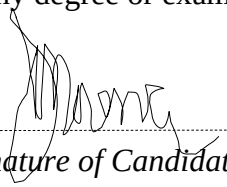
Supervisor(s):

Mr Antony Higginson and Professor Kevin Brooks

May 2025

PLAGIARISM DECLARATION

I declare that this dissertation is my unaided work. It is being submitted to the Degree of Master of Science at the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination to any other University.



(Signature of Candidate)

30 day of April , 2025

ABSTRACT

This study presents a data-driven approach to modelling mill-float-mill-float (MF2) platinum concentrator performance when processing blended feeds of Merensky and UG2 ores. Conventional phenomenological models are often limited by data availability and delayed laboratory results, hindering timely decision-making in complex mineral processing environments. Using real plant data from Two Rivers Platinum, this research seeks to understand how the MF2 circuit is operated with and without blended feeds and apply machine learning (ML) algorithms to predict ore mineralogy and plant performance. After comprehensive data cleaning and feature selection, four ML models: Linear Regression (LR), Random Forest (RF), Extra Tree (ET), and k-Nearest Neighbours (kNN) were developed and evaluated. The ET algorithm achieved superior accuracy in predicting ore mineralogy, while RF was most effective in forecasting key plant performance indicators. Despite constraints in data granularity, particularly with weekly mineralogical data, the models demonstrated robust performance and potential for integration into production planning. These findings underscore the value of ML in enhancing metallurgical plant responsiveness and optimising recovery in MF2 circuits handling variable ore blends. The research advocates for more granular data collection to unlock the full potential of predictive modelling and highlights machine learning as a scalable tool for process control in the platinum mining sector using blended feeds.

DEDICATION

To my beloved wife, whose unwavering love, support, and encouragement have been the foundation of this journey. You have been my partner in every challenge, and I am eternally grateful for that.

To my mother, whose unwavering belief in me has been a constant source of strength.

To my brother and sister, for their love and understanding, which has kept me grounded.

This work is a testament to your love, sacrifices, and faith in me.

ACKNOWLEDGEMENTS

I humbly acknowledge the Almighty Lord for His unwavering strength and guidance, which have been the cornerstone of my master's research journey. His mercy and grace have been the source of my perseverance and inspiration, enabling me to navigate complex challenges with humility and gratitude.

I express my deepest gratitude to my supervisors, Mr Antony Higginson and Professor Kevin Brooks. Their unwavering support, mentorship, and constant encouragement have been instrumental in the success of this research paper and in shaping my skills. I am forever indebted to them for believing in my abilities and providing this opportunity.

I am grateful to the data analytics, numeric and control engineering unit. Your unwavering support and collaboration have been instrumental in the success of this research.

I sincerely thank Tumelo Machete and Cynthia Bhengu for their tremendous support and patience. The data you provided made this study possible

I sincerely thank my sponsor, African Rainbow Minerals, for their generous financial assistance in making this study possible. Their support is deeply appreciated, and their contribution to this research is invaluable.

CONTENTS

PLAGIARISM DECLARATION ..... **i**
ABSTRACT ..... **ii**
DEDICATION ..... **iii**
ACKNOWLEDGEMENTS ..... **iv**
CONTENTS ..... **v**
LIST OF FIGURES ..... **ix**
LIST OF TABLES ..... **ix**
NOMENCLATURE ..... **x**
1. INTRODUCTION ..... **- 1 -**
1.1. PROBLEM STATEMENT ..... - 2 -
1.2. RESEARCH OBJECTIVES ..... - 3 -
2. LITERATURE REVIEW ..... **- 4 -**
2.1. Process Overview ..... - 5 -
2.1.1. Mineralogy ..... - 5 -
2.1.2. Comminution ..... - 6 -
2.1.3. Flotation ..... - 7 -
2.1.4. Mill-Float-Mill-Float Circuit ..... - 7 -
2.2. Model Development ..... - 9 -
2.2.1. Phenomenological Modelling ..... - 9 -
2.2.2. Data-based Modelling ..... - 11 -
2.2.3. Model Validation ..... - 22 -
2.2.4. Data Limitations in Machine Learning ..... - 23 -
2.2.5. Data Partitioning ..... - 24 -
2.2.6. Detection of False Data ..... - 24 -
2.2.7. Overfitting ..... - 25 -
2.2.8. Plant Optimisation ..... - 25 -
2.2.9. Hyperparameter Tuning ..... - 26 -
2.2.10. Geometallurgy ..... - 27 -
2.2.11. Comminution Modelling ..... - 28 -

2.2.12. Flotation Modelling ..... - 30 -
2.3. Sci-kit Learn ..... - 33 -
3. METHODOLOGY ..... - 34 -
3.1. Source of Data ..... - 34 -
3.1.1. Dataset Limitations and Granularity Constraints ..... - 35 -
3.2. Structure of the Dataset ..... - 35 -
3.3. Data Pre-Processing ..... - 36 -
3.3.1. Time Series Detection ..... - 37 -
3.3.2. Correlation Analysis ..... - 37 -
3.3.3. Removal of Unnecessary Feature ..... - 38 -
3.3.4. Quality Assurance ..... - 38 -
3.3.5. Outlier Detection ..... - 39 -
3.4. Feature Engineering and Selection ..... - 39 -
3.5. Partition of the Data ..... - 40 -
3.6. Model Selection ..... - 40 -
3.6.1. Linear Regression ..... - 41 -
3.6.2. Random Forest ..... - 41 -
3.6.3. Extra Tree ..... - 42 -
3.6.4. k-Nearest Neighbours ..... - 43 -
3.7. Model Evaluation Metrics ..... - 44 -
3.8. Naïve Benchmark ..... - 45 -
4. RESULTS ..... - 46 -
4.1. Data Preprocessing and Distribution Analysis ..... - 46 -
6.1. Time Series ..... - 46 -
6.2. Influence of Merensky Incorporation ..... - 47 -
6.3. Correlation Analysis ..... - 48 -
4.4. Real-Time Prediction of Ore Mineralogy ..... - 50 -
4.4.1. Feature Engineering and Selection ..... - 51 -
4.4.2. Linear Regression ..... - 51 -
4.4.3. Random Forest Regression ..... - 54 -

4.4.4. Extra Tree Regression ..... - 56 -
4.4.5. K-Nearest Neighbours Regression ..... - 59 -
4.4.6. Performance Evaluation of the Multiple Predictive Models ..... - 62 -
4.5. Real-Time Prediction of Plant Performance ..... - 63 -
4.5.1. Naïve Benchmark ..... - 63 -
4.5.2. Feature Engineering and Selection ..... - 64 -
4.5.3. Linear Regression ..... - 65 -
4.5.4. Random Forest Regression ..... - 66 -
4.5.5. Extra Tree Regression ..... - 68 -
4.5.6. K-Nearest Neighbours Regression ..... - 70 -
4.5.7. Performance Evaluation of the Multiple Predictive Models ..... - 72 -
4.6. Execution Time Analysis of Predictive Models in Plant Performance ..... - 73 -
5. DISCUSSION ..... - 74 -
5.1. Plant Performance Evaluation ..... - 74 -
5.2. Prediction of Ore Mineralogy ..... - 76 -
5.3. Prediction of Plant Performance ..... - 79 -
5.4. Execution Time Analysis of Predictive Models in Plant Performance ..... - 80 -
6. CONCLUSION ..... - 82 -
7. REFERENCES ..... - 84 -
APPENDIX A: Python Script for Data Cleaning ..... - 90 -
A1. Library ..... - 90 -
A2. Data Collection and Formatting ..... - 93 -
A3. Time Series Testing ..... - 94 -
A4. Data Preparation ..... - 96 -
A4.1. Removal of Empty Columns ..... - 96 -
A4.2. Removal of Columns with weak correlations ..... - 97 -
A4.3. Removal of NaN entries ..... - 98 -
A4.4. Removal of Negative values ..... - 99 -
Removal of Negative values ..... - 100 -
APPENDIX B: Python Script for Ore Mineralogy Prediction ..... - 105 -

B1. Feature Selection ..... - 105 -
B2. Model Development ..... - 107 -
B2.1. Naïve Benchmark ..... - 107 -
B2.2. Linear Regression ..... - 108 -
B2.3. Random Forest Regression ..... - 113 -
B2.4. Extra Tree Regression ..... - 118 -
B2.5. k-Nearest Neighbour Regression ..... - 123 -
APPENDIX C: Python Script for Plant Performance Prediction ..... - 128 -
C1. Feature Selection ..... - 128 -
C2. Model Development ..... - 132 -
C2.1. Naïve Benchmark ..... - 132 -
C2.2. Linear Regression ..... - 133 -
C2.3. Random Forest Regression ..... - 138 -
C2.4. Extra Tree Regression ..... - 143 -
C2.5. k-Nearest Neighbour Regression ..... - 148 -
APPENDIX D: Descriptive Statistics of the Database ..... - 153 -

LIST OF FIGURES

Figure 2.1: Typical PGM Processing Flowsheet ..... - 4 -
Figure 2.2: Generic MF2 PGM concentrator schematic (C. Rule, 2011) ..... - 8 -
Figure 2.3: The implementation of the tree-based algorithm (Jearanaitanakij, 1999) .... - 14 -
Figure 2.4: Schematic diagram of an SVM (Zou et al., 2021) ..... - 15 -
Figure 2.5: kNN model with k = 5 (W. Zhang, 2017) ..... - 17 -
Figure 2.6: Neural Network learning process (M. Mijwil, 2021)**.** ..... - 18 -
Figure 2.7: Procedure for plant optimisation (From B. A. Wills & Finch, 2016a) ..... - 26 -
Figure 2.8: Froth Zone Mass Balance via Bubble Load (Yianatos et al., 2020) ..... - 31 -
Figure 2.9: Copper recovery profiles in rougher flotation ..... - 32 -
Figure 4.1: Total milled throughout ratio time series ..... - 47 -
Figure 4.2: Correlation Matrix of the features and targets used ..... - 49 -
Figure 4.3: Coefficient Values for the features of each target ..... - 51 -
Figure 4.4: Performance Metrics of the Linear Regression ..... - 52 -
Figure 4.5: LR prediction time series and actual vs pred. plot for ore mineralogy ..... - 53 -
Figure 4.6: Performance Metrics of the RF regression ..... - 54 -
Figure 4.7: RF prediction time series and actual vs pred. plot for ore mineralogy ..... - 55 -
Figure 4.8: Performance Metrics of the ET regression ..... - 56 -
Figure 4.9: ET prediction time series and actual vs pred. plot for ore mineralogy ..... - 58 -
Figure 4.10: Performance Metrics of the kNN regression ..... - 59 -
Figure 4.11: kNN prediction time series and actual vs pred. plot for ore mineralogy ..... - 61 -
Figure 4.12: Coefficient Values for the features of 4E and Chromium in concentrate .... - 64 -
Figure 4.13: LR prediction time series and actual vs pred. plot for plant performance .. - 66 -
Figure 4.14: RF prediction time series and actual vs pred. plot for plant performance .. - 68 -
Figure 4.15: ET prediction time series and actual vs pred. plot for plant performance .. - 70 -
Figure 4.16: kNN prediction time series and act. vs pred. plot for plant performance ... - 71 -
Figure D7.1: Operational and Ore Mineralogy data distribution ..... - 159 -

LIST OF TABLES

Table 2.1: Typical Bulk composition of PGM ores (Corin et al., 2021). ..... - 6 -
Table 3.1: Grid Search default and optimised hyperparameters for LR ..... - 41 -
Table 3.2: Random Search default and optimised hyperparameters for RF ..... - 42 -
Table 3.3: Random Search default and optimised hyperparameters for ET ..... - 43 -
Table 3.4: Grid Search default and optimised hyperparameters for kNN ..... - 44 -
Table 4.1: Time Series Test Results ..... - 46 -
Table 4.2: Influence of Merensky on the average PSD post MF1 ..... - 48 -
Table 4.3: Performance Metrics of the Predictive Model ..... - 62 -
Table 4.4: Naïve Baseline Results for Plant Performance ..... - 63 -
Table 4.5: Performance Metrics of the LR regression ..... - 65 -
Table 4.6: Performance Metrics of the RF regression ..... - 67 -
Table 4.7: Performance Metrics of the ET regression ..... - 68 -
Table 4.8: Performance Metrics of the kNN regression ..... - 70 -
Table 4.9: Performance Metrics of the Predictive Model ..... - 72 -
Table 4.10: Case Execution Length for Plant Performance ..... - 73 -

NOMENCLATURE

Abbreviation	Full Form
4E	Platinum, Palladium, Rhodium and Gold
6E	Platinum, Palladium, Rhodium, Gold, Iridium and Ruthenium
ADF	Augmented Dickey-Fuller
ANN	Artificial Neural Network
ARIMA	Autoregressive Integrated Moving Average
BIC	Bushveld Igneous Complex
BWI	Bond Work Index
ET	Extra Tree
HPT	Hyperparameter Tuning
kNN	k-nearest Neighbours
KPI	Key Performance Indicators
KPSS	Kwiatkowski-Phillips-Schmidt-Shin
LR	Linear Regression
ML	Machine Learning
MF1	Mill-Float
MF2	Mill-Float-Mill-Float
MF3	Mill-Float-Mill-Float-Mill-Float
PGE	Platinum Group Elements
PGM	Platinum Group Metals
PSD	Particle Size Distribution
RF	Random Forest
RMSE	Root Mean Squared Error
ROM	Run-of-mine
R^2	R-squared
Sklearn	Sci-kit Learn
SVM	Support Vector Machines
TRP	Two Rivers Platinum
UG2	Upper Group Two

1. INTRODUCTION

The Bushveld Igneous Complex (BIC), located within the Transvaal basin, hosts abundant mineral deposits, including platinum group minerals (PGM) (Maier & Bowen, 1996; Von Gruenewaldt et al., 1985). The primary PGM sources in the BIC are Merensky, Upper Group Two (UG2) and Platreef, with UG2 and Platreef dominating PGM production (Corin et al., 2021). These ores require distinct mineralogical processing methods to optimise PGM recovery (Monele Mokadze et al., 2021). The Merensky employs the mill-float (MF1) process, involving primary milling and flotation. UG2 uses the mill-float-mill-float (MF2) circuit, including a regrinding and a rougher/cleaner/re-cleaner cascade configuration, to recover finely disseminated silicate-associated PGMs (Nel et al., 2004). The production of the Platreef requires the mill-float-mill-float-mill-float (MF3) circuit (Corin et al., 2021). Because of the distinct processes, with the incorporation of the Merensky ore into the processing stream, the MF2 PGM throughput would be diminished (Corin et al., 2021; Monele Mokadze et al., 2021). The reduction in throughput is caused by longer milling times required by the Merensky ore due to higher concentrations of pyroxene and feldspar (Dzvinamurungu et al., 2013). Furthermore, Dzvinamurungu et al. (2013) state that UG2 recovery in the flotation section is higher than Merensky owing to the abundance of pentlandite and pyrrhotite, which naturally float faster (Dzvinamurungu et al., 2013).

This research uses data-driven techniques to dynamically adjust process parameters to maximise PGM production for plants with blended feeds. Rapid optimisation of processing operations can be realised by fully leveraging the advantages offered by analytics (Ji et al., 2019). Mineral processing operations utilising this opportunity would facilitate the expeditious enabling of process optimisation of PGM recovery. By integrating these advancements, users can optimise their operations and capitalise on the benefits of data-driven decision-making in mineral processing activities.

Digitalisation refers to using data in digital format to establish automated processes, leading to increased efficiency in task execution. These initiatives create value by giving companies a competitive edge (Szalavetz, 2022). This is achieved as data can be leveraged to identify weak

points within the process. With machine learning (ML) techniques, which include the statistical exploitation of the collected data, additional process parameters (e.g., product quality) can be predicted to affect value creation and insights resulting from data analytics (Kadlec et al., 2011).

1.1. PROBLEM STATEMENT

Integrating Merensky into the MF2 circuit significantly decreases PGM throughput due to the requirement for extended milling times. This is attributed to the abundance of pyroxene and feldspar in Merensky ore (Dzvinamurungu et al., 2013). In addition, the dependence on traditional laboratory analysis, with a 24-hour delay in providing assay results for sampled flows, converts process plants into operational 'black boxes' as they do not allow for rapid decision-making.

Moreover, existing models available in the literature are predominantly phenomenological. Thus, alternative models must be considered. Phenomenological models often require specific inputs, such as breakage functions in the milling and bubble size in the flotation, which are usually difficult to measure or not measured. Furthermore, the outputs of empirical models are susceptible to estimation errors, making them unreliable in guiding plant optimisation.

As a result, the following research questions have been identified:

- How does the incorporation of Merensky into the MF2 concentrator influence circuit performance?
- Can ML methods be employed to develop a daily predictive model for accurately forecasting ore mineralogy in the MF2 concentrator plant?
- Can ML techniques be utilised to create a data-based model for accurately predicting MF2 plant performance?

1.2. RESEARCH OBJECTIVES

Based on the findings in Section 1.1, the primary objectives of this research are:

- To analyse process data that describe the operational performance of an MF2 plant with and without incorporating Merensky ore.
- To develop a machine learning algorithm for determining the mineralogy of the ore based on the process data.
- To develop a machine learning algorithm for the prediction of plant performance.

2. LITERATURE REVIEW

Metal recovery from run-of-mine (ROM) ore involves conventional processing steps, including size reduction, concentration, smelting, and subsequent refining. Figure 2.1 illustrates the standard flowsheet for PGM processing.

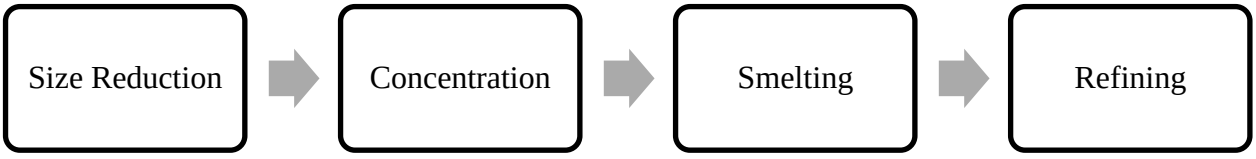


Figure 2.1: Typical PGM Processing Flowsheet

Mineral processing or ore dressing for PGMs in South Africa, as outlined by Danha et al. (2015), generally encompasses the following steps:

- Size reduction (comminution): This step involves reducing the size of the ore through comminution processes, thereby liberating the valuable minerals from the ore matrix.
- Concentration: The concentration process aims to separate the gangue and valuable minerals, increasing the purity and concentration of functional components.
- Smelting: Smelting involves chemical separation processes that further segregate mineral components, facilitating their extraction.
- Refining: Following smelting, the refining step purifies the valuable minerals, ensures their quality, and transforms them into marketable products.

A comprehensive understanding of ore mineralogy is crucial for designing, troubleshooting, and optimising process units before designing the flowsheet. This study focuses exclusively on the comminution and concentration units, as optimising these stages enhances gangue mineral rejection. This, in turn, leads to reduced energy demands in the subsequent smelting and refining stages (Evans et al., 2009).

2.1. Process Overview

2.1.1. Mineralogy

The BIC is a significant geological formation that hosts three primary platinum group element (PGE) reefs: Merensky, UG2, and Platreef. The BIC contributes 75% of the total output from these reefs to the world's platinum production (Corin et al., 2021).

Merensky ore comprises orthopyroxene, plagioclase feldspar, pyroxene, and phlogopite with occasional olivine (all are silicate minerals). Moreover, secondary minerals, which are talc, serpentine, chlorite, and magnetite, are also contained in the Merensky Reef (Corin et al., 2021). The main components of Merensky's Base Metal Sulphides are chalcopyrite, pyrrhotite, pentlandite, and traces of troilite, pyrite, cubanite, and millerite. The Merensky reef also contains cooperate (PtS), sperrylite (PtAs2), braggite (Pt, Pd) NiS) and different alloys that are the PGMs. The PGE grades average 3-8 g/t, and the PGM grain size is typically 150 µm (Corin et al., 2021; Dzvinamurungu et al., 2013).

On the other hand, the UG2 reef, a significant player in the mineral processing landscape, mainly comprises chromite with a lesser presence of silicate minerals. The average PGE grades for UG2 range between 4 and 7 g/t with grain sizes below 10 µm, making it a fascinating area of study (Corin et al., 2021; Dzvinamurungu et al., 2013; Hay & Roy, 2010). Larger grain sizes above 30 µm are rare in UG2 as they are typically found within silicate pockets, less prevalent in this reef. (Hay & Roy, 2010).

Like UG2, the Platreef also has fine-grained PGEs with rich pyroxenites, serpentinites and silicates (Corin et al., 2021). The Platreef ore contains a lower Pt: The Pd ratio is iron-rich (McDonald et al., 2005).

Table 2.1 summarises the average ore composition of the Merensky, Platreef, and UG2 varieties.

Table 2.1: Typical Bulk composition of PGM ores (Corin et al., 2021).

Mineral Name	Merensky	UG2	Platreef
	Volume %		
Pyroxene	55 – 60	15 – 30	30 – 40
Feldspar	30 – 40	3 – 9	18
Chromite	6	50 – 75	-
Talc	<1	<1	<1
Serpentine	2 – 3	1	5
Amphibole	1 – 2	<1	4
Chlorite	1 – 2	<1	4
Mica	<1	<1	1
BMS	<1	<1	2
Other (Silicates, Fe-oxides and carbonates)	1 – 2	<1	5

It is standard procedure to mix the high sulphide Merensky ore with the UG2 to minimise the $\%Cr_2O_3$ (Hay & Roy, 2010). This practice reduces milling time as the plagioclase content is reduced (Dzvinamurungu et al., 2013). However, this comes at the expense of chromium content in the concentrate, as UG2 has a higher chromium concentrate when compared to Merensky, as seen in Table 2.1.

2.1.2. Comminution

Comminution is the breaking of the ore into smaller fragments to facilitate the liberation of valuable minerals from the gangue. This process generates a mix of gangue, partially liberated, free and wholly encapsulated minerals (Wills & Finch, 2016a). The liberation of the valuable minerals from the gangue is essential for flotation as it makes it possible to efficiently extract and recover by separating these two minerals (Corin et al., 2021; Wills & Finch, 2016a). The optimal

-75 µm grind target for PGMs is 55–65% for Merensky ore and 70–80% for UG2 ore(Hay & Roy, 2010).

2.1.3. Flotation

Flotation involves a separation technique for selectively isolating valuable minerals from the gangue minerals. This procedure consists of the application of a collector to induce hydrophobicity on the surface of the valuable mineral (Brooks et al., 2019). In the flotation cell, an agitated slurry is aerated by introducing bubbles with a ~1 mm diameter. Agitation of the slurry maintains suspension of the fine solid particles while the heavier particles, due to their hydrophilic tendencies, sink to the bottom of the cell (King, 2001). Standard MF2 circuits consist of roughers and a dedicated series of cleaners that produce a high and low-grade final concentrate. These roughers are followed by a dedicated set of cleaners, which produce high-grade concentrate and recycle the low-grade into the primary roughers (Hay & Roy, 2010). This configuration is shown in Figure 2.2.

The chromium concentration in the final concentrate typically remains below 3%, with the highest recorded concentration at 3.5%. This limitation is imposed because elevated levels of chromium have adverse effects on matte temperatures during the conversion stage, leading to refractory lining damage (Kitagawa et al., 2018).

2.1.4. Mill-Float-Mill-Float Circuit

The MF2 circuit typically encompasses two comminution and rougher flotation stages, with the second comminution and flotation stage encompassing a re-grind and low-grade recovery. This configuration targets fine PGM (usually below 10 µm) while preventing over-grinding of the chromium and other gangue minerals, which leads to high chromium recovery and low PGM recovery (Corin et al., 2021). This is due to the UG2 ore's more complex mixture of fine-grained silicates and chromium (C. M. Rule, 2009). A generic MF2 PGM concentrator is shown in Figure 2.2.

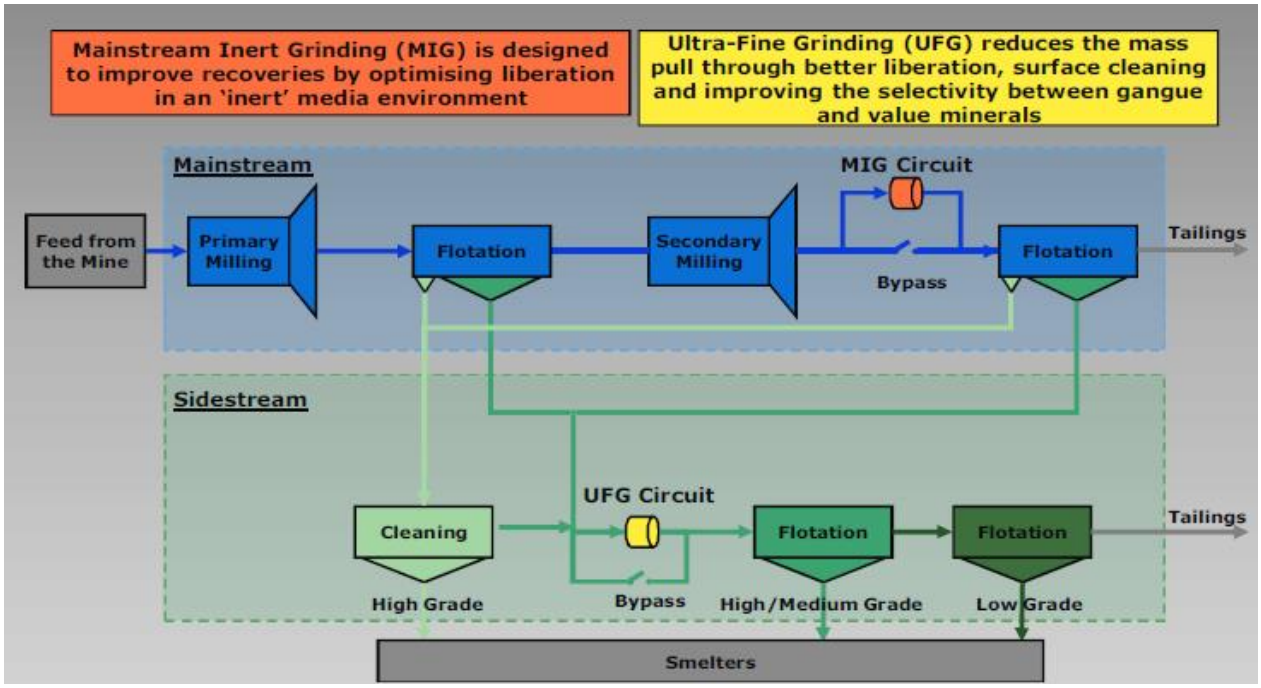


Figure 2.2: Generic MF2 PGM concentrator schematic (C. Rule, 2011)

The primary milling stage, which typically uses ball mills, semi-autogenous mills, and, less often, autogenous mills, is instrumental in ensuring the proper dosing of the reagent. This is crucial as insufficient dosing can lead to larger particles becoming hydrophilic, resulting in poor recoveries (C. Rule, 2011). Moreover, the sulphide minerals associated with PGMs may oxidise, rendering them hydrophilic (Corin et al., 2021). The hydrophilic material in the flotation stage plays a pivotal role in the MF2 circuit. This material is then sent to the regrind stage (secondary milling), a ball mill or IsaMill. The success of the regrind stage is primarily determined by the properties of this material (C. Rule, 2011).

The MF2 circuit plays a crucial role as it significantly reduces the propensity of over-grinding, thereby enhancing the recovery process (Corin et al., 2021). Over-grinding leads to the fine-grained gangue absorbing the reagent and being recovered in the concentrate (C. Rule, 2011).

2.2. Model Development

Model development uses an iterative methodology to test the derived model until it meets the specified criteria. This iterative methodology allows for the tuning and optimisation of parameters to ensure that the model aligns with the desired performance and accuracy benchmarks (McGahey & Cameron, 2002). There are two modelling approaches: phenomenological and data-based. Phenomenological models demonstrate superior extrapolation capabilities relative to data-based models; however, their development can be cost-intensive due to limited information on the governing process phenomena. In contrast, data-based models are more cost-effective but necessitate extensive datasets (Neumann et al., 2006).

Model development involves sequential steps, such as data collection, pre-processing, evaluation, modelling, and validation. This study will encompass data evaluation, modelling, and model validation. The omission of the other steps is based on their well-defined and understood nature, which renders them less susceptible to scrutiny.

2.2.1. Phenomenological Modelling

Phenomenological models are mathematical models that describe the relationship between empirically/experimentally observed relationships without considering underlying mechanisms (Huffaker et al., 2018). Mathematical model development is crucial for process analysis as it details the dynamic characteristics and steady-state precisely (Hodouin et al., 2001). As outlined by Wills and Finch (2016 b), using mathematical modelling tools substantially reduces costs related to trial-and-error when optimising plant performance (Wills & Finch, 2016b). Commonly used phenomenological models for comminution and flotation are energy-based and kinetic models. Although phenomenological models are discussed in this paper, they are not considered for this research due to the nature of the available data, which does not directly correspond to the input requirements of phenomenological models.

Population Balance Model

A population balance model (PBM) is an integral-partial differential equation that describes the changes in systems by measuring extensive properties like mass, size and/or molecule count. These models describe the changes in the number density function (Totis et al., 2021). PBMs provide a comprehensive approach to model particulate milling and flotation processes. Using flexible methods such as Monte Carlo simulations, discretisation, or the moment method, PBM adapts to complex systems, offering enhanced accuracy, especially when dealing with heterogeneous materials or varying particle properties (Bagchi, 2023)

Despite the advantages of PBMs, it has limitations, particularly in scale-up applications. For example, kinetic parameters derived from lab results may underestimate the PSD in larger batches due to scale-dependent factors like secondary nucleation from particle-impeller collisions (Rosenbaum et al., 2019). Additionally, most PBMs assume perfect mixing, which leads to unreliable PSD predictions. Furthermore, most PBMs use a one-dimensional approach, which struggles with complex morphologies (Rosenbaum et al., 2019).

Kinetic Models

Kinetic models are essential for modelling flotation processes as they describe the rate at which valuable particles are recovered in the froth phase. The classical first-order kinetic model:

$$f(k) = e^{-kt}$$

Where:

k = rate constant

It assumes that the flotation rate is constant across the particles, leading to a simple exponential recovery equation. This assumption, however, often fails for complex ores with varying particle properties. Modified kinetic models address this by incorporating distribution functions for the

flotation rate constant, offering a more accurate representation of the process (Polat & Chander, 2000).

The assumption of constant flotation rate in phenomenological modelling is not valid, as this parameter is influenced by changes in PSD and is difficult to measure. Given that the flotation rate varies with PSD, the assumption of constancy does not hold. Consequently, due to the challenges in measuring flotation rate and the variability associated with PSD (Polat & Chander, 2000).

2.2.2. Data-based Modelling

Data-based modelling methods are often employed on "soft sensors" that typically predict challenging-to-measure process parameters (McCoy & Auret, 2019). ML algorithms are commonly used data-based models as they can handle data and learn trends within the data to make predictions (Mahesh, 2018). Based on the literature, supervised models are commonly used.

Supervised learning or modelling involves the prediction of outcomes for future data points based on the relationship of the provided set of input-output pairs within a given dataset, if only the inputs are provided (Poole & Mackworth, 2017). Each algorithm has a distinct data processing framework that allows for the easy completion of complex predictions in real time (Mutinda & Geletu, 2025). Widely used supervised ML algorithms include linear, tree-based and support vector machines (SVM).

Meanwhile, a time series consists of data points collected at different intervals, making it a key tool for tracking patterns over time. This type of analysis is particularly valuable in finance, where analysts, investors, and policymakers use it to interpret and forecast market movements (Mutinda & Geletu, 2025).

Linear Algorithms (Ordinary Least Squares)

Linear models predict a real-valued function from numeric input-output features within a training dataset (Poole & Mackworth, 2017). It relates to the output (or response) y_t to the input x variables based on the equation:

$$y_t = a + b_1 x_{t1} + \cdots + b_q x_{tq} + \epsilon_t$$

Where a is the intercept, b is the slope of the best-fit line, and ϵ_t is the error term, the difference between the predicted and observed terms. Linear regression (LR) assumes that:

- The relationship between the input and output is linear
- There is little to no multicollinearity
- Multivariate normality
- No autocorrelation
- Homoscedasticity

Linear models are characterised by their simplicity in implementation and interpretation. They require relatively low computational resources, making them efficient regarding training time and execution (Anandhi & Nathiya, 2023). When the assumptions of linearity, independence, and normality are satisfied, linear models can yield accurate predictions. The coefficients in the model provide a direct quantification of the relationship between independent variables and the response, which facilitates the interpretation of feature effects (Anandhi & Nathiya, 2023).

However, linear models are limited by their assumption of a linear relationship between predictors and the target variable, which restricts their applicability in modelling nonlinear patterns. They are sensitive to outliers and multicollinearity, which can distort parameter estimates and reduce model stability (Anandhi & Nathiya, 2023). The validity of linear models also depends on statistical assumptions such as normally distributed residuals and homoscedasticity. Additionally, their performance can degrade when applied to datasets with high noise levels or interdependent predictors (Anandhi & Nathiya, 2023).

Tree-Based Algorithms

Tree-based algorithms employ tree diagrams with nodes and branches representing possible outcomes. The node represents an event, and the branches represent a condition. Each note represents the attribute within a group to be classified, while each branch represents the probability that the node can assume (Mahesh, 2018).

Tree-based models possess several technical advantages. They offer rapid implementation and operate based on explicit decision rules, facilitating logical flow in model execution. These models demonstrate high predictive accuracy, particularly when employed in ensemble configurations. They manage datasets containing noise or missing values without requiring extensive preprocessing. Furthermore, tree-based methods apply to classification and regression problems, enhancing flexibility across different modelling tasks (Jadhav & Channe, 2013; Montgomery & Olivella, 2018).

Nonetheless, there are inherent limitations. Individual decision trees are susceptible to overfitting, especially without pruning or regularisation. As trees grow deeper, they may become increasingly complex and redundant, affecting model efficiency. Although ensemble methods such as Random Forests (RF) can improve predictive performance, they often reduce model interpretability due to the aggregated nature of multiple trees, making it more difficult to extract transparent decision-making processes (Jadhav & Channe, 2013; Montgomery & Olivella, 2018).

RF is an example of a tree-based model. RF incorporates randomisation to generate various decision trees and aggregates their outputs through a voting mechanism for regression and classification problems (Rigatti, 2017). The implementation diagram, Figure 2.3, shows the tree-based algorithm, where each node represents a decision point and branches represent conditions leading to outcomes.

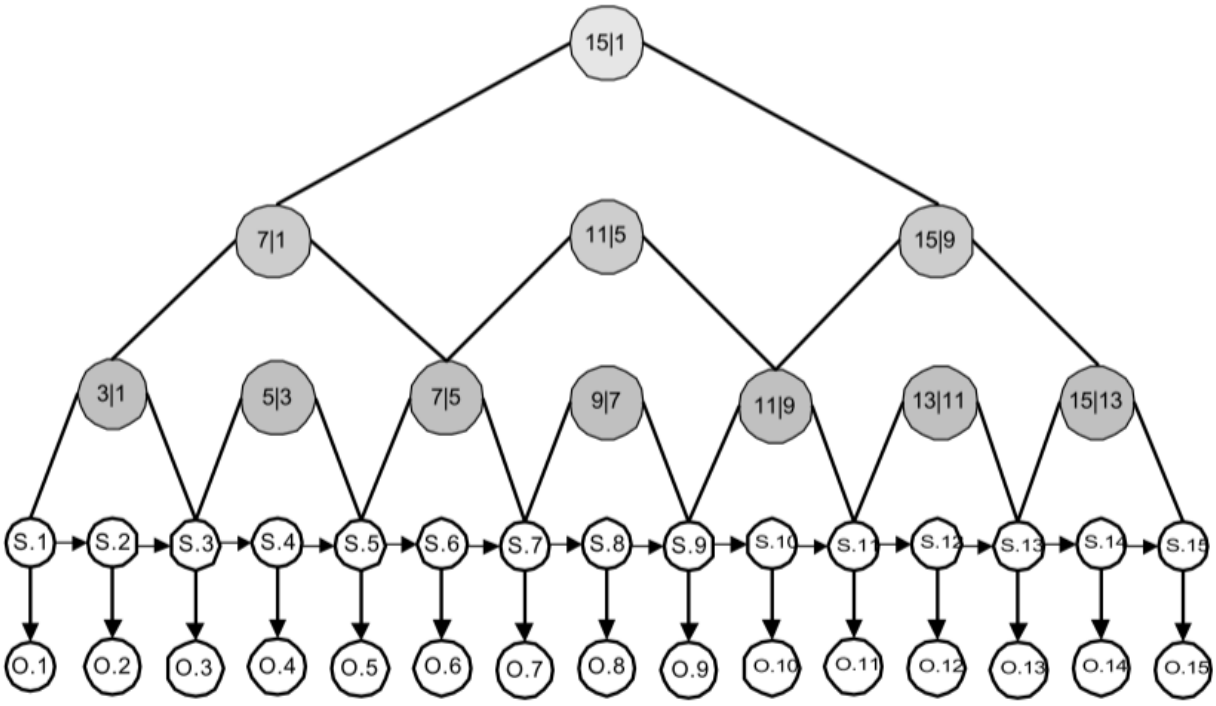


Figure 2.3: The implementation of the tree-based algorithm (Jearanaitanakij, 1999)

Support Vector Algorithms

SVM are ML algorithms in which the regression problems are defined as convex optimisation problems, precisely quadratic problems. Characteristics of SVM for the regression are loss function, optimisation, kernel trick and sparseness (Wang & Hu, 2005). SVM maximises classification boundaries and hyperplanes within its design philosophy for classification problems and their computational complexity (Wang & Hu, 2005). Figure 2.4 from Zou et al. (2021) illustrates the optimal hyperplane and support vectors, clearly demonstrating the margin maximisation principle. The advantages and disadvantages of SVMs adopted from Bin Sulaiman (2014) are:

Advantages:

- Performs well on both linear and nonlinear classification tasks.
- Strong theoretical foundation ensures a unique optimal solution.
- Suitable for high-dimensional data.
- Effective with appropriate kernel selection.

Disadvantages:

- Computationally demanding for large datasets.
- Difficult to interpret and tune.
- Designed for binary classification; multi-class requires extensions.

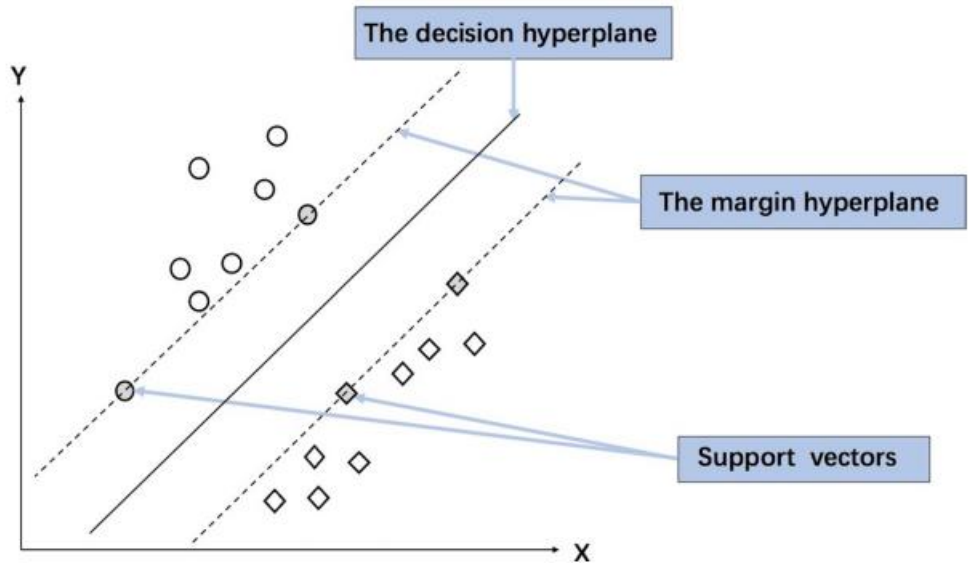


Figure 2.4: Schematic diagram of an SVM (Zou et al., 2021)

K-Nearest Neighbours Algorithms

The K-Nearest Neighbours (kNN) algorithm is a simple supervised learning ML algorithm commonly used for classification tasks. The algorithm operates on the principle of proximity, where a new data point is classified based on the majority class among its k closest neighbours. It involves calculating the distance (often Euclidean) between the new and existing data points, then assigning the new data point to the class most common among the nearest neighbours (Pandey & Jain, 2017; Suyal & Goyal, 2022). The algorithm can employ different metrics, such as Manhattan or Chebyshev, depending on the type of data and problem context (Tarakci & Ozkan, 2021). Pandey and Jain (2017) state that the lack of a training phase makes the kNN computationally efficient (Pandey & Jain, 2017). The advantages and limitations adopted from Jadhav and Channe (2014) of kNN MLs are:

Advantages:

- Simple to understand and easy to implement.
- No explicit training phase.
- Effective in low-dimensional datasets.
- Adaptable to different distance metrics.

Disadvantages:

- Sensitive to noisy and irrelevant features.
- Memory-intensive.
- Performance drops in high-dimensional spaces.

Figure 2.5 from Zhang (2017) illustrates a kNN scenario showing a test sample surrounded by its five closest neighbours and the resulting classification based on majority voting.

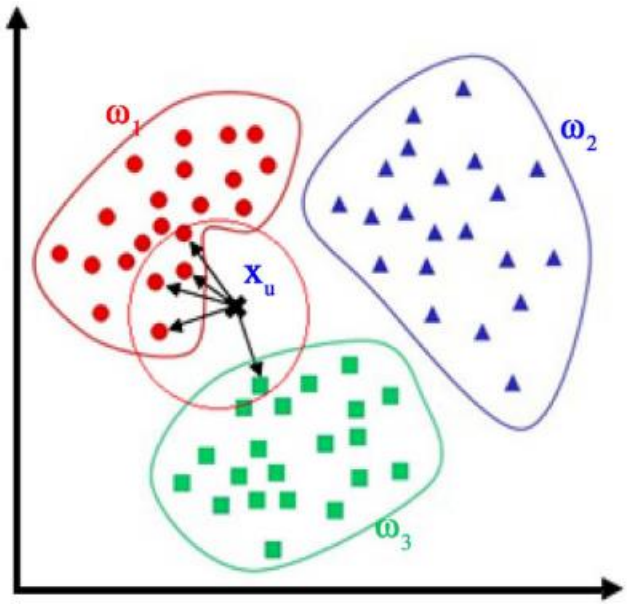


Figure 2.5: kNN model with k = 5 (W. Zhang, 2017)

Neural Network Algorithms

An artificial neural network (ANN) is an ML algorithm inspired by biological neural networks (Choi et al., 2020). Each node (analogous to cell bodies) in the ANN communicates with other nodes via connections (analogous to axons and dendrites). Connections between nodes in an ANN are weighted based on their ability to provide a desired outcome, much like synapses in biological networks strengthen with correlated outputs (Choi et al., 2020). Typically, in an ANN framework, there is a layer of input nodes, a layer of output nodes, and one or more hidden layers between the two, as shown in Figure 2.6.

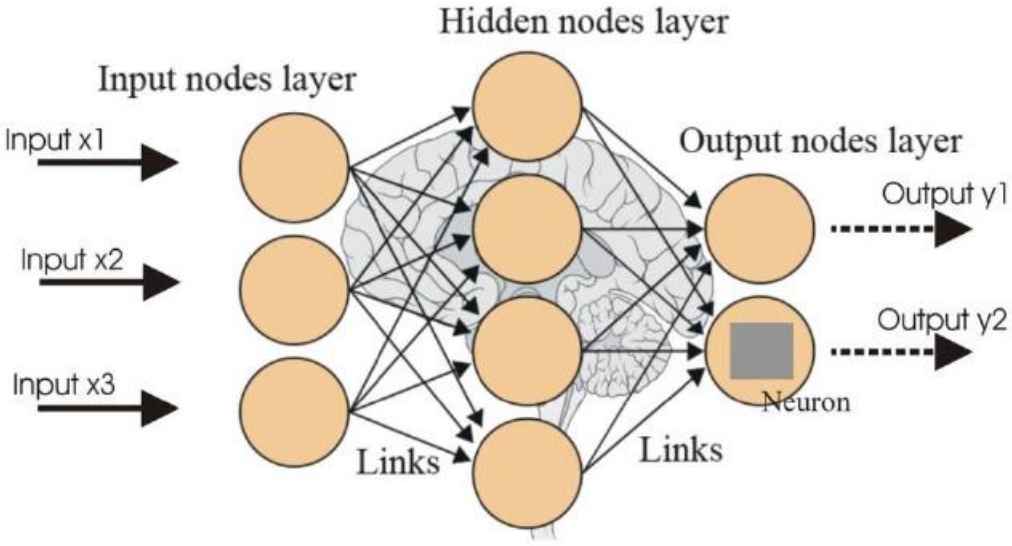


Figure 2.6: Neural Network learning process (M. Mijwil, 2021).

A common type of ANN is the feedforward neural network. It passes information from one layer to another without looping back (Choi et al., 2020). Deep learning, a subfield of ANNs, uses multi-layered networks to process information, with each layer learning progressively more complex functions. These algorithms are effective in tasks like image and speech recognition. They typically achieve a higher accuracy than traditional ANNs while requiring more computational resources (Di Franco & Santurro, 2021).

ANNs offer several advantages in computational modelling. One key strength is their ability to store information across the entire network rather than in a central database, meaning the loss of some information in one part of the network does not prevent it from functioning. ANNs can also operate with incomplete knowledge; after training, they are capable of producing outputs even when some input data is missing, with the degree of performance loss depending on the importance of the missing information. Their fault tolerance further enhances their robustness, as the corruption of one or more nodes does not stop the network from generating outputs (M. Mijwil, 2021). Additionally, ANNs possess distributed memory, requiring selected examples for training, and their success depends heavily on the quality and coverage of these examples; incomplete

training data may lead to inaccurate outputs. ANNs degrade gradually rather than failing abruptly, showing a slow decline in performance over time. They are also capable of machine learning, as they learn from events and make decisions by drawing parallels to similar past occurrences. Lastly, their parallel processing capability allows ANNs to handle multiple tasks simultaneously, contributing to their computational efficiency.

Despite these advantages, ANNs have notable limitations. They are heavily dependent on hardware, requiring processors with parallel processing capabilities to align with their structure. A major drawback is their "black-box" nature, as ANNs often produce solutions without offering explanations for how decisions were made, which undermines user trust. Determining the proper network structure is another challenge, as there is no definitive rule for configuring ANNs — the optimal structure is typically found through trial and error and experience (M. Mijwil, 2021). Furthermore, ANNs require numerical data, so problems must be translated into numerical values before being processed, and the method of representation directly impacts the network's performance, relying on the user's skill. Lastly, the training duration of an ANN is unpredictable; training ends when the network's error reaches a certain threshold, but this does not guarantee an optimal solution, adding further uncertainty to the modelling process (M. Mijwil, 2021).

Autoregressive Integrated Moving Average

The Autoregressive Integrated Moving Average (ARIMA) model is a widely used classical statistical approach for time series forecasting. It integrates three fundamental components: autoregression (AR), differencing (I), and moving average (MA), making it particularly effective for modelling time-dependent data with trends and seasonality (Mutinda & Geletu, 2025). The general ARIMA(p, d, q) model is mathematically expressed as:

$$y_t = \phi_1 y_{t-1} + \phi_1 y_{t-2} + \cdots + \theta_q \epsilon_{t-q} + \epsilon_t$$

Where y_t represents the observed value at time *t*. The parameters ϕ_1 correspond to the autoregressive (AR) coefficients, which capture the influence of past values on the current

observation. Similarly, θ_1 is the moving average (MA) coefficient, representing the effect of past error terms on the present value. The term ϵ_t denotes the error component (white noise), accounting for random fluctuations in the data. The values of p and q define the orders of the AR and MA components, respectively, while d specifies the number of differencing operations required to transform the data into a stationary series, ensuring stable statistical properties for forecasting (Mutinda & Geletu, 2025).

ARIMA models offer several advantages, including high accuracy and robustness in financial forecasting, particularly in capturing trends and seasonality in crude oil prices. The structured Box-Jenkins methodology ensures a rigorous approach to time series modelling, and ARIMA is versatile and applicable to various economic indicators. It also effectively handles time dependencies through autoregressive and moving average components (Hosseini et al., 2025).

However, ARIMA has disadvantages, including computational complexity due to extensive parameter selection, sensitivity to data quality, and the assumption of stationarity, which can be challenging with non-stationary or shifting data. It struggles with sudden market shocks and is more suited for short-term forecasting (Hosseini et al., 2025).

Partial Least Squares

The partial least squares (PLS) is an algorithm developed to linearise models which were non-linear in the parameters (Barker & Rayens, 2003). PLS is a component-based Structural Equation Modelling (SEM) method primarily used for predictive purposes. Unlike covariance-based SEM (CovSEM), which relies on distributional assumptions, PLS is a distribution-free approach, making it more flexible with different types of data. It operates through a two-step process: first, it estimates path relationships in the outer (measurement) model to calculate latent variable scores, and second, it uses Ordinary Least Squares regressions on these scores to estimate the structural model. The main objective of PLS is to maximise the explained variance by predicting latent variables from dependent variables, rather than focusing on achieving a good fit to the data, which is the goal of CovSEM. PLS is advantageous in that it requires smaller sample sizes, can handle a

larger number of variables, and uses simpler algorithms. Additionally, it is better suited for exploratory research, particularly in cases involving formative constructs, and offers more practical insights into the formation of latent variables. Overall, PLS is focused on prediction, while CovSEM aims for parameter estimation and data fit (Barker & Rayens, 2003).

PLS-SEM offers several advantages, particularly for prediction-oriented research and complex models. It accommodates small sample sizes, non-normal data distributions, and formative and reflective measurement models. Unlike CB-SEM, it explicitly estimates latent variable scores and optimises explained variance, making it suitable for exploratory studies and models with high dimensionality (Sarstedt et al., 2014).

However, PLS-SEM has limitations. It provides inconsistent estimates for factor models, making it less suitable for confirmatory research. Its focus on variance explanation can introduce bias, especially in small samples. Additionally, the absence of universally accepted global goodness-of-fit measures complicates model validation, reducing its efficacy for strict hypothesis testing (Sarstedt et al., 2014).

Principal Component Analysis

Principal Component Analysis (PCA) is a linear dimensionality reduction technique used to transform a dataset with multiple variables into a smaller set of uncorrelated components, known as principal components. These components are ordered by the amount of variance they capture, with the first principal component explaining the most variance and each subsequent component capturing progressively less. PCA is particularly useful for simplifying complex datasets, reducing noise, and identifying underlying patterns by projecting data onto a lower-dimensional space (Demšar et al., 2013).

PCA offers several advantages. It effectively reduces dimensionality, retaining most of the dataset's variance, which enhances the efficiency of statistical models by eliminating redundancy caused by correlated variables. This data orthogonalisation ensures that the transformed components are uncorrelated, making it easier to apply further analyses like regression or

clustering. Additionally, PCA aids in visualising high-dimensional data by revealing the directions along which data vary the most, uncovering hidden structures and relationships within the dataset (Demšar et al., 2013).

However, PCA also has limitations. One major drawback is the loss of interpretability, as principal components are linear combinations of the original variables, making them less intuitive and harder to explain. Furthermore, PCA assumes linear relationships between variables, limiting its ability to capture complex, nonlinear data structures. The results are also sensitive to scaling, meaning that using either the covariance or correlation matrix can affect the outcome, requiring careful preprocessing, such as standardising variables with differing units. Lastly, PCA is highly susceptible to outliers, which can distort the principal components and misrepresent the underlying data patterns, necessitating robust alternatives when working with noisy datasets (Demšar et al., 2013).

2.2.3. Model Validation

Model validation is the systematic assessment of the performance (accuracy) of the developed model to fulfil its desired criteria. In engineering, the graphical comparison between the measured values and model-predicted values is a common practice (Ling & Mahadevan, 2013). Ling and Mahadevan (2013) introduce hypothesis testing-based methods, which include classical, Bayesian interval, and Bayesian equality hypothesis testing and non-hypothesis testing-based techniques, which include reliability-based and area metric-based methods (Ling & Mahadevan, 2013).

The application of Bayesian equality hypothesis testing is confined to models where both input and output are stochastic (Ling & Mahadevan, 2013). Stochastic models account for random variance in one or more parameters over time, predicting potential output parameters (Rajasekar et al., 2022).

Hypothesis Testing Method

- Classical – involves testing the null hypothesis against the alternative hypothesis. The test statistic is derived from validation data, and the corresponding p-value (the probability of the test statistic matching the observed value) is computed (Costello & Watts, 2022).
- Bayesian – derives from Bayes' theorem, establishing the relationship between two conditional probabilities. It provides the probability of observing the data given the hypothesis (Ling & Mahadevan, 2013).

Non-Hypothesis Testing Method

- Reliability-based – evaluates the probability of observed data and model prediction differing within a specified tolerance limit (Ling & Mahadevan, 2013).
- Area Metric-based – this method measures the difference between the cumulative distribution function of the measured data against the predicted output (Ling & Mahadevan, 2013).

2.2.4. Data Limitations in Machine Learning

Machine learning performance is constrained by data quality and diversity. Small datasets, particularly in specialised fields like action recognition, hinder generalisation, while limited participant diversity restricts applicability to broader populations. Many datasets rely on scripted actions, failing to capture real-world variability. Ethical and privacy constraints further limit data availability, particularly in sensitive applications like fall detection and violence recognition (Mastorakis, 2018).

Technical challenges such as occlusions, sensor inconsistencies, and annotation errors degrade model reliability, as many datasets assume ideal recording conditions that rarely reflect real-world environments. Additionally, one-size-fits-all approaches struggle to accommodate diverse physical characteristics and movement patterns. Consequently, models trained on biased or incomplete data exhibit poor adaptability and reduced real-world performance (Mastorakis, 2018).

2.2.5. Data Partitioning

Data splitting, also known as train-test partitioning, is a crucial process in machine learning that involves dividing a dataset into separate subsets for training, validation, and testing. This technique ensures that models are trained on one portion of the data and evaluated on another, preventing overfitting and improving generalisability (Muraina, 2022). The choice of split ratios, such as 70/30, 80/20, or 90/10, significantly impacts model performance, as an optimal balance between training and test data enhances predictive accuracy (Nguyen et al., 2021).

The impact of data splitting on model performance is substantial. If the training set is too small, the model may not learn patterns effectively, leading to underfitting. Conversely, an excessively large training set may reduce the test set size, limiting the ability to evaluate generalisation performance (Muraina, 2022). Studies suggest that increasing the training data size up to 80% improves both training and test accuracy, but beyond this threshold, performance gains may diminish (Nguyen et al., 2021).

The advantages of data splitting include improved model generalisation, reduced bias, and prevention of overfitting. Additionally, proper partitioning helps optimise hyperparameters and ensures robust evaluation of machine learning models (Muraina, 2022). However, it also has drawbacks, such as the risk of suboptimal split ratios leading to poor model performance and the assumption that data distributions in training and test sets are representative of real-world scenarios. Moreover, computational efficiency can be affected when experimenting with multiple split configurations (Muraina, 2022).

2.2.6. Detection of False Data

Intrusion detection is commonly used in network security to prevent security breaches. However, intrusion detection systems are rule-based systems (J. Zhang et al., 2008). For automatic generation control systems, outlier detection identifies instances in the dataset that do not conform to normal activities (Qu et al., 2023).

Qu et al. (2023) outlier detection method

The RF algorithm can integrate and process different types of original data. Qu et al. (2023) propose an RF to detect and evaluate new data. The RF is an ensemble of decision trees that conglomerate their outputs. Each tree in the forest is trained on different subsets of the data, and the results are combined by voting to improve detection accuracy (Qu et al., 2023). The RF algorithm identifies patterns in the data and detects deviations in the new data, which can be a possible attack. The framework constructs multiple decision trees from the training data. During detection, the RF model votes on the data to decide if an attack is occurring. The probability of an attack is determined based on the proportion of trees that agree on a specific outcome (Qu et al., 2023).

2.2.7. Overfitting

Overfitting occurs when an ML algorithm learns patterns and trends specific to the training data and fails to generalise to the unseen test data. This phenomenon is typically caused by the model capturing noise within the training data (Mutasa et al., 2020). Overfitting results in a decline in performance on new and unseen data. One indicator of overfitting is a significant discrepancy between the c-statistics of the training and test datasets (Barreñada et al., 2024).

2.2.8. Plant Optimisation

Figure 2.4 shows the recommended procedure for plant optimisation. This systematic approach facilitates efficient and cost-effective plant optimisation, leveraging mathematical models for informed decision-making and minimising reliance on resource-intensive trial-and-error methodologies (Wills & Finch, 2016b).

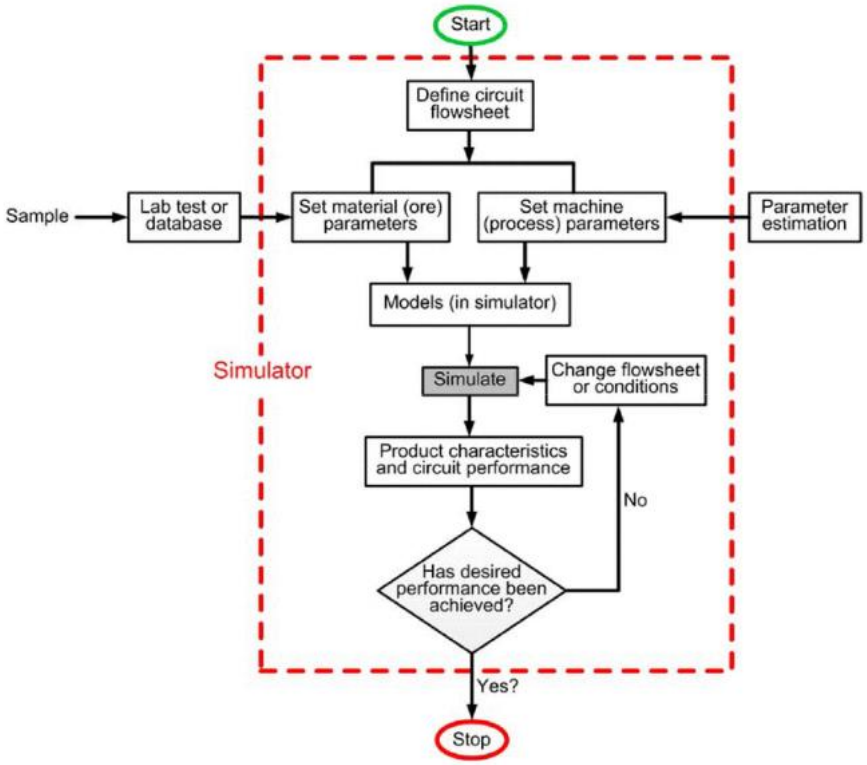


Figure 2.7: Procedure for plant optimisation (From B. A. Wills & Finch, 2016a)

2.2.9. Hyperparameter Tuning

Hyperparameter tuning (HPT) is the fine-tuning of ML models. The grid and random search are the two commonly used methods.

Grid Search

Grid search approach exhaustively searches over a predefined subset of hyperparameters. The ML are trained and evaluated for each possible combination of these parameters. The grid search follows a systematic approach, ensuring all combinations are tested. Based on this, the method is computationally expensive, especially when dealing with high-dimensional spaces where the number of combinations can grow exponentially (Belete & Huchaiah, 2022; Liashchynskyi & Liashchynskyi, 2019). However, the grid search has limitations regarding continuous

hyperparameter spaces. This is because it requires discretisation of the range of values, which may lead to suboptimal results if the actual optimal values lie between the selected grid points (Liashchynskyi & Liashchynskyi, 2019).

Random Search

The random search randomly samples hyperparameter values from the defined ranges instead of evaluating all possible combinations. Unlike the grid search, the random search is helpful in high-dimensional spaces where only a few hyperparameters significantly impact model performance (Liashchynskyi & Liashchynskyi, 2019). Moreover, the random search can handle both discrete and continuous hyperparameter spaces effectively and can outperform the grid search when computational resources are limited (Liashchynskyi & Liashchynskyi, 2019).

2.2.10. Geometallurgy

Optimising run-of-mine (ROM) ore is a strategic priority in mining operations. This optimisation requires precise adjustment of process parameters to maximise both valuable mineral recovery and equipment efficiency. Key parameters include average grain size, textural characteristics, and the chemical composition of target minerals, with grain size and texture playing particularly critical roles (Kalashnikov et al., 2021; Mwanga et al., 2015).

In geometallurgical modelling, Lishchuk (2016) categorises three distinct approaches: traditional, proxy, and mineralogical. The traditional approach relies on metallurgical testing and statistical analysis to correlate ore feed characteristics (often chemical assays) with metallurgical outcomes. The proxy approach uses a small set of metallurgical response tests to infer ore behaviour indirectly, typically through indicators like hardness or magnetic susceptibility. In contrast, the mineralogical approach builds a model using quantitative mineralogical data — including modal composition, grain size distributions, and liberation characteristics — to predict processing performance. As demonstrated by Lamberg et al. (2013), this method enables particle-based modelling at varying levels of complexity (1D, 2D, and 3D), offering more detailed forecasts by

explicitly incorporating mineral textures and liberation behaviour into process simulations(Lamberg et al., 2013; Lishchuk et al., 2016).

Lamberg et al. (2013) model

Lamberg et al. (2013) present a mineralogical approach to geometallurgical modelling that integrates geological and metallurgical data into a predictive, spatial framework for production planning and management. The model comprises three interconnected components: the geological model, which provides modal mineralogical compositions and textural classifications; the process model, which translates this mineralogical information into forecasts of metallurgical behaviour; and the production model, which simulates operational scenarios. Each sample is linked to an archetype, defined by its textural type, which contains a library of particle types with characteristic size and liberation profiles. The process model operates at three levels of complexity: 1D (unsized mineral data), 2D (mineral-by-size), and 3D (mineral-by-size-by-liberation), with the 3D model providing the most detailed representation by explicitly accounting for liberation behaviour. This particle-based modelling approach allows the prediction of processing performance based on mineralogy and texture, supporting more robust production planning.

2.2.11. Comminution Modelling

Comminution process modelling tools are generally used to evaluate real comminution. Alternatively, they are used as a benchmark for process operation by determining process parameters such as throughputs and residence times that will produce the optimal particle size distribution (PSD) (Danha et al., 2015; Fuerstenau et al., 2004). PSD is a density function that quantifies the relative particle count organised by size (Switzer, 2013).

Danha et al., 2015 Comminution model

Danha et al. (2015) propose a batch phenomenological comminution model that classifies the PSD into three categories: feed, intermediate, and fine. The feed class represents coarse particles, the intermediate class corresponds to the optimal size for flotation, and the fine class encompasses

particles too small for adequate flotation. This categorisation enables the independent determination of the ideal PSD regardless of the feedstock.

This classification scheme facilitates the selection of the optimal PSD for flotation, ensuring efficient separation. The model is a function of selection (Si), residence time (t), and breakage function (bij). The selection represents the empirically determined breakage rate for each class size, while breakage accounts for the distribution of primary product particles broken from class j and transitioning to size class i. The breakage of a lone particle will produce progeny sizes of an entire spectrum (King, 2001).

Tangsathitkulchai (2002) advocates successfully applying reactor design principles to mill circuit simulation and design. The model Danha et al. (2015) introduced employs a modified Arrhenius equation for reaction kinetics. Here, the reaction rate (k) is substituted with the mass fraction of the flotation-defined ideal particle size category concerning the feed class of the PSD. This categorisation allows for conceptualising the size classes as reaction products or rates. The model demonstrates an exponential alteration in mass fraction with residence time, as shown below by Danha et al. (2015):

$$m_j(t) = \varphi\,[m_j(0)\exp^{-S_{jA}t} + \frac{b_{ijA}S_{iA}m_i(0)}{S_{jA}-S_{iA}}[\exp^{-S_{iA}t} - \exp^{-S_{jA}t}]] + (1$$

$$-\varphi)[m_j(0)\exp^{-S_{jA}t} + \frac{b_{ijB}S_{iB}m_i(0)}{S_{jB}-S_{iB}}\,[\exp^{-S_{iB}t} - \exp^{-S_{jB}t}]]$$

Where:

i, j = Feed and product size classes, respectively.

A, B = Soft and hard components, respectively.

$m_i(t)$, $m_j(t)$ = Mass fraction of class i or j material at time *t*.

φ = Fraction of "soft" material (Mohs scale based).

S_{iA}, S_{jA} = Breakage rates for soft material in classes *i* and *j*.

S_{iB}, S_{jB} = Breakage rates for hard material in classes *i* and *j*.

b_{ijA}, b_{ijB} = Breakage distribution functions (fraction from i→ji→j) for soft and hard components.

t = residence time.

2.2.12. Flotation Modelling

King (2001) states that while flotation principles are well established, developing quantitative predictive models is quite challenging due to the intricate nature of the processes influencing the overall flotation process (King, 2001).

Yianatos et al. (2020) Flotation Model

Yianatos et al. (2020) introduced a data-based flotation model, referred to as the single-cell model, which improves the representation of flotation cell performance by delineating two distinct zones within the cell: the pulp zone and the froth zone. This two-zone structure departs from traditional models that often consider the cell as a single, homogeneously mixed unit. By explicitly modelling these zones, the single-cell model enables independent estimation of pulp recovery and froth recovery, which are critical for accurately predicting metallurgical outcomes.

To construct a comprehensive mass balance, the model incorporates additional flow streams—specifically, entrainment, drop-back, bubble load, and top-of-froth grade. These streams capture material transport mechanisms often neglected in conventional models, thus enhancing the model's capacity to reflect real operational conditions. The mass balance framework is illustrated in Figure 2.8, which forms the basis for parameter estimation and model validation.

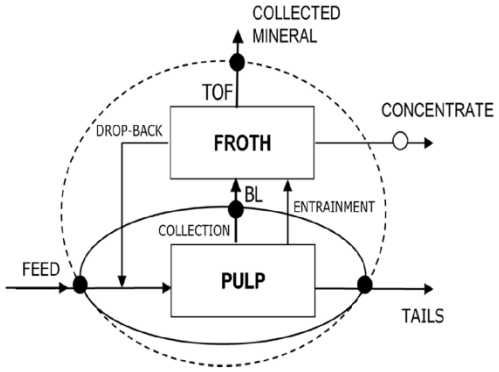


Figure 2.8: Froth Zone Mass Balance via Bubble Load (Yianatos et al., 2020)

Furthermore, Yianatos (2020) emphasises the importance of several operational variables for evaluating and optimising copper recovery in flotation circuits. These include:

- Solids flow rate;
- Slurry solids concentration (%);
- Feed mineralogy and liberation characteristics;
- Froth depth in each cell;
- Total cell volume;
- Froth cross-sectional area;
- Internal launder configuration;
- Concentrate discharge length.

By systematically varying these parameters, the model generates copper recovery profiles categorised into Fine, Intermediate, and Coarse particle size classes. These profiles, shown in Figure 2.9 allow for detailed performance assessment and operational tuning across different particle sizes, ultimately supporting the development of more responsive and efficient flotation strategies.

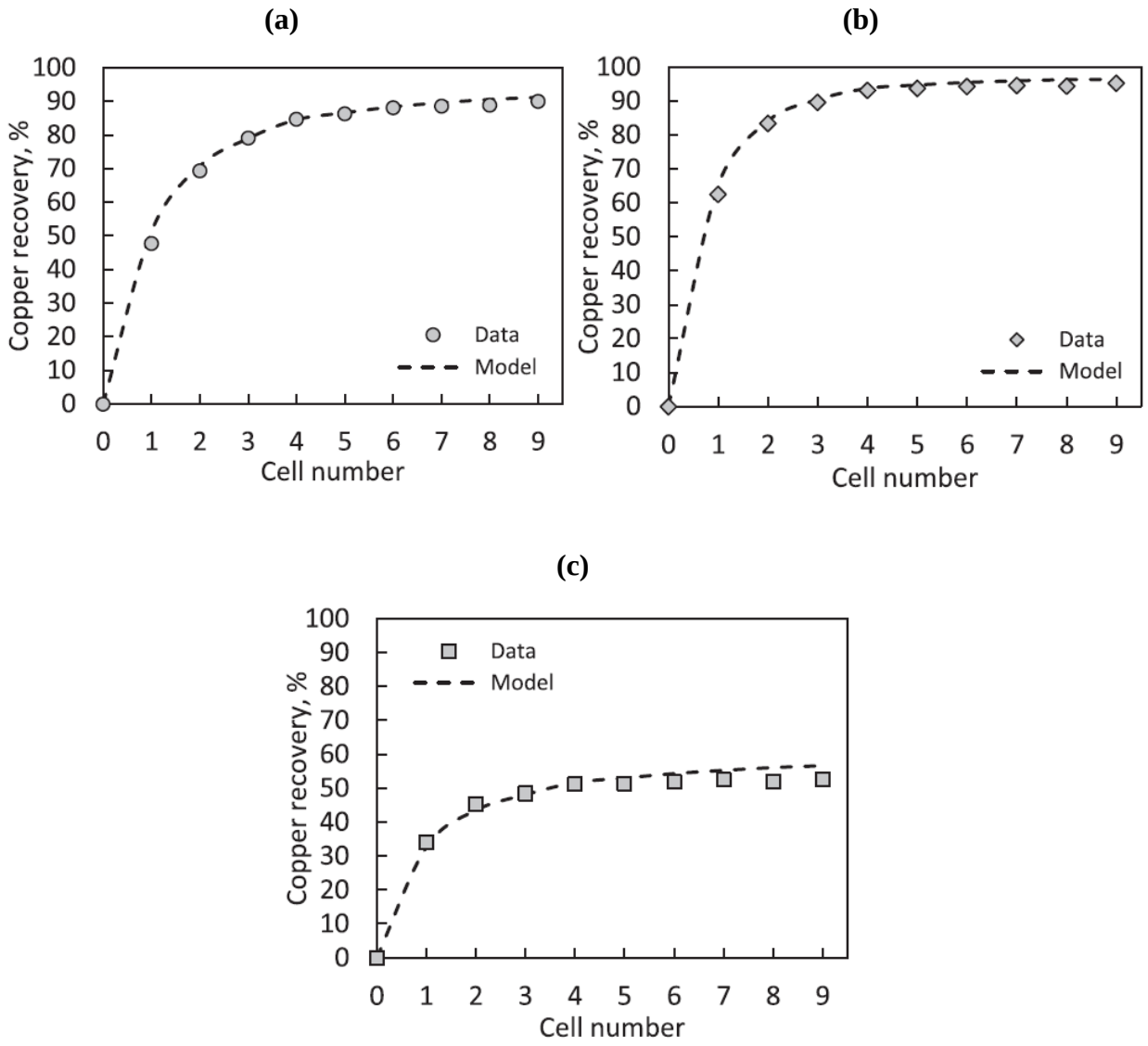


Figure 2.9: Copper recovery profiles in rougher flotation

Osei (2019) Flotation Model

Osei (2019) developed data-driven models: Multiple Linear Regression (MLR), RF, and Artificial Neural Network (ANN) to predict the percentage of silica concentrate in iron ore froth flotation in real-time, using a Kaggle dataset. Key input features included pulp pH, amina flow rate, and ore pulp density. Although feature selection methods such as Random Forest ranked the aeration rate

(specifically, Flotation.Column.02.Air.Flow) as insignificant, it was still incorporated during the model development phase for completeness.

Interestingly, the feature engineering process revealed that conventional physical parameters such as particle size distribution (PSD), as well as bubble shape and size, were not considered significant or were omitted entirely. The exclusion of PSD, however, limits the model's representational power, as PSD plays a critical role in mineral liberation and can provide valuable insights for optimising milling parameters and, consequently, flotation performance (Brooks et al., 2019).

The models were evaluated using RMSE as a performance metric. The RMSE for the Multiple Linear Regression model was 1.03, for the ANN model it was 0.55, and for the Random Forest model, 0.91. Among the three, the ANN demonstrated the best predictive performance, indicating its superior capability to generalise in non-linear systems such as froth flotation..

2.3. Sci-kit Learn

Sci-kit learn (Sklearn) is a comprehensive open-source ML package in Python. It comprises efficient ML and statistical modelling tools, including regression, clustering, classification, and dimensionality reduction, via a consistent interface in Python (Hao & Ho, 2019). The library is primarily written in Python and built upon NumPy, SciPy, and matplotlib (Tutorials Point, 2019).

It covers four main topics related to ML: data transformation, supervised learning, unsupervised learning, model evaluation and selection. It provides functions for data preprocessing, such as scaling, normalisation and binarisation (Hao & Ho, 2019).

3. METHODOLOGY

The research methodology establishes the reliability and validity of the data used to produce reliable results. This section adopts a typical data science approach to investigate the research questions and detail the steps. Data science methodologies help model and analyse complex systems by identifying and focusing on critical parameters that influence system behaviour, with model performance used to evaluate how well the system is represented. To achieve this, the data source, structure, pre-processing, model selection, and validation will be discussed.

3.1. Source of Data

Two Rivers Platinum (TRP), located in Burgersfort on the southern lobe of the eastern limb of the BIC, serves as the source of the dataset under consideration. TRP's primary ore sources are the Merensky and UG2 deposits, each requiring specific processing methods to optimise PGM recovery (Monele Mokadze et al., 2021). TRP uses an MF2 circuit comprising crushing, primary and secondary milling and flotation to recover PGMs with a chromium concentration of 1.5% and a chromite recovery plant with gravity separation spirals. Their operational strategy entails exclusively processing UG2 when the UG2 stockpiles meet the required quantity. When UG2 stockpiles fall short, supplementation with Merensky ore is implemented to achieve the necessary plant throughput of 495 tons per hour.

The TRP plant metallurgists supplied the input datasets in Excel and Word formats. The Word document contains detailed analyses of the target variables (recovery and chromium content in the concentrate) while the Excel file includes process, assay, and grind data. The dataset has the daily weighted mass processed and the corresponding laboratory results. Tailings are estimated by multiplying the volumetric flow rate by density. Samples are taken at noon, with subsequent laboratory analysis. Recovery is calculated using the two-product formula.

For mineralogical data, samples are collected daily downstream of the primary and secondary mills, chrome plant tailings, and high and low-grade concentrates. Laboratory results report 6E

values daily, with 4E derived using a factor of 1.05. Complete elemental profiles are averaged weekly from the daily samples.

3.1.1. Dataset Limitations and Granularity Constraints

The primary limitation of the dataset lies in its low temporal resolution, with mineralogical data available only as weekly composite samples, while operational data is collected daily. This mismatch in data granularity constrains the ability to model short-term process dynamics or predict rapid changes in concentrate mineralogy. Additionally, the absence of real-time or high-frequency measurements from critical units, such as the mills or flotation banks, limits the richness of the input features. As a result, the model relies heavily on blend composition and throughput data, which may not fully capture the complex, nonlinear interactions affecting mineralogical outcomes.

3.2. Structure of the Dataset

The Excel format dataset contains 87 columns with 1353 samples. The columns cover elemental data such as platinum, palladium, iridium, ruthenium, gold, rhodium, nickel, copper, cobalt, sulphur, and chromium percentages. Measured process data include total crushed tons, total milled tons, Merensky mill tons, dispatched concentrate tons, 6E dispatched ounces, main chromite grade, and produced chromite dry tons. The process data is from 1st January 2020 to 13th September 2023. The research's goal is to predict:

- platinum, palladium, rhodium and gold (4E) content in the concentrate (ounce/day),
- chromium concentration in the concentrate (%),
- weekly elemental composition of the ore.

Mineralogical data is collected daily but reported weekly to smooth out short-term fluctuations and provide an understanding of the central tendency of processed ore. To synchronise mineralogical and operational datasets, weekly composite samples were created, with each representing the average of daily samples over a calendar week. This approach was adopted due to the limited availability of mineralogical data, spanning from January 2022 to September 2023.

Aggregating the operational data to weekly data would result in fewer than 100 data points, significantly reducing the statistical power and limiting the model's ability to detect meaningful patterns or correlations. Maintaining the daily-resolution data would improve the model training and support more reliable trend analysis and forecasting by increasing the model's probability of capturing the operational variability.

3.3. Data Pre-Processing

The data pre-processing began with visualising the dataset to understand its global landscape comprehensively. This initial analysis, including studying the changes in the data over time, was conducted in Jupyter Notebooks using Python 3's matplotlib library, and this provided a solid foundation for subsequent processing and model development. The pre-processing stage involved preparing the dataset for an in-depth, thorough and meticulous analysis, including generating the count, mean, median, min, max, and variance. The primary objective was to ensure the quality and reliability of the dataset. The analysis revealed that all the data in the columns were stored as numeric, except for the date and Chromite Yield in Chrome Plant (%) columns, which were detected as non-numeric (text) data. These two columns' formats were converted to numeric for the Chromite Yield in Chrome Plant (%) and date format for the date column.

Furthermore, the analysis revealed that the columns Repeats_PRF, Repeats_SRT, Repeats_FC, IRS-AB_Cr_2O_3, IRS-AB_4E, IRS-AB_6E, Repeats, Chrome, and 6E Final Tailings Correction Factor exhibited low observation counts (fewer than 50 records). This suggests that these variables were either sparsely recorded due to their limited informational value, high measurement cost, or their use only under specific circumstances. As such, they were excluded from the analysis as non-predictive features, given their negligible contribution to the model's explanatory power.

Moreover, several columns serve auxiliary roles such as correction factors or reconciliation tools used primarily to resolve discrepancies between production data and third-party assays or to settle disputes with end users. These variables often exhibit near-constant or duplicative values and do not contribute additional variance.

3.3.1. Time Series Detection

A time series test was performed using the Augmented Dickey-Fuller (ADF) and Kwiatkowski-Phillips-Schmidt-Shin (KPSS) tests. This rigorous validation process is crucial in confirming the dataset's stationarity, a fundamental assumption for many statistical models.

Augmented Dickey-Fuller: ADF assesses the null hypothesis of an autoregressive integrated moving average (ARIMA) with the stationary and alternative. The conditions for a stationary series are that the mean, variance, and covariance must be time-invariant. A non-stationary series implies that the series is without a unit root (Mushtaq, 2011).

Kwiatkowski-Phillips-Schmidt-Shin (KPSS) is represented as follows:

$$y_t = \alpha + \delta_t + dr_t + \varepsilon_t \quad \text{where } \varepsilon_t \sim N(0, \sigma_\varepsilon^2)$$

$$r_t = r_{t-1} + u_t, \text{where } u_t \sim N(0, \sigma_u^2)$$

Where δ_t shows the stationary errors. KPSS uses the Lagrange multiplier test with a null hypothesis. If the null hypothesis is zero (σ_ε^2 = constant), the series is considered stationary. Rejecting the null hypothesis indicates that the data is non-stationary (Kagalwala, 2022).

3.3.2. Correlation Analysis

A correlation analysis is also conducted to detect weak correlations to other features. A correlation analysis was used to study the relationship between explanatory variables, which have a linear relationship. This enables the removal of redundant features that may cause multicollinearity. The clusters of correlated features were determined using the Pearson correlation coefficient cutoff of [0, 0.3]. The Pearson correlation coefficient, r, ranges from [0, 1]. A coefficient value of 1 indicates a strong correlation, while a 0-value coefficient suggests no relationship between the two variables.

3.3.3. Removal of Unnecessary Feature

For plant performance prediction, only features available to the plant before the ore is processed were considered for model inputs. This approach enhances the model's applicability in metallurgical plant environments, where operators rely on routinely available process measurements and require model outputs (4E concentrate flow, chromite content in the concentrate (%) and elemental assay data) to support decision-making and operational adjustments. These input features are:

- total crushed tons;
- total milled tons;
- merensky milled tons;
- PSD (this is due to the consistency and lack of variance; See Appendix D: Descriptive Statistics);
- Merensky ratio (calculated).

All the available features were considered for the elemental composition of the feed ore, as the operational data would be available to the operator when they want to make an analysis.

3.3.4. Quality Assurance

Discrepancies or inconsistencies within the dataset must be mitigated to ensure that insights drawn from the analysis are accurate. As such, empty rows within the data frame must be removed to maintain data consistency and integrity. Furthermore, negative values within the dataset were removed as metallurgical circuits cannot yield negative performance values unless the instrument malfunctions.

Data collected when the circuit was under maintenance/shutdown (Total Milled Tons = 0) was removed. This is done to ensure that 'dirty data' does not influence the detection of outliers during outlier detection. Lastly, the data type of each feature is converted to numeric (float) to ensure uniformity and accuracy in data representation.

3.3.5. Outlier Detection

Anomalies or outlier detection techniques are crucial to ensure the accuracy of trend analysis. This study developed a unique method based on the assumption of nonlinear relationships in plant data. This method, supported by the works of Auret & Aldrich (2012) and Osei (2019), was used in conjunction with RF ML to identify outliers, adding an element of novelty and intrigue to the process.

A custom function designed to fit an RF to the target iteratively was employed to enhance data quality. Based on the R^2 score, this iterative approach ensures a thorough process, reassuring the audience about the robustness of the data quality enhancement. The process is initiated by duplicating the original dataset. The RF is then fitted and trained on the entire dataset, and an R^2 score is determined. If the R^2 value is below the chosen R^2 threshold, the function will identify points in the original dataset with a significant residual difference between the actual and predicted significance value. The identified outliers are eliminated from the original dataset. This procedure is repeated until the R^2 value exceeds the threshold of 0.2.

3.4. Feature Engineering and Selection

Feature selection is an essential step in model development processes. This procedure extracts prominent features for each target variable that will be predicted. This enhances the ML's predictive power and prevents overfitting (Fernandes et al., 2019). For this study, the Gini feature importance based on the RF classifier was employed to reduce the dimensionality of the TRP dataset. This procedure streamlines the dataset by ensuring that the subsequent ML algorithms used for prediction only have the most relevant features as inputs.

Gini Feature Importance: The Gini feature importance offers a computationally efficient way of obtaining feature importance scores. This procedure is suitable for high-dimensional datasets such as time series and spectra. RFs ensemble learners based on randomising the decision trees, provide feature importance measures, including one based on statistical permutation tests and another from

the classifier's training procedure. The Gini importance, as described, ranks the features by their relevance in the classification. (Menze et al., 2009).

The Gini importance is determined during the RF training by evaluating the decrease in Gini impurity at each node, which measures the performance of the split that separates the classes. This decline is recorded and accumulated for all features across all trees in the forest, illustrating how often a feature is used for splitting and its overall discriminative value. (Menze et al., 2009).

3.5. Partition of the Data

The TRP dataset was partitioned into two parts for training and testing purposes. The train-test split function from the sklearn model selection model was used to split 70% of the dataset for training the selected model and 30% for testing purposes. The train test split function also shuffles and splits the data to prevent sampling biases.

3.6. Model Selection

All companies see data as an essential asset for digitalisation. ML algorithms provide insights (trends, patterns and correlations) learned from data. However, these insights are developed when past observations of the dataset are strong predictors for future insights (Osei, 2019). For this research, four supervised ML regression algorithms, LR, RF, Extra Tree (ET), and kNN regression models, were derived using the selected feature variables for each target variable. The selection of these ML algorithms aimed to encompass a diverse range of ML algorithms, including linear, tree-based, and instance-based algorithms. The selection also considered the trade-offs in computational complexity and training time based on the literature. LR offers fast training but limited flexibility for nonlinear data, while tree-based models like RF and ET are computationally more intensive but handle nonlinearity and interactions well. Conversely, kNN requires minimal training time but incurs higher computational cost during prediction, especially with large datasets.

3.6.1. Linear Regression

Following the feature selection, the important features for every target variable are used to generate an LR model. The model is instantiated, trained using the transformed train dataset, and tested using the transform.

For HPT, the grid search was employed to determine the optimal hyperparameters and the results are shown Table 3.1.

Table 3.1: Grid Search default and optimised hyperparameters for LR

Prediction	Hyperparameter	Default	Defined parameters	Optimised
4E content in the concentrate	fit_intercept	True	[True, False]	False
Chromium content in the concentrate	fit_intercept	True	[True, False]	False
Ore Mineralogy	fit_intercept	True	[True, False]	False

3.6.2. Random Forest

RF, developed by Breiman, utilises randomisation to produce many decision trees. Multiple samples are drawn with replacements from the training set to grow a tree. Furthermore, a split of the explanatory data is randomly selected at each node during the training phase. Post-training, prediction scores are determined by averaging the predictions from the regression trees. This prevents overfitting and variance. (Rigatti, 2017) RF algorithms perform exceptionally well with non-linear relationship tasks. (Osei, 2019).

In Python, the RandomForestRegressor function from Sklearn was utilised. By default, the number of trees in the forest (n_estimators), the number of features to consider when looking for the best

split (max_features), and the number of samples to draw (max_samples) are 100, 1.0, and 1, respectively.

The random search strategy was employed for HPT. The hyperparameters mentioned above were tuned because they influence the RF model's performance. (Probst et al., 2019). The default and optimised hyperparameters using the random search strategy are shown in Table 3.2.

Table 3.2: Random Search default and optimised hyperparameters for RF

Prediction	Hyperparameter	Default	Defined parameters	Optimised
4E content in the concentrate	n_estimators	100	1 - 300	168
	max_depth	None	1 -15	7
	min_samples_leaf	1	1 - 40	2
	max_features	1	Sqrt, log2, None	Log2
Chromium content in the concentrate	n_estimators	100	1 - 300	284
	max_depth	None	1 -15	7
	min_samples_leaf	1	1 - 40	7
	max_features	1	Sqrt, log2, None	None
Ore Mineralogy	n_estimators	100	1 - 300	400
	max_depth	None	1 -15	9
	min_samples_leaf	1	1 - 40	2
	max_features	1	Sqrt, log2, None	-

3.6.3. Extra Tree

The ET algorithm improves the RF by prioritising the randomisation of cut points for the feature variables. This reduces the variance without bootstrapping. This algorithm is highly effective in high-dimensional and complex problems, producing piece-wise multilinear approximations that

capture intricate data relationships (Polamuri et al., 2019). Like the RF, ET requires n_estimators, max_features and max_samples in Python's Sklearn library.

A random search strategy was utilised for HPT. Table 3.3 shows the results of the random search HPT exercise.

Table 3.3: Random Search default and optimised hyperparameters for ET

Prediction	Hyperparameter	Default	Defined parameters	Optimised
4E content in the concentrate	n_estimators	100	1 - 300	250
	max_depth	None	1 -15	8
	min_samples_leaf	1	1 - 40	3
	max_features	1	Sqrt, log2, None	None
Chromium content in the concentrate	n_estimators	100	1 - 300	177
	max_depth	None	1 -15	9
	min_samples_leaf	1	1 - 40	18
	max_features	1	Sqrt, log2, None	None
Ore Mineralogy	n_estimators	100	1 - 300	200
	max_depth	None	1 -15	10
	min_samples_leaf	1	1 - 40	1
	max_features	1	Sqrt, log2, None	-

3.6.4. k-Nearest Neighbours

The kNN algorithm is mainly used for trend recognition due to its simplicity and low error rate (Kuang & Zhao, 2009). kNN requires three parameters: a labelled dataset and a distance metric to determine the distance between the points and k (number of nearest neighbours) (Souza et al., 2020).

The K Neighbours Regressor in the Sklearn library was utilised. Grid search tuned the number of neighbours due to its computational simplicity and the few hyperparameters available. The initial inputs and optimised hyperparameters are shown in Table 3.4.

Table 3.4: Grid Search default and optimised hyperparameters for kNN

Prediction	Hyperparameter	Default	Defined parameters	Optimised
4E content in the concentrate	n_neighbors	5	[1, 3, 5 …39, 41]	9
Chromium content in the concentrate	n_neighbors	5	[1, 3, 5 …39, 41]	39
Ore Mineralogy	n_neighbors	5	[1, 3, 5 …39, 41]	29

3.7. Model Evaluation Metrics

This research utilised two evaluation metrics: RMSE and R-squared (R^2). The RMSE is calculated as:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} e_i^2}$$

This prediction error is computed by taking the difference between the calculated target variable and the actual target variable, applying the square function, and averaging all the other classes and internal validation data.

Lastly, R^2 was used to measure the performance of all four ML models. R^2 measures the performance of the ML-fitted regression line when compared to the presented data. A high R^2 indicates that the ML is a good fit, while a negative value indicates that the fitted regression is worse than a horizontal line, giving all predictions the mean value. (Tran et al., 2022).

3.8. Naïve Benchmark

However, all models will be compared using the evaluation metrics stated in section 5.7. A naïve benchmark will be established to comprehend the model's performance further. The naïve benchmark is a sanity check that ignores all predictor information by calculating the average 4E and chromium content outcomes. These outcomes indicate whether the ML algorithm is an excellent predictive model. A good predictive model outperforms the benchmark criterion in terms of predictive accuracy.

For the detailed methodology, see Appendices B and C, which used Python code in Jupyter Notebooks. The algorithms discussed in the method were implemented using Sklearn.

4. RESULTS

This section presents the results obtained through the methodology outlined in Section 3 to offer insights and identify trends to address the research questions formulated in the study's introduction.

4.1. Data Preprocessing and Distribution Analysis

The data preprocessing procedure eliminated 70% of the original dataset. Most of the data was eliminated in the discrepancy stage as the mineralogical data was only recorded from 2022, while the plant performance data was recorded from 2020. The distribution of the target cleaned data is shown in APPENDIX D. The distribution shape of the ore mineralogy is generally skewed to the left, while the 4E and chromium content in the concentrate is skewed to the right, as shown in Figure D7.1.

6.1. Time Series

The outcome of the data processing is outlined below to ensure the use of high-quality data for model development. A time series analysis using the ADF and KPSS was conducted to confirm the stationarity of the dataset. The test outcomes are summarised in Table 4.1.

Table 4.1: Time Series Test Results

Test Method	p-value
ADF	2.37e-12
KPSS	0.01

The test method results in Table 4.1 show the p-values for the ADF and KPSS are 2.37e-12 and 0.01, respectively. For the ADF, a low p-value indicates that process data is stationary as the mean and variance remain constant over time. This suggests that the ARIMA ML would perform poorly poorly due to the consistent mean and variance over time. For the KPSS, a p-value of 0.01 indicates sufficient evidence that the process data does not have seasonal trends. This points to the idea that the KPSS test method detects stationary trends.

Furthermore, a visual observation is shown in Figure 4.1, which shows the fraction of the throughputs and supports the ADF by showing linear or stationary trends, as there is no clear indication of temporal dependencies or links between previous values.

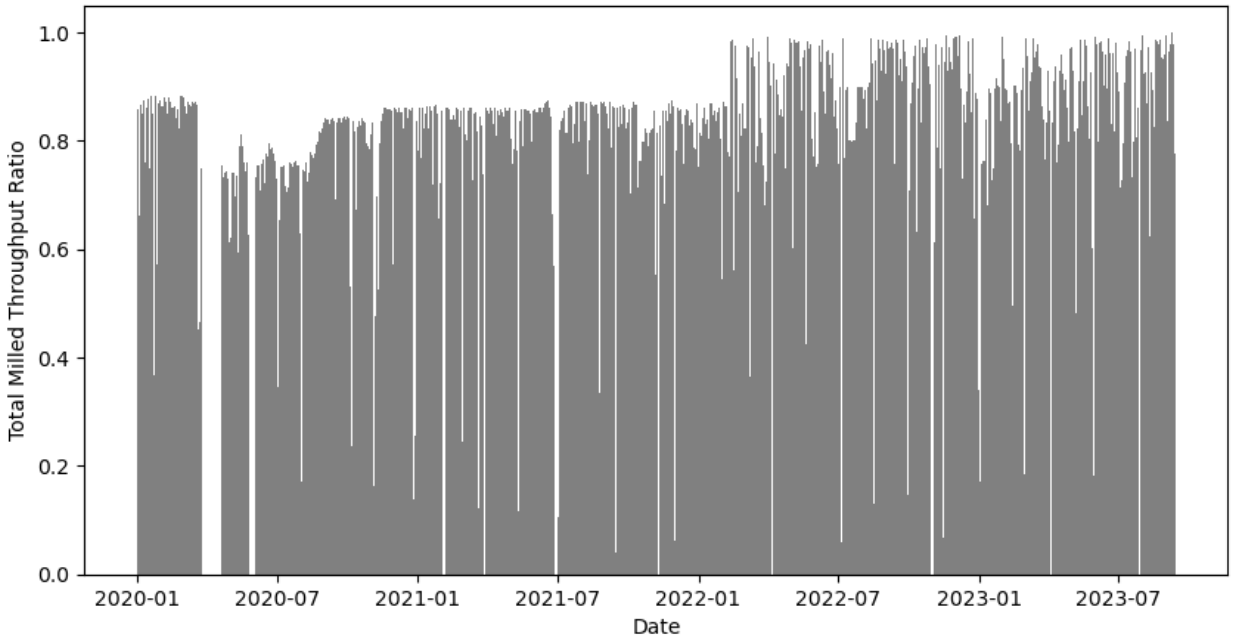


Figure 4.1: Total milled throughout ratio time series

6.2. Influence of Merensky Incorporation

One of the primary objectives of this research is to evaluate the crucial role of the Merensky ore in the MF2 circuits. This involves a comprehensive assessment of key performance indicators (KPI) which are used by TRP, specifically the 4E product recovery (%) and chromium content in the concentrate (%).

The platinum concentrate required post-flotation plays a pivotal role in the MF2 operation. To optimise the MF2, it's essential to focus on the 4E product recovery (%) and adhere to the stipulated chromium content threshold, as these factors are crucial for efficient 4E extraction and smelter lining damage mitigation. Table 4.2 illustration summarises the average grind data for the four

classes, 4E product recovery (%), chromium content in the concentrate (%), throughput, and their respective types of processed ore.

Table 4.2: Influence of Merensky on the average PSD post MF1

Feed	Average 4E Recovery (%)	Average Chromium Content (%)	Average Throughput (tons/day)	Average -75 µm (%)	Average +75 µm (%)	Average +106 µm (%)	Average +150 µm (%)
UG2	79.1	1.48	10 200	39.5	10.3	12.2	37.4
Merensky-UG2	78.2	1.38	9 830	39.6	10.5	12.4	37.3

The findings show that the UG2 has an average 4E product recovery (%) of 79.1%, and the Merensky-UG2 blend has an average 4E product recovery (%) of 78.1%. This observation indicates that the recovery of UG2 and Merensky-UG2 ore is marginally different. Similarly, the average chromium content in the concentrate in the UG2 and Merensky-UG2 blend is statistically the same, as the difference between the values is marginal.

To understand the TRP MF2 operation at a high level, the grind data for the primary mill indicates a marginal difference between the UG2 and Merensky-UG2 operational modes. This suggests that the mill performance is strictly controlled to produce consistent grind data. However, the throughput between the two operational modes is significantly different, as the throughput for UG2, on average, is 10,200 tons/day while the throughput for the blend is 9,830 tons/day. This suggests that the mill residence time is increased to maintain consistent grind data in the blend, as in the UG2. This is due to the excessive presence of pyroxene and plagioclase in Merensky.

6.3. Correlation Analysis

The correlation results are displayed in a Python-generated correlation heatmap, Figure 4.2, which illustrates the relationships between various variables in the dataset. These relationships are crucial

for comprehending the data's underlying trends and selecting relevant features for model development.

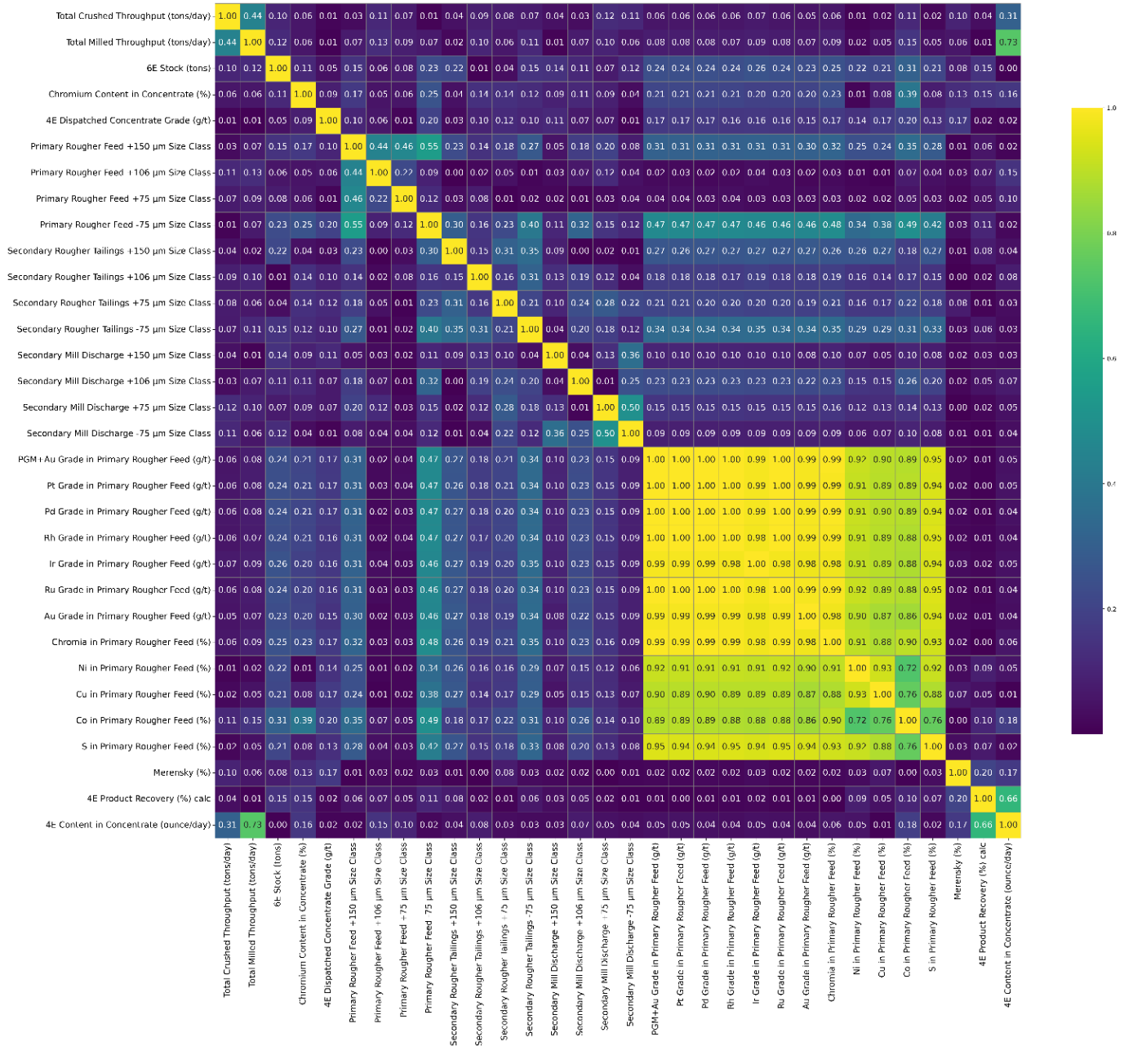


Figure 4.2: Correlation Matrix of the features and targets used

Figure 4.2 illustrates the correlation between pairs of variables, the target variables, and the available features. The Pearson coefficient for this heatmap is between 0 and 1, where 1 indicates a perfect correlation, and 0 indicates no correlation. The heatmap illustrates that the highest correlated feature for 4E and chromium content in the concentrate is the Total Milled Throughput and the primary rougher feed +75µm, excluding the target features (mineralogy, 4E and chromium content in the concentrate). For these targets, the highest Pearson correlation values are 0.73 and 0.25, respectively. This suggests a dependency of the target variables on the primary milling operations. Moreover, it suggests that the 4E and chromium content in the concentrate are generally less dependent on the secondary mill operation.

Among the operational parameters, the percentage passing +75 µm in the primary rougher feed showed the strongest correlation with the ore mineralogy, with a Pearson correlation coefficient of approximately 0.5. This suggests that the degree of liberation, as represented by particle size, has a moderate influence on the mineralogical composition detected in laboratory analyses. The identification of individual minerals was based on linear correlations between their measured concentrations and the reference mineral used in the analysis, as they have the same Pearson correlation coefficient. Overall, as indicated by the correlation heatmap, ore mineralogy exhibits generally weak correlations with throughput-related variables. This is expected, as ore mineralogy is primarily influenced by upstream factors such as feed characteristics and comminution efficiency, whereas throughput is more closely linked to downstream operational strategies driven by production targets.

4.4. Real-Time Prediction of Ore Mineralogy

The LR, RF, ET, and kNN models were employed for ore mineralogical prediction, a key focus of this research. The dataset characterises the ore mineralogy by PGM+Au grade, platinum (Pt) grade, palladium (Pd) grade, rhodium (Rh) grade, iridium (Ir) grade, ruthenium (Ru) grade, gold (Au) grade, Chromium (Cr2O3) %, nickel (Ni) %, copper (Cu) %, cobalt (Co) % and sulphur (S) %.

4.4.1. Feature Engineering and Selection

In pre-model development, it is important to consider features most relevant to the target variable based on statistical significance. The most relevant features are identified using the RF classifier and Gini importance ranking with a significance level 0.03. The ranking and feature importance for the ore mineralogy model development visualisation are shown in Figure 4.3.

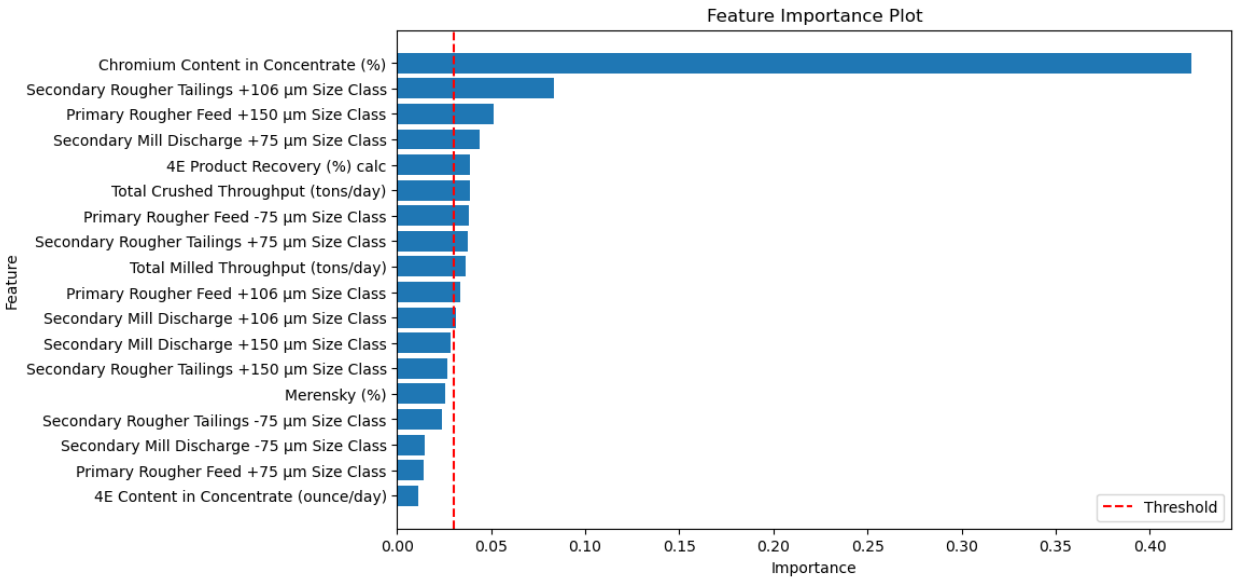


Figure 4.3: Coefficient Values for the features of each target

For the feature selection and assessment of the most relevant features in ore mineralogy prediction, the chromium content in concentrate (%) was identified as the most critical feature with importance above 0.4. Notably, the RF classifier deemed the Merensky (%) in the ore insignificant. This is due to the low Merensky (%) in the ore, which insignificantly impacts the ore mineralogy.

4.4.2. Linear Regression

Post-model development, the performance metrics of the ore mineralogy prediction using the LR are displayed in Figure 4.4 to address the research question of building an ore mineralogy data-based model.

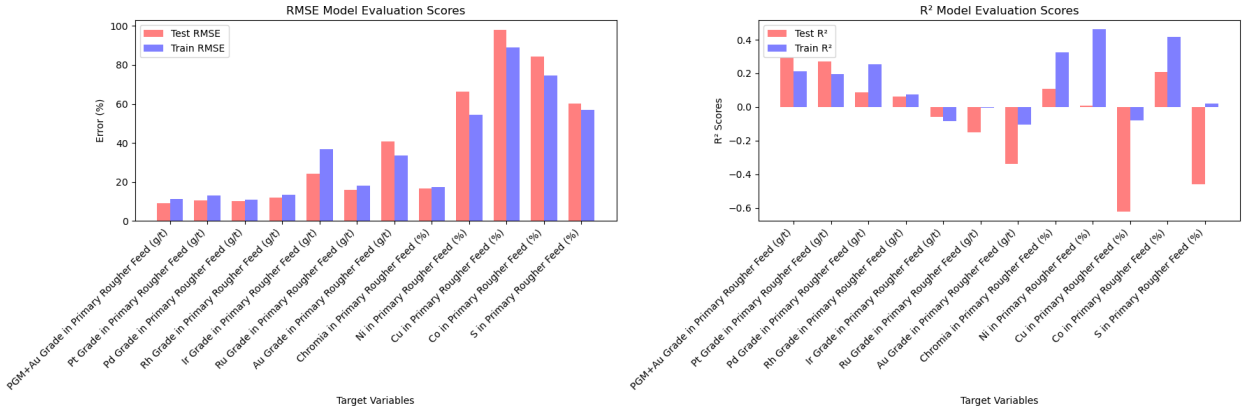


Figure 4.4: Performance Metrics of the Linear Regression

The LR regression model demonstrated a poor performance across the different characteristics. This reveals the LR's limitation in capturing nonlinear relationships in the dataset. For the PGM+Au grade, the RMSE is determined to be 11.3% and 8.97% for the training and test data, respectively. Similar patterns are experienced by the Pt grade, Pd grade, Rh grade, Ru grade and Cr_2O_3 %. For the other characteristics, the RMSE is high, indicating that the LR exhibited weak performance in predicting these characteristics.

The R^2 for the PGM+Au grade, Pt grade, Pd grade, Rh grade, Cr_2O_3 %, Ni % and Co % are positive. This suggests that the relationship between these grades and the features is linear. Conversely, for the Au grade, Ru grade, Co % and S %, the negative R^2 values indicate that the data's relationship with the process operation is possibly nonlinear. A visual representation of the original data points and the LR predicted points is shown in Figure 4.5.

From Figure 4.5, the LR generally captured the extreme values of the displayed characteristics except for the Au grade in the time series plot. Furthermore, except for Au grade, the actual data and predicted data plots' regression line insignificantly deviate from the x=y line, the standard by which performance is measured visually.

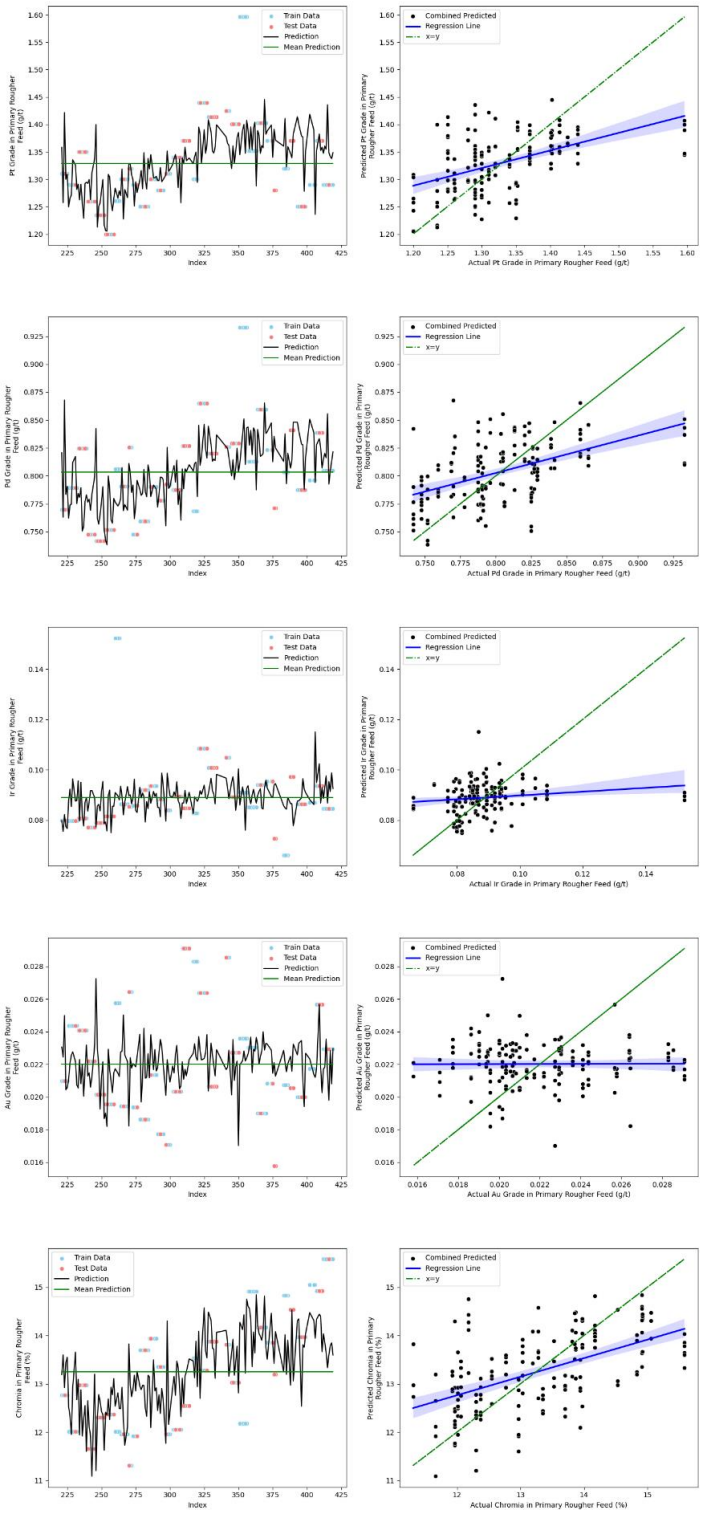


Figure 4.5: LR prediction time series and actual vs pred. plot for ore mineralogy

4.4.3. Random Forest Regression

To address the research question, the performance metrics using both the training and test data of the RF model are shown in Figure 4.6.

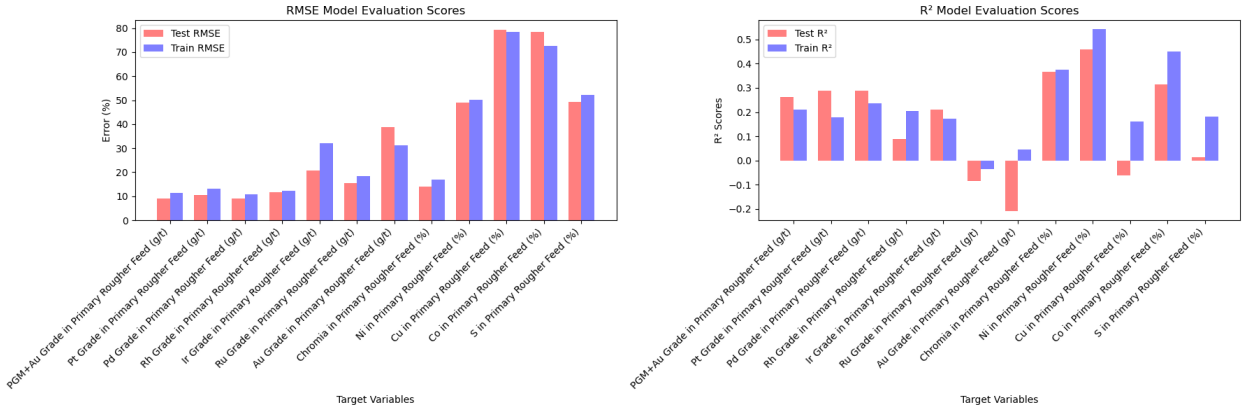


Figure 4.6: Performance Metrics of the RF regression

Unlike the LR, the RF model demonstrates better performance across the different characteristics. This reveals the RF's potential to capture linear and nonlinear relationships within the dataset. For the PGM+Au grade, Pt grade, Pd grade, Rh grade, Ru grade, and Cr_2O_3 %, the RMSE remained low, indicating the model's ability to predict these characteristics accurately. As with the LR, the other characteristics exhibited high RMSE values for training and test data.

For the R^2 values, the RF model captured some of the variance for the PGM+Au grade, Pt grade, Pd grade, Rh grade, Ir grade, Cr_2O_3 %, Ni % and Co %. However, the RF failed to capture any variance for the Ru grade for the training and test data. Furthermore, for the S %, the R^2 of the training and test differed significantly. This could indicate overfitting as the model needs to be generalised. Visual representations of the selected ore mineralogy (Pt grade, Pd grade, Rh grade, Au grade and Cr_2O_3 %) are shown in Figure 4.7.

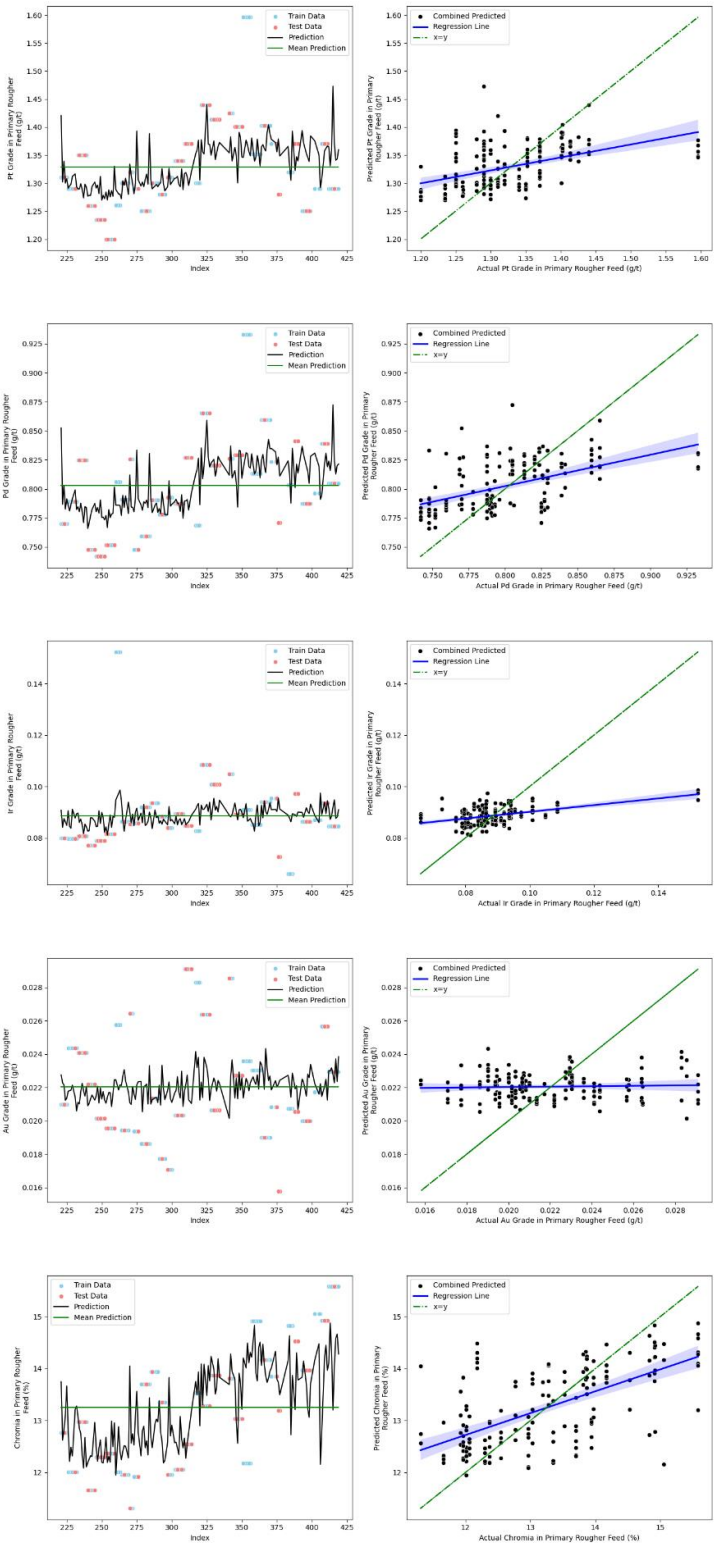


Figure 4.7: RF prediction time series and actual vs pred. plot for ore mineralogy

Based on Figure 4.7, the RF has captured the extreme values for Cr_2O_3 % based on the time series. However, the plot's regression line poorly aligns well with the x=y line for the actual vs. predicted data plot, indicating a poor predictive tool for Cr_2O_3 % data with non-extreme values. The model for the Au grade captured none of the variance.

4.4.4. Extra Tree Regression

The performance metrics for the ore mineralogy prediction using the ET algorithm are shown in Figure 4.8. This addresses the research question of developing an ore mineralogy data-based model for prediction.

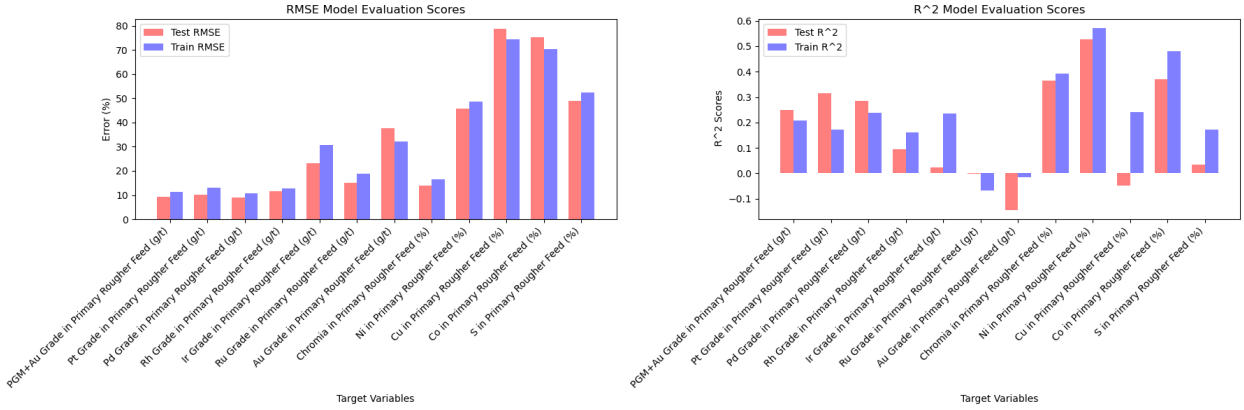


Figure 4.8: Performance Metrics of the ET regression

The RMSE and R2 evaluation metrics were determined for each characteristic in the ore. Like the RF model, the ET demonstrates a better performance when compared to the LR, further revealing the nonlinear relationship between the ore mineralogy and the process operation. For the PGM+Au grade, Pt grade, Pd grade, Rh grade, Ru grade and Cr2O3 %, the error percentages are 11.3%, 13.1%, 10.8%, 12.7% and 16.5% for the training data and 9.24%, 10.2%, 9.03%, 11.7% and 13.9% for the test data. For the poorly predicted characteristics, the RMSE for the Ir grade, Au grade, Ni %, Cu %, Co % and S % is 30.8%, 32.2%, 48.5%, 74.5%, 70.4% and 52.5%, respectively, for the training data. For the respective characteristics, the RMSE for the test data is 23.1%, 37.7%, 45.7%, 78.8%, 75.3% and 48.8%.

For the R^2 values, the RF model captures some of the variance for the PGM+Au grade, Pt grade, Pd grade, Rh grade, Ir grade, Cr2O3 %, Ni %, and Co %. Conversely, the ET for both datasets does not capture the variance of the Ru and Au grades at all. Furthermore, the variance of the S % significantly differed between the training and test datasets, indicating possible overfitting due to the model's inability to generalise using the training dataset.

Figure 4.9 illustrates the visual representation of the selected characteristics (Pt grade, Pd grade, Rh grade, Au grade and Cr_2O_3 %). Based on the visual representation, the Pt, Pd, Rh, and Cr_2O_3 % trends are well captured except for the extreme values. For the Au grade, the ET fails to capture the trends. The plots' regression lines align with the x=y reference line for the actual vs predicted data plot. However, for the Au grade, the plots' horizontal regression line indicates that the ET model captured none of the Au grade's variance.

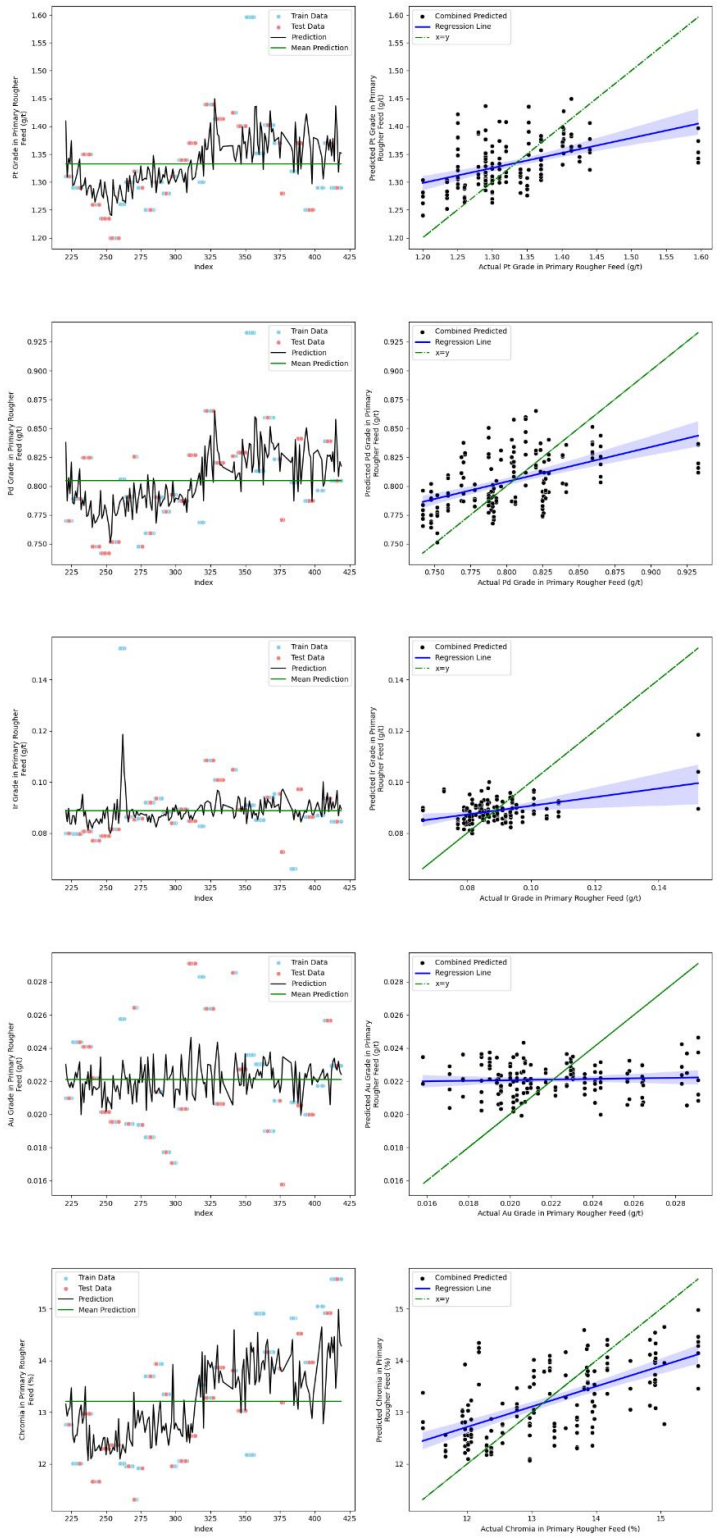


Figure 4.9: ET prediction time series and actual vs pred. plot for ore mineralogy

4.4.5. K-Nearest Neighbours Regression

The evaluation metrics of the kNN regression to predict ore mineralogy are shown in Figure 4.10. These results address the research question of constructing a data-based model to predict ore mineralogy based on plant performance.

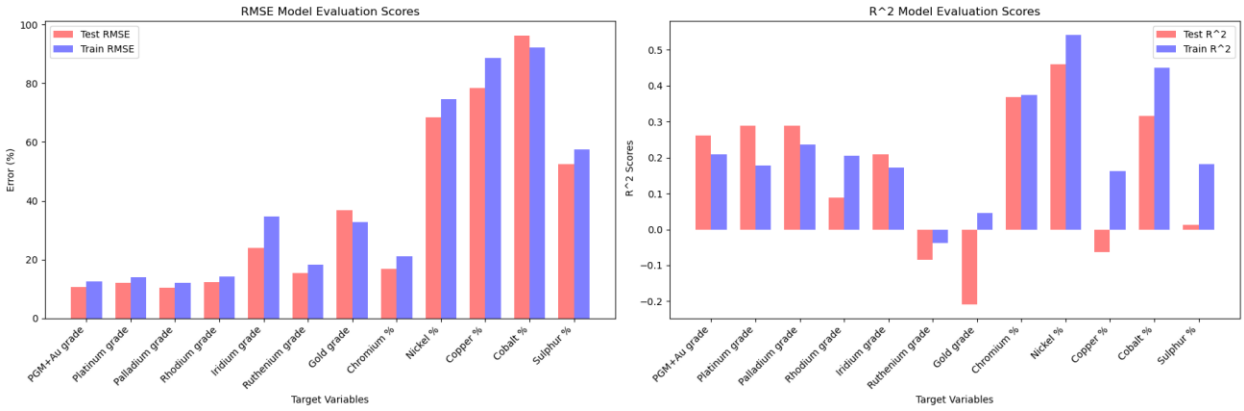


Figure 4.10: Performance Metrics of the kNN regression

For the kNN ML evaluation, the RMSE and R^2 evaluation metrics, which measure the average prediction error and the proportion of the variance in the dependent variable that is predictable from the independent variable(s), respectively, are determined for each ore characteristic. Based on Figure 4.10, the kNN illustrates a varying degree of performance. Like the tree-based models, PGM+Au grade, Pt grade, Pd grade, Rh grade, Ru grade and Cr_2O_3 % are 12.5%, 14.1%, 12.1%, 14.2%, 18.2% and 21.0% for the training data and 10.5%, 12.0%, 10.5%, 12.3%, 15.4% and 16.8% for the test data respectively. However, the training data RMSE for the poorly predicted characteristics are 34.6%, 32.9%, 74.7%, 88.8%, 92.1% and 57.5% for Ir grade, Au grade, Ni %, Cu %, Co % and S % and for the test data, the RMSE is 23.9%, 36.8%, 68.3%, 78.3%, 96.4% and 52.6% respectively.

For the R^2 scores, the kNN captures the variance of PGM+Au grade, Pt grade, Pd grade, Rh grade, Ir grade, Cr_2O_3 %, Ni %, Co % and S %. However, the variance of Ru grade is not captured by the

kNN. For the Au grade and Cu %, the kNN fails to generalise as the variance of the training data is captured while the unseen data's variance is not.

Figure 4.11 shows the fits of the selected characteristics (Pt grade, Pd grade, Rh grade, Au grade, and Cr_2O_3 %). Based on the visual representation, all the trends could be better captured. The plots' regression lines deviate significantly from the x=y reference line, representing perfect prediction for the actual vs. predicted data plot. This implies that the predictive accuracy of the kNN for ore mineralogy is limited.

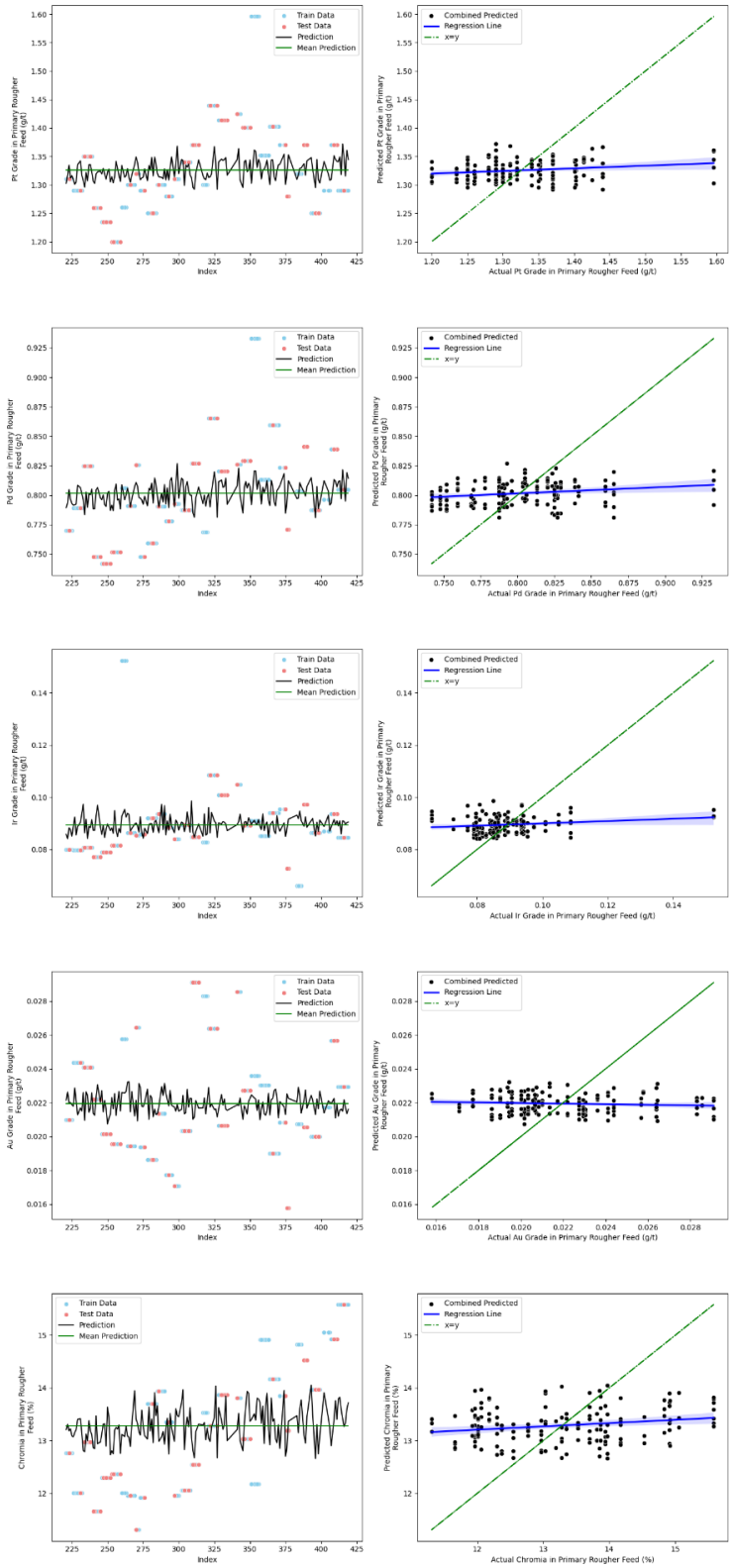


Figure 4.11: kNN prediction time series and actual vs pred. plot for ore mineralogy

4.4.6. Performance Evaluation of the Multiple Predictive Models

The RMSE and R^2 metrics presented in Table 4.3 represent the mean performance values computed across all predicted ore characteristics. In a multi-output regression setting, where multiple dependent variables such as Pt grade, Pd grade, Cr_2O_3 %, and others are predicted simultaneously, evaluation metrics are determined individually. These values are then averaged to provide an overall performance indicator for each model. This approach facilitates a general comparison across models by summarising predictive accuracy and explained variance into single representative values. A summary of the four mentioned ML algorithms' evaluation metrics is illustrated in Table 4.3.

Table 4.3: Performance Metrics of the Predictive Model

Regression Model	Model Type	Training Data		Test Data	
		Average RMSE (%)	Average R^2	Average RMSE (%)	Average R^2
Linear	Linear	35.7	0.141	37.3	-0.048
Random Forest	Tree-based	33.3	0.021	32.1	-0.458
Extra Tree	Tree-based	32.7	0.233	31.5	0.172
k-Nearest Neighbours	Other	39.4	0.161	36.2	0.227

Table 4.3 presents the averaged RMSE and R^2 for the LR, RF, ET, and kNN training and test datasets. The LR has an average RMSE of 35.7% for the training data and 37.3% for the test data. For the average R^2, the LR exhibits 0.141 on the training data and -0.048 on the test data. These results indicate limited predictive capability, with the model explaining only a small proportion of variance in the training data and performing worse than a baseline mean predictor on the test data. This suggests that the LR model does not generalise well and is not suitable for capturing the underlying patterns in the dataset.

For the tree-based models RF and ET, the respective averaged RMSE for the training data is 33.3% and 32.7%, and for the test data, the respective averaged RMSE is 32.1% and 31.5%. For the R^2 scores, the RF and ET have R^2 scores of 0.021 and 0.233 for the training data and -0.458 and 0.172 for the test data. The tree-based models reflect a slightly better performance for the ET, indicating its potential for improved performance compared to the LR and RF.

The kNN model, while recording an average RMSE of 39.4% for the training data and 36.2% for the test data, shows promise with an R^2 value of 0.161 for the training data and 0.227 for the test data. This indicates that the kNN generally captures the variance of the ore mineralogy when compared to the other models. The model's accuracy can be significantly improved with further refinement, offering hope for its future potential. However, the ET outperforms it when it comes to the RMSE%.

4.5. Real-Time Prediction of Plant Performance

Considering the identified KPIs in Section 4.26.2, the development of a data-based plant performance model focused on the 4E and chromium content in the concentrate. Like the ore mineralogy, the LR, RF, ET, and kNN ML algorithms were investigated. The four models were trained using 70% of the dataset, and 30% of the data was used to validate the model.

4.5.1. Naïve Benchmark

As a sanity check, the naïve benchmark, the mean of the target values, is used to justify using the specified ML algorithms. The results are shown in Table 4.4.

Table 4.4: Naïve Baseline Results for Plant Performance

Target Variable	RMSE (%)
4E content in concentrate	20.7
Chromium content in concentrate	20.0

4.5.2. Feature Engineering and Selection

Before developing the model, the most relevant features to the target variable are based on the statistical significance level of 0.03. Using the RF classifier and Gini importance ranking, a method that ranks the importance of features in a predictive model, the feature importance and ranking for the 4E and chromium content in the concentrate predictive model visualisation is shown in Figure 4.12.

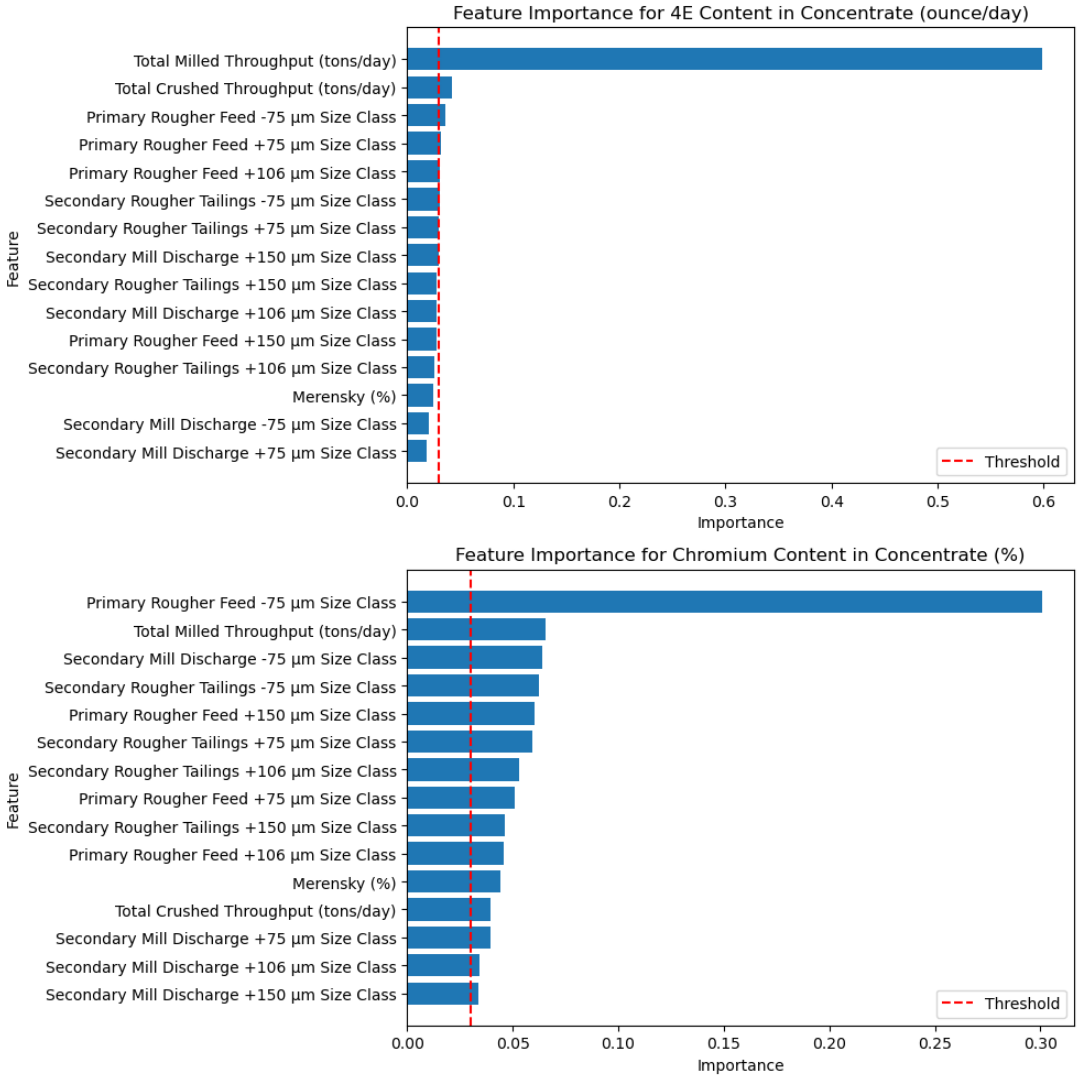


Figure 4.12: Coefficient Values for the features of 4E and Chromium in concentrate

The feature selection and assessment of the most relevant features in the plant performance prediction for the 4E content in concentrate was a thorough process. It indicated that eight features (Primary Rougher Feed +150 µm Size Class, Primary Rougher Feed +75 µm Size Class, Primary Rougher Feed -75 µm Size Class, Secondary Rougher Tailings +75 µm Size Class, Secondary Rougher Tailings -75 µm Size Class, Secondary Mill Discharge +150 µm Size Class, Total Crushed Throughput, Total Milled Throughput) emerged as the most relevant features. Primary rougher feed +75 µm size class showed the most significant positive influence with a coefficient of 680, and secondary mill discharge +150 µm size class had a notable negative impact coefficient of -167.

All the features were selected for model development for the chromium content in the concentrate, with Merensky% % emerging as the most significant factor (coefficient = -0.382).

4.5.3. Linear Regression

The plant performance evaluation metrics using the LR algorithm are shown in Table 4.5. This addresses the research question of developing a plant performance predictive model.

Table 4.5: Performance Metrics of the LR regression

	Training Data		Test Data	
	RMSE (%)	**R^2**	**RMSE (%)**	**R^2**
4E content in concentrate	11.2	0.532	10.7	0.574
Chromium in concentrate	15.0	-0.017	17.6	-0.188

The LR model achieved an RMSE for 4E and chromium content in the concentrate of 11.2% and 15.0% for the training data, respectively, and 10.7% and 17.6% for the test data, respectively.

The LR model's R^2 scores for the training data were 0.532 and -0.017, while the test data's R2 values for the 4E and chromium content in the concentrate were 0.574 and -0.188. For the

chromium content in the concentrate, the performance metrics indicate that the relationship between the features and the targets is possibly nonlinear.

In Figure 4.13, the visual representation of the LR model demonstrates its ability to capture the variance of the 4E content in the concentrate, aligning its regression line with the x=y line, which is the performance standard. However, the variance of the chromium content in the concentrate needs to be better captured, as the regression line needs to align better with the x=y line.

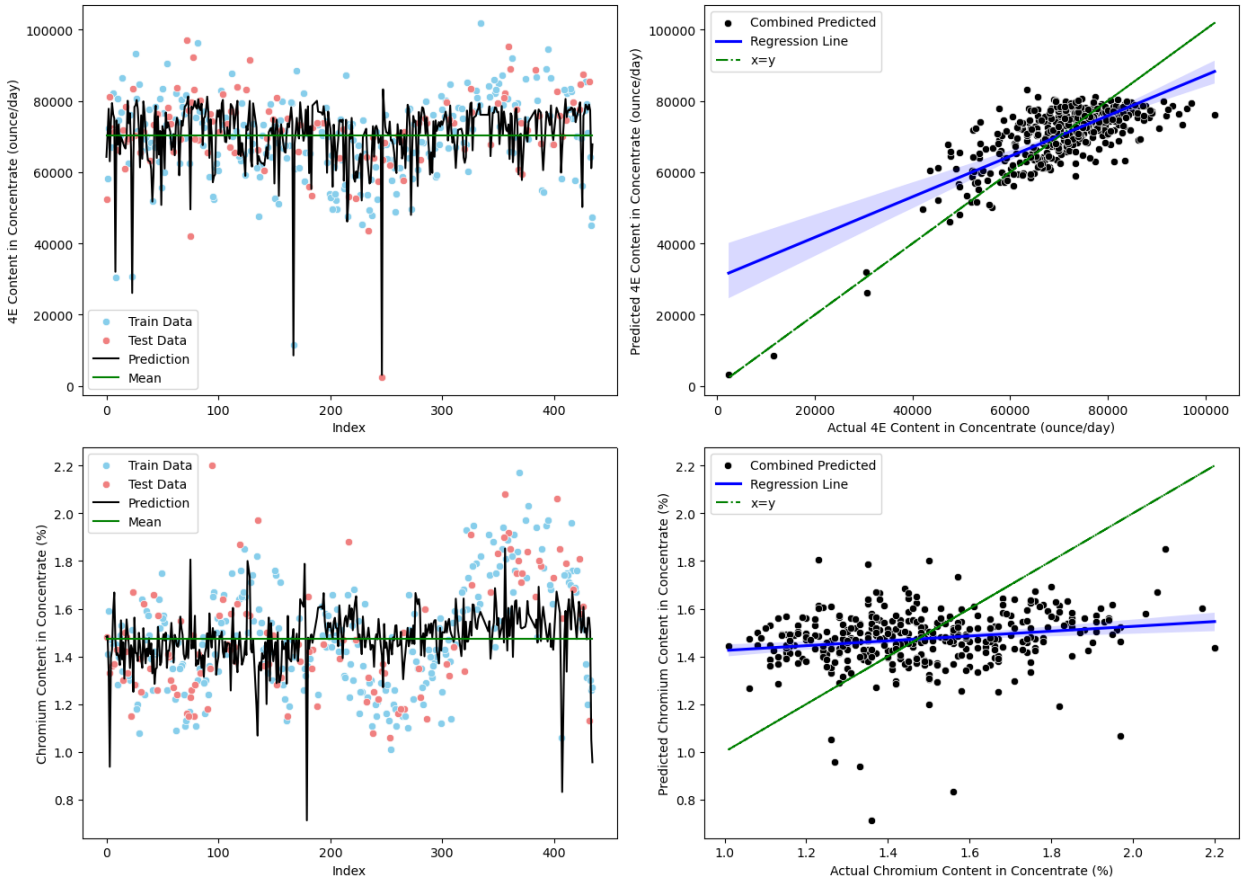


Figure 4.13: LR prediction time series and actual vs pred. plot for plant performance

4.5.4. Random Forest Regression

To address the research question, the evaluation metrics (RMSE and R^2) for the RF's plant performance are illustrated in Table 4.6.

Table 4.6: Performance Metrics of the RF regression

	Training Data		Test Data	
	RMSE (%)	**R^2**	**RMSE (%)**	**R^2**
4E content in concentrate	12.3	0.435	13.1	0.360
Chromium in concentrate	12.9	0.248	15.5	0.101

For the 4E content in the concentrate, the RF achieved an RMSE of 12.3% for the training data and 13.1% for the test data. Similarly, the RF achieved an RMSE of 12.9% for the training and 15.5% for the validation data. With the R^2, the RF captured 43.5% of the training data and 36.0% of the unseen data for the 4E content in the concentrate. Lastly, the RF captures 24.8% of the training data's variance, while the RF captures 10.1% of the test data. These results imply that the nonlinear models adequately capture the trends of plant data. Further, the similarities between training and test data evaluation metrics indicate that the RF generalises well.

Figure 4.14 illustrates the visual representation of the RF model predictions and its ability to capture the variance of the 4E content in the concentrate. The time series visualisation shows that the RF model generally captures the variance of the 4E content in the concentrate. However, it struggles with extreme values and the chromium content in the concentrate. Both predictions show that the regression line somewhat aligns with the x=y line for the actual vs predicted plot.

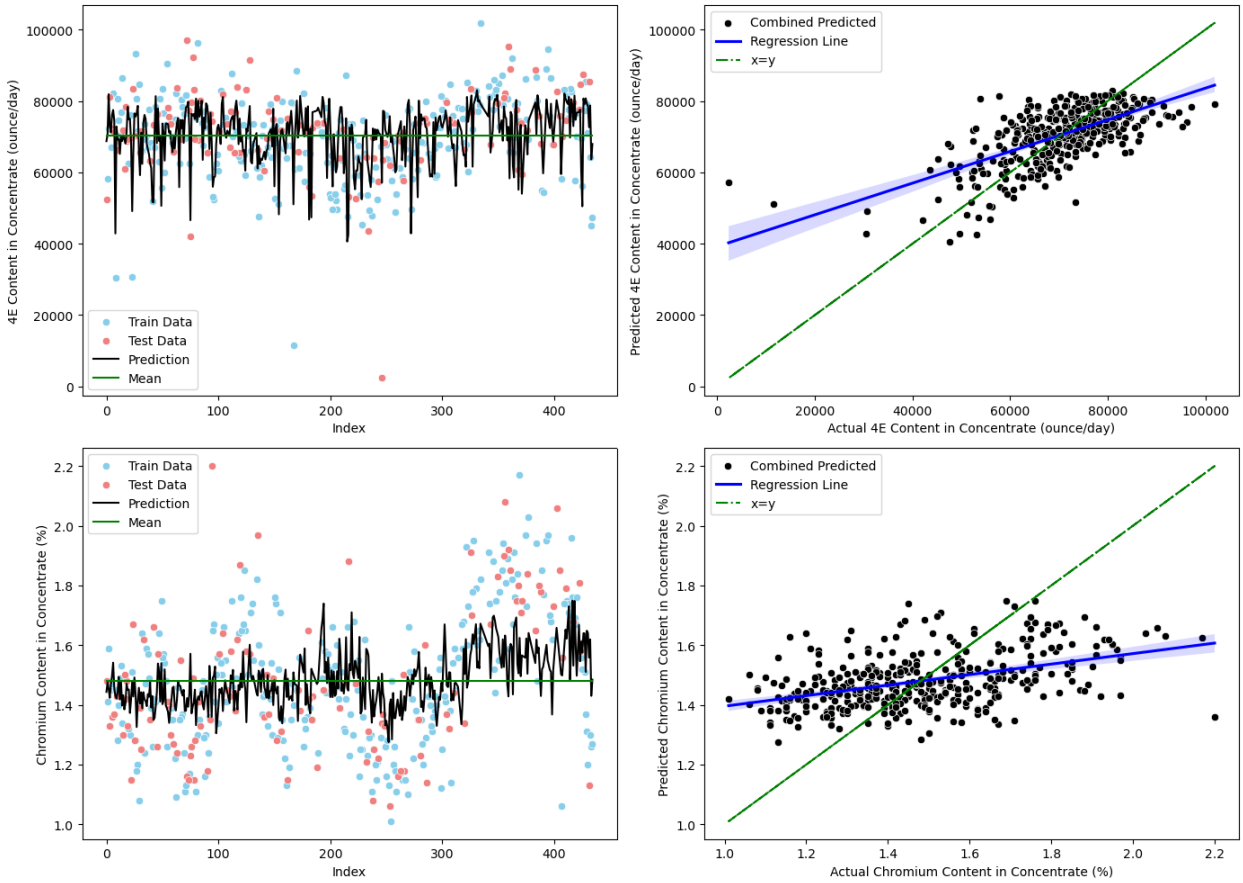


Figure 4.14: RF prediction time series and actual vs pred. plot for plant performance

4.5.5. Extra Tree Regression

Table 4.7 addresses the crucial research question of developing a predictive model for plant performance using the ET algorithm. The plant performance evaluation metrics obtained using the ET algorithm are presented here.

Table 4.7: Performance Metrics of the ET regression

	Training Data		Test Data	
	RMSE (%)	**R^2**	**RMSE (%)**	**R^2**
4E content in concentrate	12.5	0.424	13.6	0.312
Chromium in concentrate	12.9	0.262	15.4	0.135

Like the RF model, the ET model predicts the 4E content in the concentrate with an RMSE of 12.5% (training data) and 13.6% (test data). This consistency underscores the effectiveness of both models. However, the ET model performs better for the chromium content in the concentrate, achieving an RMSE of 12.9% for the training data and 15.4% for the test data.

Regarding the R^2 score of the 4E content in the concentrate, the ET model captured 42.4% of the training data's variance and can explain 31.2% of the unseen data. However, for the chromium in the concentrate, the model captured 26.2% of the training data's variance and 13.5% of the test data.

The ET time series plot visualisations in Figure 4.15 indicates that the ET decently captures the variance in 4E content, except for extreme value, but also struggles with chromium content accuracy. In both cases, the regression line roughly aligns with the x=y line in the actual vs. predicted plot.

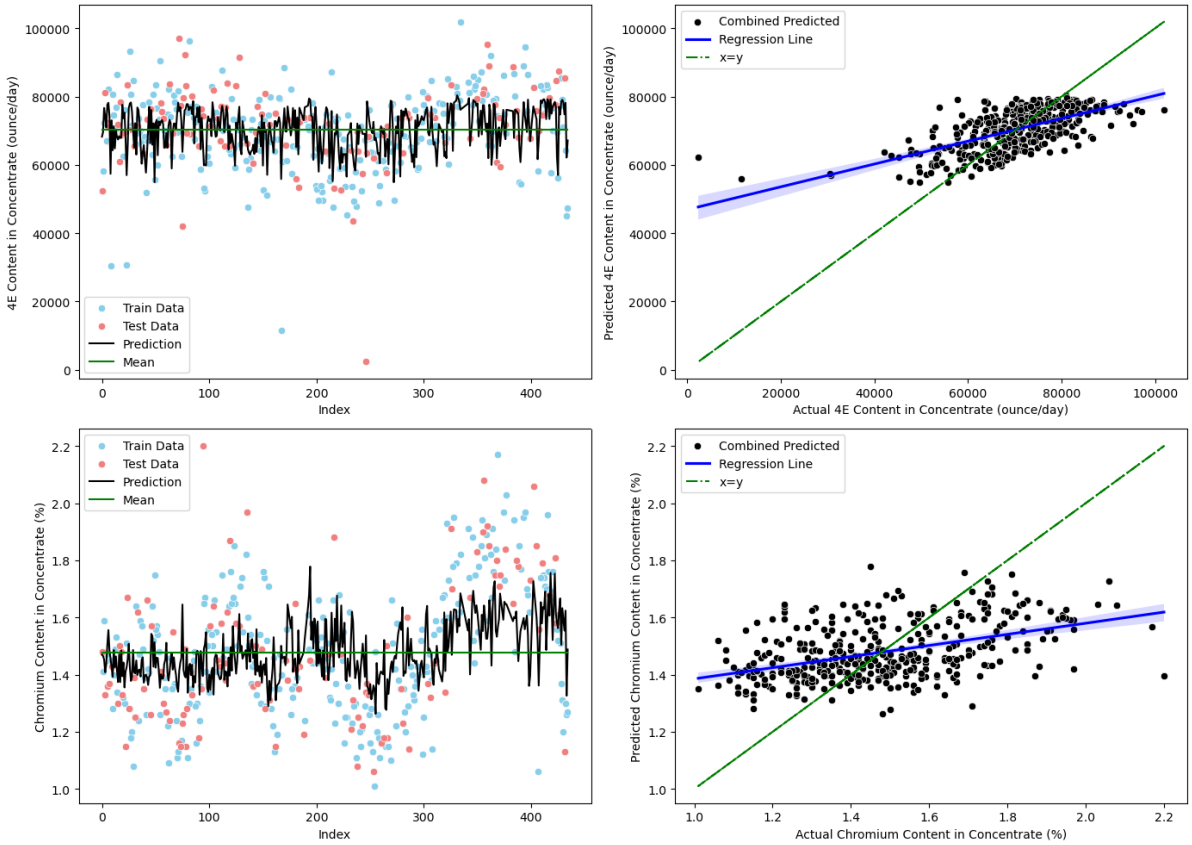


Figure 4.15: ET prediction time series and actual vs pred. plot for plant performance

4.5.6. K-Nearest Neighbours Regression

Table 4.8 presents the evaluation metrics for the ET algorithm, focusing on developing a predictive model for plant performance:

Table 4.8: Performance Metrics of the kNN regression

	Training Data		Test Data	
	RMSE (%)	**R^2**	**RMSE (%)**	**R^2**
4E content in concentrate	12.6	0.409	13.7	0.304
Chromium in concentrate	15.0	-0.024	16.7	-0.045

The kNN model predicts 4E content with an RMSE of 12.6% on the training data and 13.7% on the test data. For chromium content, the RMSE is 15.0% for training data and 16.7% for test data. The kNN's R^2 values for the training and test data indicate that it performs worse than the mean when predicting chromium content on average, which has significant implications for its practical use.

Figure 4.16 illustrates that the kNN model decently captures the variance in 4E content, except it struggles with chromium content. Similarly, the regression line roughly aligns with the x=y line in the actual vs. predicted plot for the 4E content and does not align with the x=y line in the chromium content actual vs. predicted plot.

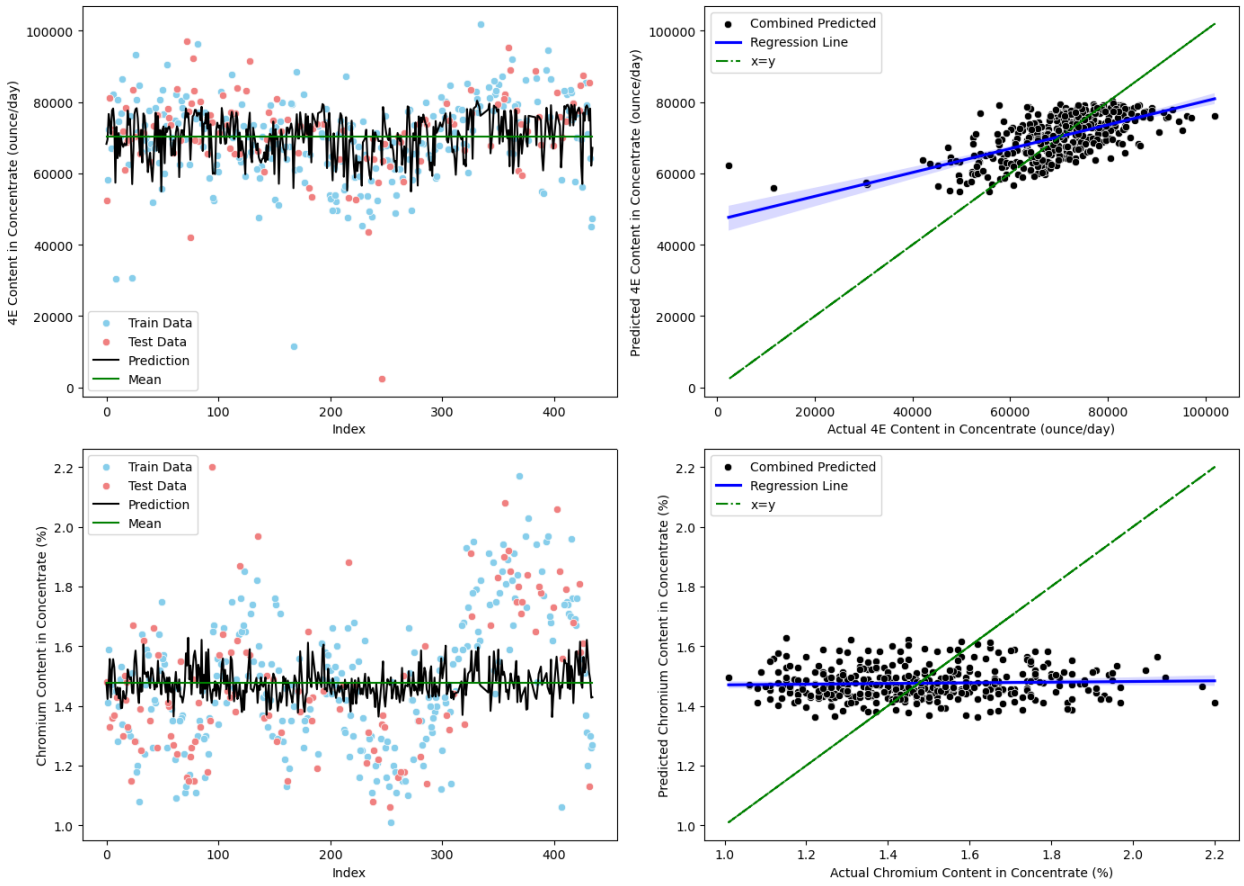


Figure 4.16: kNN prediction time series and act. vs pred. plot for plant performance

4.5.7. Performance Evaluation of the Multiple Predictive Models

The listed predictive models developed in this study were trained and validated using the test data using a 70/30 split. The performance of the models was assessed using the RMSE and R^2, and the results are summarised in Table 4.9.

Table 4.9: Performance Metrics of the Predictive Model

Regression Model	Model Type	Target Feature	Training Data		Test Data	
			RMSE (%)	R^2	RMSE (%)	R^2
Linear	Linear	4E content in concentrate	11.2	0.53	10.7	0.57
		Chromium in concentrate	15	-0.02	17.6	-0.19
Random Forest	Tree-based	4E content in concentrate	12.3	0.44	13.1	0.36
		Chromium in concentrate	12.9	0.25	15.5	0.10
Extra Tree	Tree-based	4E content in concentrate	12.5	0.42	13.6	0.31
		Chromium in concentrate	12.9	0.26	15.4	0.14
k-Nearest Neighbours	Other	4E content in concentrate	12.6	0.41	13.7	0.30
		Chromium in concentrate	15	-0.03	16.7	-0.05

Table 4.9 compares the predictive performance of four regression models across two target variables: 4E content and chromium content in concentrate. The LR model produced the lowest RMSE and highest R^2 values for 4E content, indicating comparatively better performance for this target. However, its performance for chromium content was limited, as reflected by higher RMSE and negative R^2 values. The RF and ET demonstrated consistent and balanced performance across both target variables, with moderate RMSE values and positive R^2 scores. The kNN model exhibited the lowest overall performance, particularly for chromium content, where it produced negative R^2 values for both training and test datasets.

4.6. Execution Time Analysis of Predictive Models in Plant Performance

This section describes the prediction time of the LR, RF, ET and kNN models. It assesses the computational efficiency of each model, complementing accuracy metrics such as RMSE and R^2. By reporting execution times across ten trials, including training, tuning, and evaluation, it enables comparison of models based on both performance and resource demands. Each model was trained, hyperparameter tuned and evaluated ten times to measure the execution minimum, average and maximum lengths.

Table 4.10: Case Execution Length for Plant Performance

Model	Minimum Time (s)	Average Time (s)	Maximum Time (s)
Linear	1.37	1.48	1.58
Random Forest	47.1	51.0	54.4
Extra Tree	32.4	34.6	35.9
k-Nearest Neighbours	0.924	1.00	1.36

Table 4.10 shows that LR and kNN execute faster than RF and ET, making them more suitable for time-sensitive or resource-constrained applications. The analysis supports trade-off decisions between predictive accuracy and computational cost, while also highlighting scalability concerns for high-latency models in real-time plant environments.

5. DISCUSSION

The discussion section comprehensively outlines the research findings and their implications in mineral processing and predictive modelling. Through an in-depth analysis of critical aspects outlined by the research questions, this section interprets the implications, limitations and future directions to answer these questions extensively.

The discussion will first interpret the results to explain how MF2 circuits can be operated with blended feeds, highlighting the 4E product recovery and chromium in the concentrate (%) and how they fit in the context of the process parameter adjustments. Following this, this section will interpret the results of developing ML algorithms with the described features and their practicality in terms of their associated errors. The practicality of these ML algorithms in real-world scenarios is a reassuring factor, providing confidence in their applicability. Lastly, it will discuss how these models fit into industry practices, considering possible uses.

5.1. Plant Performance Evaluation

To address the research question, “Does including Merensky ore in the MF2 concentrator affect circuit performance?” the study investigated the operational impact of blending UG2 with Merensky ore in the MF2 by observing the 4E product recovery (%) and chromium content in the concentrate. These parameters were chosen because they illustrate the classic purity vs. recovery dynamics required to determine the balanced operational curve for MF2 circuits and are highlighted in the TRP documentation as KPIs. The findings significantly improve understanding by quantitatively demonstrating how ore blending alters this trade-off, revealing shifts in recovery efficiency and concentrate grade trends that were not evident from qualitative or historical observations alone. This enables more informed process decisions when integrating Merensky into UG2-dominant circuits.

Based on the results shown in Table 4.2, there is an insignificant operational difference between the UG2 and Merensky-UG2 feeds. However, the 4E product recovery (%) and chromium content in the concentrate in the UG2 is slightly higher than in the Merensky-UG2 feed. Dzvinamurungu

et al. (2013) attribute the UG2's higher recovery to the abundance of pentlandite and pyrrhotite, which naturally float faster. Moreover, the slight increase in the UG2 ore in the chromium in the concentrate is mainly attributed to the abundance of chromium in UG2, as illustrated in Table 2.1. Although the differences in the parameters observed are minimal, they have a significant financial impact. These financial implications underscore the criticality of maximising recovery while adhering to the chromium limitations. The chromium content below 1.5% preserves the smelters' operational integrity by preventing refractory lining damage. Thus, the study provides a data-driven basis for determining an operational envelope where the benefits of higher recovery are not offset by penalties associated with elevated chromium. This nuanced understanding enables more informed control of ore blending strategies to optimise both metallurgical and economic performance in MF2-type concentrators.

The grind data and throughput data were analysed further to understand the operation of the milling and flotation stages. Although King's (2001) comminution and flotation models highlight that the reagent dosing and aeration rates are required for the flotation stage analysis, such data is unavailable; hence, the grind data was studied instead. The grind data can still provide a decent interpretation of the operational procedure, as this provides an idea of the expected mass pull based on the fines. Furthermore, only the grind data from the primary mill was considered, as Corin et al. (2021) state that most of the 4E is recovered in the primary milling and flotation stages. This behaviour is supported by the feature importance of 4E content in the concentrate plot in Figure 4.12 which shows the primary rougher feed as the prominent feature.

The average grind data analysis between the blended and unblended feeds reveals a minor decrease in fines (-75 µm) for UG2 relative to the Merensky-UG2 blend. However, this insignificant difference suggests that the fines are strictly controlled. A similar difference is indicated for the other primary rougher feed grind data. This consistent pattern observed across the grind data strongly suggests that the milling process in this operation is strictly controlled.

To maintain a statistically consistent operational performance in terms of a consistent 4E product recovery, chromium in the concentrate and grind data, the trade-off is the throughput. Hay & Roy

(2010) and Dzyinamurungu et al. (2013) documented this intricate interplay between these process parameters and operational objectives. This is due to the increase in the residence time in the mill to maintain consistent grind data between the blended and unblended ores. The Merensky incorporated ore required extended milling due to the plagioclase content, which absorbs some mill breakage forces (Dzvinamurungu et al., 2013).

The correlation between residence time, grind data (PSD), and PGM concentration (4E product recovery) has been a subject of considerable attention by Dzyinamurungu et al. (2013), Corin et al. (2021), Monele Mokadze et al. (2021) and Hay & Roy (2010) highlighting a negative correlation between the residence time vs throughput and 4E product recovery (%) vs type of ore. This study aligns with these findings, revealing how the incorporation of Merensky-UG2 ore affects circuit performance by diminishing 4E product recovery and necessitating an increase in residence time at the primary milling stage, consequently impacting milled and concentrate throughput. Thus, while including Merensky ore in the MF2 circuit does not cause significant changes to KPIs, it does influence circuit performance in more subtle operational ways. Though not statistically significant, these minor variations in recovery and grind have practical and financial implications.

It is worth noting that this study diverges from Hay & Roy (2010), where UG2 ore was incorporated into Merensky, decreasing residence time. These differences contribute to knowledge and highlight theories that Hay & Roy (2010) and Dzyinamurungu et al. (2013) posit. These results thus add value and meaning to the discourse of ore blending within mineral processing circuits concerning optimising operational efficiency and maximising resource recovery.

5.2. Prediction of Ore Mineralogy

In response to the second research question, "Can ML methods be employed to develop a predictive model for accurately forecasting ore mineralogy in the MF2 concentrator plant?". This work will demonstrate and underline the potential and accuracy of ML algorithms, such as LR, RF, ET, and kNN. The successful application of these models in the present study, rigorously

tested with training and test datasets, demonstrates their potential in predicting performance and areas of improvement by RMSE% % and R^2 metrics. The results varied among the models.

Based on Table 2.1, the difference in the chromium content in the ore is significant. This suggests that the chromium in the concentrate (%) would be an important feature in ore mineralogy. Table 4.2 supports this as the chromium content in the concentrate is higher in the UG2-only processed throughput. The LR model's poor performance score was evidenced on some variables by high RMSE values and a negative R^2 score; thus, it had worse prediction abilities on ore mineralogy. However, RF could better predict the PGM + Au grade with an RMSE of 8.97 % and R^2 of 0.25 for test data. This is due to the PGE differences in UG2 and Merensky, as highlighted by Corin et al. (2021) and Dzyinamurungu et al. (2013). However, Merensky (%) did not emerge as a significant feature, which contradicts expectations. This may be due to the tightly controlled milling operation, where the minor variations in recovery and grind data are statistically insignificant and do not meaningfully influence the chromium content in the concentrate (%). As a result, Merensky (%) was found to have a weak correlation with the target variables. This is supported by Figure 4.2, which indicates a correlation coefficient near 0 with the ore mineralogy. Other variables, such as Pt, Pd, and Rh grades, showed the same trend as PGM + Au grade, for instance, their relationship to chromium content. Moreover, similar performance is explained since the correlation matrix indicates that the ore mineralogy correlates consistently with the primary rougher feed. The Au grade was poorly predicted, which proved that the LR model did not capture this non-linear relationship between Au grade and other factors. Poor predictions were obtained for Ni %, Cu %, Co %, and S %, thus further underpinning the lack of any linear relationship with these targets. Tree-based algorithms, mainly RF, showed better performance compared to LR. RF showed decent prediction capability for PGM + Au, Pt, Pd, Rh, and Cr2O3 %.

This lends credence to the potential of RF in modelling nonlinear trends and relationships concerning plant performance. Indeed, RF has been reported by Rigatti et al. (2017) to cope well with complex data by simply averaging the predictions, yielding an accurate result, a feature displayed in the present work. In contrast, RF suffered from extreme values, mainly in the case of Au grade prediction, which means this method is limited by its capacity to handle data variability.

ET handled nonlinear relationships well but produced higher RMSE and lower R^2 scores for some variables, notably Co % and Cu %. Although RF performed far better than LR, it still did not provide the accuracy noted in ET, proving that more refined decision splits by RF are significant factors in its inferior performance. These are indicated by higher RMSE and lower R^2 scores for some variables; hence, despite holding its advantages, ET ultimately tends to overcome RF barriers. kNN model provided average coverage of about overall moderate, but high variability contributed to its reshaping with an RMSE of 39.1% and negative R^2 scores for Au grade. This is because the kNN model is sensitive to the choice of 'k' and relies upon only the local data points; thus, it might not be appropriate in all scenarios for predicting ore mineralogy. The results obtained have underscored the importance of realising each model's limitations when applying it in practice.

A critical limitation affecting model performance was the low temporal resolution of the dataset. Operational data was collected daily, whereas mineralogical data was only available as weekly composite samples. This mismatch constrained the models' ability to capture short-term variability and limited their responsiveness to dynamic process changes. Enhanced data granularity, particularly through daily sampling and reporting of ore mineralogy, could significantly improve predictive accuracy and model robustness. To answer the second research question, ML methods, particularly tree-based algorithms like RF and ET, can indeed be employed to develop predictive models for forecasting ore mineralogy in the MF2 concentrator, however, their accuracy is constrained by data quality and temporal resolution.

These findings illustrate the potential and limitations of ML models for predicting ore mineralogy. While ET shows the most promise, improvements in data collection practices and model refinement are necessary. With further development, such models could support proactive decision-making in plant operations, enabling real-time adjustments to optimise recovery and maintain concentrate purity, ultimately improving the operational efficiency of the MF2 concentrator.

5.3. Prediction of Plant Performance

This study's central research question is: Can ML techniques be utilised to create a data-based model for accurately predicting MF2 plant performance with blended and unblended feeds? The operational performance of the MF2 concentrator plant is predicted using each of the ML algorithms, which are involved in a comparative analysis of its operational performance. In this part of the work, the potential of the ML algorithms to build predictive models that could be useful to guarantee a correct forecast of the operational performance of the MF2 concentrator plant instils optimism about the future of plant performance prediction.

Only features that would be available to the user were considered to enhance the practicality of using the selected ML algorithms. These features include how the milling operations will be handled and the projected throughput based on available stockpiles. Based on Corin et al. (2021), the primary milling and flotation stages produce a high-grade concentrate, so features that describe these stages should be considered important. This behaviour was exhibited for the 4E content in the concentrate, as the throughput and primary rougher feed emerged as important features. For the chromium in the concentrate, the fines post-milling emerged as important features.

The prediction focused on 4E and chromium in the concentrate through LR, RF, ET, and kNN models. It also gauged the performance of the models against a naïve benchmark. The naïve benchmark fixed the RMSE to the mean of the targets as 16.9% and 16.4% for the 4E and chrome content baselines, respectively. The performance of ML models was better than these benchmark points in many instances. The RF had the lowest RMSE % and R2 values compared to the other models. However, similar to the ore mineralogy, the extreme values were not well captured. While not the top performer, the ET model performed similarly to the RF. The kNN model, in contrast, showed markedly worse results, especially in predicting chromium content. The LR model performed well when predicting 4E content but was poor when predicting the chromium content, as shown by the RMSE % and R2. These challenges were highlighted by Osei (2019), as the LR poorly captured complex data structures.

In evaluating the RF model's performance, it is evident that its enhanced accuracy in predicting 4E and chromium content aligns with the literature, particularly regarding its capability to manage nonlinear relationships and complex variable interactions. However, while RF excels in these areas, it struggles with handling extreme values, leaving considerable room for improvement. This suggests that although RF performs well with complex data, its ability to manage variability may be lacking. The ET model, while similar to RF in handling nonlinear relationships, exhibited lower accuracy. This highlights ET's potential benefits and limitations compared to RF, emphasising the need for further refinement to address these models' weaknesses. The answer to the final research question, ML techniques, especially RF and ET. can be effectively used to predict MF2 plant performance with both blended and unblended feeds, although their accuracy is influenced by the models' ability to handle extreme values and the complexity of operational variables. To improve the model's performance, according to Osei (2019), more data on operational factors such as pulp pH, reagents, air flow rate and levels in the bank may potentially enhance the predictive accuracy of this work. If such data had existed in this study, it could have provided an entirely detailed insight into plant performance.

5.4. Execution Time Analysis of Predictive Models in Plant Performance

Experiments for real-time prediction were conducted using the time function in Python on a laptop equipped with an 11th Generation Intel(R) Core (TM) i5-1155G7 processor, 8 GB RAM, 512 GB SSD, and a central processing unit running at 2.50 GHz on a 64-bit Windows 11 system. With this computer, every prediction was repeated ten times to get proper results and to define minimum, mean, and maximum execution times.

This iterative exercise ensured that none of the operating system routines would intervene in the results obtained from the experiment. The tests indicated that for the most extended run, the RF had a maximum run time of 51 seconds on plant performance. The ET model only required up to 35 seconds for related computations in plant performance. This represents the advantage of using an ET model instead of RF in computational speed.

However, the RF model's running time remained within a 24-hour cutoff, taken as a passing test. This efficiency reassures operators in an MF2 plant, even during the most time-critical applications, though at the cost of some accuracy.

Even though the results mentioned above are statistically significant, using a prior knowledge-based model based on historical data alone significantly limits its predictive capability for adaptability and responsiveness to dynamic changes in the process conditions. The absence of a real-time monitoring system further limits the capabilities of predictive models. This, in turn, demands constant development in the usual predictor models via research. Future research should be directed toward involving real-time data acquisition systems and implementing advanced modelling tools with sophisticated control strategies so that the performance of models increases to a higher level of accuracy.

6. CONCLUSION

While introducing Merensky ore into the MF2 concentrator should influence the performance, this study determined that blending UG2 with Merensky in the MF2 concentrator has an insignificant operational difference regarding the 4E product recovery and chromium content in the concentrate. However, the UG2 ore slightly outperforms the Merensky-UG2 blend regarding the 4E product recovery. The chromium content in the concentrate was marginally lower in the Merensky-UG2 blend. This is primarily attributed to the ore mineralogical differences as UG2 has a higher PGE grade than Merensky, while UG2 has a higher chromium (%). Although the differences are insignificant statistically, they have significant financial implications.

To achieve this behaviour, the milling operation is strictly controlled due to the similarity in the grind data of the two operational modes. However, incorporating Merensky ore into the UG2 feed requires extended residence time due to its high plagioclase content, which affects the plant throughput. This behaviour aligns with previous literature, highlighting the trade-off between residence time, grind data, and throughput when blending these ores.

While insights into the operational performance of MF2 with UG2 and Merensky-UG2 ores are mainly attributed to the differences in the ore composition, using these insights to build and evaluate ML algorithms that examine operational performance and predict ore composition would be an exciting feat. This study examined various ML algorithms such as LR, RF, ET and kNN to predict the ore mineralogy based on the plant performance data. The ET model, on average, demonstrated the best performance. This is attributed to the algorithm's ability to handle nonlinear relationships. However, the model failed to capture extreme values, particularly in predicting the Au grade. The LR algorithm performed poorly due to its inability to capture non-linear relationships, rendering it the least effective model for ore mineralogy prediction.

During the feature selection phase, the chromium content in the concentrate was determined to be a significant feature. Considering the plant performance data analysis, this conclusion is expected, thus reinforcing the importance of considering it in future models.

Each model has its limitations, with the LR struggling with nonlinear relationships, kNN struggling with its sensitivity to local points and the RF struggling due to its more refined splits. Despite the ET's struggles, more granular daily data collection could lead to better predictive accuracy.

The same algorithms were used in ore mineralogy to predict the plant performance regarding 4E and chromium content in the concentrate. The features selected for this exercise were those anticipated to be accessible to the user at the time of prediction. Overall, all the ML algorithms outperformed the naïve benchmark test of assuming a mean prediction irrespective of the changes to the process parameters. The RF indicated an improvement over the ET in accuracy; however, neither model captured the extreme values and variance. The kNN indicated a poor performance, especially in predicting the chromium content in the concentrate.

Although the RF outperformed the other algorithms, its shortcomings, according to Osei (2019), are attributed to the lack of data that describes the flotation stage. This includes operational factors such as pulp pH and reagent addition rates that may potentially enhance the predictive accuracy of this work. Though the RF is computationally less efficient than the ET, the run time of 51 seconds outperforms the current 24 hours required to determine the plant performance.

7. REFERENCES

Anandhi, P., & Nathiya, Dr. E. (2023). Application of linear regression with their advantages, disadvantages, assumption and limitations. *International Journal of Statistics and Applied Mathematics*, *8*(6), 133–137. https://doi.org/10.22271/maths.2023.v8.i6b.1463

Bagchi, I. N. (2023). Coal beneficiation: theory and practice. In *Mineral Processing* (pp. 1–53). Elsevier. https://doi.org/10.1016/B978-0-12-823149-4.00007-7

Barker, M., & Rayens, W. (2003). Partial least squares for discrimination. *Journal of Chemometrics*, *17*(3), 166–173. https://doi.org/10.1002/cem.785

Barreñada, L., Dhiman, P., Timmerman, D., Boulesteix, A.-L., & Van Calster, B. (2024). *Understanding random forests and overfitting: a visualization and simulation study*.

Belete, D. M., & Huchaiah, M. D. (2022). Grid search in hyperparameter optimization of machine learning models for prediction of HIV/AIDS test results. *International Journal of Computers and Applications*, *44*(9), 875–886. https://doi.org/10.1080/1206212X.2021.1974663

Brooks, K., Westcott, M., & Bauer, M. (2019). *A Combined MPC for Milling and Flotation-A Simulation Study*. https://www.researchgate.net/publication/335527791

Choi, R. Y., Coyner, A. S., Kalpathy-Cramer, J., Chiang, M. F., & Peter Campbell, J. (2020). Introduction to machine learning, neural networks, and deep learning. *Translational Vision Science and Technology*, *9*(2). https://doi.org/10.1167/tvst.9.2.14

Corin, K. C., McFadzean, B. J., Shackleton, N. J., & O'Connor, C. T. (2021). Challenges related to the processing of fines in the recovery of platinum group minerals (PGMs). In *Minerals* (Vol. 11, Issue 5). MDPI AG. https://doi.org/10.3390/min11050533

Costello, F., & Watts, P. (2022). *Why bother with Bayesian t-tests?* http://arxiv.org/abs/2211.02613

Danha, G., Hildebrandt, D., Glasser, D., & Bhondayi, C. (2015). Application of basic process modelling in investigating the breakage behaviour of UG2 ore in wet milling. *Powder Technology*, *279*, 42–48. https://doi.org/10.1016/j.powtec.2015.03.034

Demšar, U., Harris, P., Brunsdon, C., Fotheringham, A. S., & McLoone, S. (2013). Principal Component Analysis on Spatial Data: An Overview. *Annals of the Association of American Geographers*, *103*(1), 106–128. https://doi.org/10.1080/00045608.2012.689236

Di Franco, G., & Santurro, M. (2021). Machine learning, artificial neural networks and social research. *Quality and Quantity*, *55*(3), 1007–1025. https://doi.org/10.1007/s11135-020-01037-y

Dzvinamurungu, T., Viljoen, K. S., Knoper, M. W., & Mulaba-Bafubiandi, A. (2013). Geometallurgical characterisation of Merensky Reef and UG2 at the Marikana Mine, Bushveld Complex, South Africa. *Minerals Engineering*, *52*, 74–81. https://doi.org/10.1016/j.mineng.2013.04.010

Evans, C. L., Coulter, B. L., Wightman, E., & Burrows, A. S. (2009). *Improving Energy Efficiency Across Mineral Processing and Smelting Operations-A New Approach*.

Fernandes, M., Canito, A., Bolón-Canedo, V., Conceição, L., Praça, I., & Marreiros, G. (2019). Data analysis and feature selection for predictive maintenance: A case-study in the metallurgic industry. *International Journal of Information Management*, *46*, 252–262. https://doi.org/10.1016/j.ijinfomgt.2018.10.006

Fuerstenau, D. W., De, A., & Kapur, P. C. (2004). Linear and nonlinear particle breakage processes in comminution systems. *International Journal of Mineral Processing*, *74*(SUPPL.). https://doi.org/10.1016/j.minpro.2004.07.005

Hao, J., & Ho, T. K. (2019). Machine Learning Made Easy: A Review of Scikit-learn Package in Python Programming Language. In *Journal of Educational and Behavioral Statistics* (Vol. 44, Issue 3, pp. 348–361). SAGE Publications Inc. https://doi.org/10.3102/1076998619832248

Hay, M. P., & Roy, R. (2010). A case study of optimising UG2 flotation performance. Part 1: Bench, pilot and plant scale factors which influence Cr2O3 entrainment in UG2 flotation. *Minerals Engineering*, *23*(11–13), 855–867. https://doi.org/10.1016/j.mineng.2010.05.002

Hodouin, D., Jamsa-Jounela, S.-L., Carvalho, M. T., & Bergh, L. (2001). State of the art and challenges in mineral processing control. In *Control Engineering Practice* (Vol. 9).

Hosseini, K., Keramati, M. A., Ali, M., Kazemi, A., & Fathi, Z. (2025). Comparison of the accuracy of Box-Jenkins and Holt-Winters methods for forecasting OPEC crude oil price. *Int. J. Nonlinear Anal. Appl*, *16*, 2008–6822. https://doi.org/10.22075/ijnaa.2024.32531.4839

Huffaker, R., Bittelli, M., & Rosa, R. (2018). *Phenomenological Modelling* (Vol. 1). Oxford University Press. https://doi.org/10.1093/oso/9780198782933.003.0009

Jadhav, S. D., & Channe, H. P. (2013). Comparative Study of K-NN, Naive Bayes and Decision Tree Classification Techniques. In *International Journal of Science and Research (IJSR) ISSN* (Vol. 5). www.ijsr.net

Jearanaitanakij, K. (1999). *Exploring and Implementing the "Monitoring of a Multi-stage Manufacturing Process" using Dynamic Bayesian Networks*. https://www.researchgate.net/publication/2508779

Ji, W., Yin, S., & Wang, L. (2019). A big data analytics-based machining optimisation approach. *Journal of Intelligent Manufacturing*, *30*(3), 1483–1495. https://doi.org/10.1007/s10845-018-1440-9

Kadlec, P., Grbić, R., & Gabrys, B. (2011). Review of adaptation mechanisms for data-driven soft sensors. In *Computers and Chemical Engineering* (Vol. 35, Issue 1, pp. 1–24). https://doi.org/10.1016/j.compchemeng.2010.07.034

Kagalwala, A. (2022). kpsstest: A command that implements the Kwiatkowski, Phillips, Schmidt, and Shin test with sample-specific critical values and reports p-values. *Stata Journal*, *22*(2), 269–292. https://doi.org/10.1177/1536867X221106371

Kalashnikov, A. O., Pakhomovsky, Y. A., Bazai, A. V., Mikhailova, J. A., & Konopleva, N. G. (2021). Rock-chemistry-to-mineral-properties conversion: Machine learning approach. *Ore Geology Reviews, 136*. https://doi.org/10.1016/j.oregeorev.2021.104292

King, R. P. (2001). *Modelling and Simulation of Mineral Processing Systems*. Butterworth-Heinemann.

Kitagawa, J., Tsubota, M., Wang, S., & Lu, G. (2018). *Noble and Precious Metals Properties, Nanoscale Effects and Applications* (M. S. Seehra & A. Bristow, Eds.). IntechOpen.

Kuang, Q., & Zhao, L. (2009). A Practical GPU Based KNN Algorithm. *The 2009 International Symposium on Computer Science and Computational Technology*, 151–155.

Lamberg, P., Rosenkranz, J., Wanhainen, C., Lund, C., Minz, F., Mwanga, A., & Amiri Parian, M. (2013). *Building a Geometallurgical Model in Iron Ores using a Mineralogical Approach with Liberation Data*.

Liashchynskyi, P., & Liashchynskyi, P. (2019). *Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS*. http://arxiv.org/abs/1912.06059

Ling, Y., & Mahadevan, S. (2013). Quantitative model validation techniques: New insights. *Reliability Engineering and System Safety*, *111*, 217–231. https://doi.org/10.1016/j.ress.2012.11.011

Lishchuk, V., Lamberg, P., & Lund, C. (2016). *Geometallurgical programs-critical evaluation of applied methods and techniques PREP*. www.ltu.se

M. Mijwil, M. (2021). Artificial Neural Networks Advantages and Disadvantages. *Mesopotamian Journal of Big Data*, *2021*, 29–31. https://doi.org/10.58496/mjbd/2021/006

Mahesh, B. (2018). Machine Learning Algorithms-A Review. *International Journal of Science and Research*. https://doi.org/10.21275/ART20203995

Maier, W. D., & Bowen, M. P. (1996). *MINERALIUM DEPOSITA The UG2-Merensky Reef interval of the Bushveld Complex northwest of Pretoria* (Vol. 31).

Mastorakis, G. (2018). *Human-like machine learning: limitations and suggestions*. http://arxiv.org/abs/1811.06052

McCoy, J. T., & Auret, L. (2019). Machine learning applications in minerals processing: A review. In *Minerals Engineering* (Vol. 132, pp. 95–109). Elsevier Ltd. https://doi.org/10.1016/j.mineng.2018.12.004

McDonald, I., Holwell, D. A., & Armitage, P. E. B. (2005). Geochemistry and mineralogy of the Platreef and "Critical Zone" of the northern lobe of the Bushveld Complex, South Africa: Implications for Bushveld stratigraphy and the development of PGE mineralisation. *Mineralium Deposita*, *40*(5), 526–549. https://doi.org/10.1007/s00126-005-0018-6

McGahey, S., & Cameron, I. (2002). Model Transformations in Multi Scale Modelling. In *Computer Aided Chemical Engineering* (Vol. 10, pp. 925–930). . https://doi.org/10.1016/S1570-7946(02)80182-1

Menze, B. H., Kelm, B. M., Masuch, R., Himmelreich, U., Bachert, P., Petrich, W., & Hamprecht, F. A. (2009). A comparison of random forest and its Gini importance with standard chemometric methods for the feature selection and classification of spectral data. *BMC Bioinformatics*, *10*. https://doi.org/10.1186/1471-2105-10-213

Monele Mokadze, A., Ndlovu, S., Shemi, A., & Dworzanowski, M. (2021). The Reduction of Chrome in UG-2 Flotation Concentrate by Hydrometallurgical Means. *International Journal of Mineral Processing and Extractive Metallurgy*, *6*(3), 41. https://doi.org/10.11648/j.ijmpem.20210603.12

Montgomery, J. M., & Olivella, S. (2018). Tree-Based Models for Political Science Data. *American Journal of Political Science*, *62*(3), 729–730. https://doi.org/10.7910/DVN/8ZJBLI

Muraina, I. O. (2022). *IDEAL DATASET SPLITTING RATIOS IN MACHINE LEARNING ALGORITHMS: GENERAL CONCERNS FOR DATA SCIENTISTS AND DATA ANALYSTS*. https://www.researchgate.net/publication/358284895

Mutasa, S., Sun, S., & Ha, R. (2020). Understanding artificial intelligence based radiology studies: What is overfitting? In *Clinical Imaging* (Vol. 65, pp. 96–99). Elsevier Inc. https://doi.org/10.1016/j.clinimag.2020.04.025

Mutinda, J. K., & Geletu, A. (2025). Stock Market Index Prediction Using CEEMDAN-LSTM-BPNN-Decomposition Ensemble Model. *Journal of Applied Mathematics*, *2025*(1). https://doi.org/10.1155/jama/7706431

Mwanga, A., Rosenkranz, J., & Lamberg, P. (2015). Testing of ore comminution behaviour in the geometallurgical context—A review. In *Minerals* (Vol. 5, Issue 2, pp. 276–297). MDPI AG. https://doi.org/10.3390/min5020276

Nel, E., Theron, J., Martin, C., & Raabe, H. (2004). *PGM Ore Processing at Impala's UG-2 Concentrator in Rustenburg, South Africa*.

Neumann, G. A., Finkler, T. F., Cardozo, N. S. M., & Secchi, A. R. (2006). Comparison between Phenomenological and Empirical Models for Gas-Phase Polymerization Process Control. *Industrial & Engineering Chemistry Research*, *45*(8), 2651–2660. https://doi.org/10.1021/ie050595c

Nguyen, Q. H., Ly, H. B., Ho, L. S., Al-Ansari, N., Van Le, H., Tran, V. Q., Prakash, I., & Pham, B. T. (2021). Influence of data splitting on performance of machine learning models in prediction of shear strength of soil. *Mathematical Problems in Engineering*, *2021*. https://doi.org/10.1155/2021/4832864

Osei, E. K. (2019). *Machine Learning-based Quality Prediction in the Froth Flotation Process of Mining*.

Pandey, A., & Jain, A. (2017). Comparative Analysis of KNN Algorithm using Various Normalization Techniques. *International Journal of Computer Network and Information Security*, *9*(11), 36–42. https://doi.org/10.5815/ijcnis.2017.11.04

Polamuri, S. R., Srinivas, K., & Krishna Mohan, A. (2019). Stock market prices prediction using random forest and extra tree regression. *International Journal of Recent Technology and Engineering*, *8*(3), 1224–1228. https://doi.org/10.35940/ijrte.C4314.098319

Polat, M., & Chander, S. (2000). First-order flotation kinetics models and methods for estimation of the true distribution of flotation rate constants. *International Journal of Mineral Processing*, *58*(1–4), 145–166. https://doi.org/10.1016/S0301-7516(99)00069-1

Poole, D., & Mackworth, A. (2017). Artificial Intelligence: Foundations of Computational Agents 2nd Edition. In *Supervised Machine Learning* (2nd ed., pp. 267–343). Cambridge University Press; 2nd edition.

Probst, P., Wright, M. N., & Boulesteix, A. L. (2019). Hyperparameters and tuning strategies for random forest. In *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* (Vol. 9, Issue 3). Wiley-Blackwell. https://doi.org/10.1002/widm.1301

Qu, Z., Zhang, X., Gao, Y., Peng, C., Wang, Y., & Georgievitch, P. M. (2023). Detection of False Data Injection Attack in AGC System Based on Random Forest. *Machines*, *11*(1). https://doi.org/10.3390/machines11010083

Rajasekar, V., Premalatha, J., & Dhanaraj, R. K. (2022). Security analytics. In *System Assurances* (pp. 333–354). Elsevier. https://doi.org/10.1016/B978-0-323-90240-3.00019-9

Rigatti, S. J. (2017). Random Forest. In *JOURNAL OF INSURANCE MEDICINE Copyright C 2017 Journal of Insurance Medicine J Insur Med* (Vol. 47). http://meridian.allenpress.com/jim/article-pdf/47/1/31/1736157/insm-47-01-31-39_1.pdf

Rosenbaum, T., Tan, L., & Engstrom, J. (2019). Advantages of utilizing population balance modeling of crystallization processes for particle size distribution prediction of an active pharmaceutical ingredient. *Processes*, *7*(6). https://doi.org/10.3390/pr7060355

Rule, C. (2011). *INTERNATIONAL AUTOGENOUS GRINDING SEMIAUTOGENOUS GRINDING AND HIGH PRESSURE GRINDING ROLL TECHNOLOGY 2011 EDITORS*.

Rule, C. M. (2009). *Energy considerations in the current PGM processing flowsheet*.

Sarstedt, M., Ringle, C. M., & Hair, J. F. (2014). PLS-SEM: Looking Back and Moving Forward. In *Long Range Planning* (Vol. 47, Issue 3, pp. 132–137). Elsevier Ltd. https://doi.org/10.1016/j.lrp.2014.02.008

Souza, D. V., Nievola, J. C., Corte, A. P. D., & Sanquetta, C. R. (2020). k-Nearest Neighbor And Linear Regression In The Prediction Of The Artificial Form Factor. *Floresta*, *50*(3), 1669–1678. https://doi.org/10.5380/rf.v50i3.65720

Suyal, M., & Goyal, P. (2022). A Review on Analysis of K-Nearest Neighbor Classification Machine Learning Algorithms based on Supervised Learning. In *International Journal of Engineering Trends and Technology* (Vol. 70, Issue 7, pp. 43–48). Seventh Sense Research Group. https://doi.org/10.14445/22315381/IJETT-V70I7P205

Switzer, A. D. (2013). Measuring and Analysing Particle Size in a Geomorphic Context. In *Treatise on Geomorphology: Volume 1-14* (Vols. 1–14, pp. 224–242). Elsevier. https://doi.org/10.1016/B978-0-12-374739-6.00385-7

Szalavetz, A. (2022). The digitalisation of manufacturing and blurring industry boundaries. *CIRP Journal of Manufacturing Science and Technology*, *37*, 332–343. https://doi.org/10.1016/j.cirpj.2022.02.015

Tarakci, F., & Ozkan, A. (2021). Comparison of classification performance of kNN and WKNN algorithms. *Selcuk University Journal of Engineering Sciences*, *20*(02), 32–37. http://sujes.selcuk.edu.tr/sujes

Totis, N., Nieto, C., Kuper, A., Vargas-Garcia, C., Singh, A., & Waldherr, S. (2021). A Population-Based Approach to Study the Effects of Growth and Division Rates on the Dynamics of Cell Size Statistics. *IEEE Control Systems Letters*, *5*(2), 725–730. https://doi.org/10.1109/LCSYS.2020.3005069

Tran, M. K., Panchal, S., Chauhan, V., Brahmbhatt, N., Mevawalla, A., Fraser, R., & Fowler, M. (2022). Python-based scikit-learn machine learning models for thermal and electrical performance prediction of high-capacity lithium-ion battery. *International Journal of Energy Research*, *46*(2), 786–794. https://doi.org/10.1002/er.7202

Tutorials Point. (2019). *Scikit-Learn i*. https://www.tutorialspoint.com/scikit_learn/scikit_learn_tutorial.pdf

Von Gruenewaldt, G., Sharpe, M. R., & Hatton, C. J. (1985). The Bushveld complex: introduction and review. *Economic Geology*, *80*(4), 803–812. https://doi.org/10.2113/gsecongeo.80.4.803

Wang, H., & Hu, D. (2005). *Comparison of SVM and LS-SVM for Regression*.

Wills, B. A., & Finch, J. A. (2016a). Wills' Mineral Processing Technology - An Introduction to the Practical Aspects of Ore Treatment and Mineral Recovery (8th Edition) - 1.5 The Need for Mineral Processing Chapter 1. *Elsevier*, *8th Edition*. https://app.knovel.com/hotlink/pdf/id:kt00URXIJ1/wills-mineral-processing/need-mineral-processing

Wills, B. A., & Finch, J. A. (2016b). *Wills' Mineral Processing Technology - An Introduction to the Practical Aspects of Ore Treatment and Mineral Recovery (8th Edition) - Modelling and Characterisation Chapter 17*.

Yianatos, J., Vallejos, P., Grau, R., & Yañez, A. (2020). New approach for flotation process modelling and simulation. *Minerals Engineering*, *156*. https://doi.org/10.1016/j.mineng.2020.106482

Zhang, J., Zulkernine, M., & Haque, A. (2008). Random-forests-based network intrusion detection systems. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, *38*(5), 649–659. https://doi.org/10.1109/TSMCC.2008.923876

Zhang, W. (2017). Machine Learning Approaches to Predicting Company Bankruptcy. *Journal of Financial Risk Management*, *06*(04), 364–374. https://doi.org/10.4236/jfrm.2017.64026

Zou, Z. M., Chang, D. H., Liu, H., & Xiao, Y. D. (2021). Current updates in machine learning in the prediction of therapeutic outcome of hepatocellular carcinoma: what should we know? In *Insights into Imaging* (Vol. 12, Issue 1). Springer Science and Business Media Deutschland GmbH. https://doi.org/10.1186/s13244-021-00977-9

APPENDICES

APPENDIX A: Python Script for Data Cleaning

This section contains the Python code for cleaning and preparing the data for model development.

A1. Library

In [1]:

```
# Installed libraries


# !pip install autoviz
# !pip install yfinance
# !pip install tensorflow
# !pip install -U tensorflow scikit-learn
# !pip install --upgrade tensorflow
# !pip install keras
# !pip uninstall tensorflow
# !pip install tensorflow
# !pip install hyperopt
# !pip install xgboost
# !pip install scikit-optimize
# !pip install tpot==0.11.7
# !pip install opencv-python
# !pip install mlxtend
# !pip install imblearn
# !pip uninstall imbalanced-learn
# !pip install imbalanced-learn
# !pip install --user --upgrade scikit-learn
# !pip install pandoc
# !pip install pypandoc
# !pip install nbconvert
```

In [2]:

```
# Libraries required

# Math functions
import pandas as pd
import math

# Data Manipulation
import numpy as np

# Data Visualisation
%matplotlib inline
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
# from autoviz.AutoViz_Class import AutoViz_Class

# Statistics
import statistics as st
from statsmodels.tsa.stattools import kpss
from statsmodels.tsa.stattools import adfuller
import statsmodels.api as sm
from statsmodels.stats.diagnostic import het_breuschpagan

# Generating Random Numbers
import random

# Download Historical Market Data
import yfinance as yf

# Machine Learning
import tensorflow as tf

import os
import textwrap

# Ignore Warnings
import warnings
warnings.filterwarnings("ignore")
```

In [3]:

```
# Scikit-learn

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.impute import KNNImputer
from sklearn.preprocessing import LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.svm import OneClassSVM
from sklearn.ensemble import RandomForestRegressor
from sklearn.feature_selection import SelectFromModel
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import FunctionTransformer
from sklearn.linear_model import LogisticRegressionCV
from sklearn.linear_model import LassoCV
from sklearn.linear_model import RidgeCV
from sklearn.pipeline import make_pipeline
from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.svm import SVR
from sklearn.preprocessing import MinMaxScaler
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
from sklearn.preprocessing import RobustScaler
from sklearn.ensemble import IsolationForest
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import RFE
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.model_selection import KFold
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from sklearn.model_selection import cross_val_predict
from sklearn.utils import resample
from sklearn.metrics import explained_variance_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report
from sklearn.preprocessing import label_binarize
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.dummy import DummyClassifier

# Tensorflow

from sklearn.base import BaseEstimator, RegressorMixin
from keras.callbacks import LearningRateScheduler
from scipy.stats import randint, uniform
from scipy.stats import boxcox, normaltest
from scipy.stats import yeojohnson
from scipy.stats import shapiro
from scipy.stats import skew
import scipy.stats as stats
from scipy.stats import gaussian_kde
from scipy.ndimage import gaussian_filter1d
from scipy.stats import randint as sp_randint
from scipy.signal import butter, filtfilt


from hyperopt import fmin, tpe, hp, STATUS_OK, Trials
from hyperopt import space_eval

# XGBoost Classifier

from xgboost import XGBClassifier
from xgboost import XGBRegressor
import xgboost as xgb


from tabulate import tabulate
from io import BytesIO
import time
```

A2. Data Collection and Formatting

In [4]:

```
# Read the data into a pandas dataframe for analysis and model creation
path = r'C:\Users\mneno\Downloads\Python Code\\'

TRP_data = pd.read_csv(path + 'Two Rivers Combined Dataframe.csv', encoding='latin1')
```

In [5]:
```
# Rename the column name that is incorrectly spelled
TRP_data = TRP_data.rename(columns={'4E Product Recvoery (%)': '4E Product Recovery (%)'})
```

In [6]:
```
# Creation of a new feature that accounts for the influence of Merensky
TRP_data['Merensky (%)'] = TRP_data['Total Merensky Milled Throughput (tons/day)'] / TRP_data['Total Milled Throughput (tons/day)']
```

In [7]:
```
# Ideally we want all the data to be float data type for analysis

column_names = TRP_data.columns[1:]

# Convert columns to numerical
for column_name in column_names:
    try:
        TRP_data[column_name] = pd.to_numeric(TRP_data[column_name], errors="coerce")
    except KeyError:
        print(f"Column '{column_name}' not found in DataFrame.")
    except pd.errors.ConversionError:
        print(f"Error converting column '{column_name}'. Check for non-numeric values.")

# Convert the 'Date' column to datetime type
try:
    TRP_data['Date'] = pd.to_datetime(TRP_data['Date'])
except pd.errors.OutOfBoundsDatetime:
    print("Error converting 'Date' column to datetime. Check for valid date values.")
```

In [8]:
```
# Split the data into Merensky and UG2 using indexing for hypothesis testing
TRP_data_Mer = TRP_data[TRP_data['Total Merensky Milled Throughput (tons/day)'] > 0]
TRP_data_UG2 = TRP_data[~(TRP_data['Total Merensky Milled Throughput (tons/day)'] > 0)]
```

A3. Time Series Testing

For this section, the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) Test and Augmented Dickey-Fuller (ADF) Tests were performed to determine whether time or season has an influence on the plant performance.

In [9]:
```
# Kwiatkowski-Phillips-Schmidt-Shin (KPSS) Test

# flatten the data
time_series_data = TRP_data['Total Milled Throughput (tons/day)'].values

kpss_stat, p_value, lags, critical_values = kpss(time_series_data)

# Print the results
print(f'KPSS Statistic: {kpss_stat}')
print(f'p-value: {p_value}')
print('Critical Values:')
for key, value in critical_values.items():
    print(f'   {key}: {value}')
```

In [10]:
```
# Interpret the results
if p_value < 0.05:
    print('Reject the null hypothesis (non-stationary). This suggests that the data'
          ' could not be a time series data.')
else:
    print('Fail to reject the null hypothesis (stationary). This suggests that the data'
          ' could be a time series data.')
```

In [11]:
```
# Augmented Dickey-Fuller (ADF) Test

# Perform the ADF test
adf_stat, p_value, used_lag, nobs, critical_values, icbest = adfuller(time_series_data)

# Print the results
print(f'ADF Statistic: {adf_stat}')
print(f'p-value: {p_value}')
print(f'Used Lag: {used_lag}')
print(f'Number of Observations: {nobs}')
print('Critical Values:')
for key, value in critical_values.items():
    print(f'   {key}: {value}')
```

In [12]:
```
# Interpret the results
if p_value < 0.05:
    print('Reject the null hypothesis (non-stationary). This suggests that the data '
          'could not be a time series data.')
else:
    print('Fail to reject the null hypothesis (stationary). This suggests that the data '
          'could be a time series data.')
```

In [13]:
```
# Time Series Plot

fig = plt.figure(figsize = (10, 5))

# creating the bar plot
plt.bar(TRP_data['Date'], TRP_data['Total Milled Throughput (tons/day)']/TRP_data['Total Milled
Throughput (tons/day)'].max(), color = 'grey', width = 1)

# Add labels and title
plt.xlabel("Date")
plt.ylabel("Total Milled Throughput Ratio")
plt.savefig(path + 'Total_Milled_Tons_distribution.png', bbox_inches='tight')
plt.show()
```

A4. Data Preparation

A4.1. Removal of Empty Columns

In this step, columns that contain no data (empty columns) will be identified and subsequently removed from the dataset to enhance data cleanliness and efficiency.

In [14]:
```
data_empty_cols = [
                    "Repeats_PRF", "Repeats_SRT", "Repeats_FC", "IRS-AB_Cr2O3",
                    "IRS-AB_4E", "IRS-AB_6E", "Repeats_Chrome",
                    '4E Final Tailings Correction Factor',
                    '6E Final Tailings Correction Factor'
]

TRP_data_no_empty_cols = TRP_data.drop(columns=data_empty_cols)
```

A4.2. Removal of Columns with weak correlations

Columns displaying a weak correlation with other columns will be eliminated from the dataset.

In [15]:
```
# Selecting only numeric columns (int64 and float64 types) from TRP_data_no_empty_cols
TRP_data_numeric = TRP_data_no_empty_cols.select_dtypes(include=['int64', 'float64'])

# Extracting the 'Date' column from TRP_data_no_empty_cols
TRP_data_date = TRP_data_no_empty_cols[['Date']]

# Calculating Pearson correlation matrix for numeric columns
pearson_corr_matrix = TRP_data_numeric.corr(method='pearson')

# Setting the correlation threshold
correlation_threshold = 0.3

# Initializing list to store columns to drop from TRP_data_numeric
columns_to_drop = []

# Iterating through columns in the correlation matrix
for column in pearson_corr_matrix.columns:
    # Finding the maximum absolute correlation (excluding self-correlation)
    max_corr = pearson_corr_matrix[column].abs().sort_values(ascending=False).iloc[1]

    # Checking if the max correlation is below the threshold
    if max_corr < correlation_threshold:
        columns_to_drop.append(column)

# Dropping columns with weak correlation from TRP_data_numeric
TRP_data_strong_corr = TRP_data_numeric.drop(columns=columns_to_drop)

# Adding the 'Date' column back to TRP_data_strong_corr
TRP_data_strong_corr = pd.concat([TRP_data_strong_corr, TRP_data_date], axis=1)

# Plotting correlation heatmap for TRP_data_strong_corr
plt.figure(figsize=(30, 25))
sns.heatmap(TRP_data_strong_corr.corr(method='pearson').abs(), annot=True, cmap="viridis")
plt.savefig(path + 'correlation heat map.png', bbox_inches='tight')
plt.show()

# Printing removed columns from TRP_data_numeric
print("The following columns have been removed from TRP_data_numeric to create
TRP_data_strong_corr:")
print(columns_to_drop)

TRP_data_strong_corr.shape
```

A4.3. Removal of NaN entries

Entries with NaN in any of the features will be eliminated from the dataset.

In [16]:
```
# Copy dataframe
TRP_data_nan_removed = TRP_data_strong_corr.copy()

# dropping rows with NaN
cols_excluding_meren_ratio = TRP_data_nan_removed.columns[:-2]

TRP_data_nan_removed = TRP_data_nan_removed.dropna(subset=cols_excluding_meren_ratio,
how='any')

TRP_data_nan_removed['Merensky (%)'] = TRP_data_nan_removed['Total Merensky Milled Throughput
(tons/day)'] / TRP_data_nan_removed['Total Milled Throughput (tons/day)']
```

In [17]:
```
# Ideally we want all the data to be float data type
column_names = TRP_data_nan_removed.columns[1:]

# Convert columns to numeric
for column_name in column_names:
    try:
        TRP_data_nan_removed[column_name] = pd.to_numeric(TRP_data_nan_removed[column_name],
errors="coerce")
    except KeyError:
        print(f"Column '{column_name}' not found in DataFrame.")
    except pd.errors.ConversionError:
        print(f"Error converting column '{column_name}'. Check for non-numeric values.")

# Convert the 'Date' column to datetime
try:
    TRP_data_nan_removed['Date'] = pd.to_datetime(TRP_data_nan_removed['Date'])
except pd.errors.OutOfBoundsDatetime:
    print("Error converting 'Date' column to datetime. Check for valid date values.")

TRP_data_nan_removed.shape
```

A4.4. Removal of Negative values

Entries with negative values in any of the features will be eliminated from the dataset.

In [18]:
```
# Copy dataframe
TRP_data_no_neg_values = TRP_data_nan_removed.copy()

# Remove all the negatives

# Convert the "Date" column to float
TRP_data_no_neg_values["Date"] = pd.to_numeric(TRP_data_no_neg_values["Date"], errors='coerce')

# Exclude Merensky column from the operation
TRP_data_no_neg_values = TRP_data_no_neg_values.drop(columns=["Total Merensky Milled Throughput
(tons/day)"])

# Remove negative values from all columns except "Date" and "Merensky"
TRP_data_no_neg_values = TRP_data_no_neg_values.mask(TRP_data_no_neg_values < 0)

# Convert the "Date" column back to datetime
TRP_data_no_neg_values["Date"] = pd.to_datetime(TRP_data_no_neg_values["Date"], errors='coerce')

# Remove rows with NaN values
TRP_data_no_neg_values = TRP_data_no_neg_values.dropna(axis=0, how='any')
```

In [19]:
```
TRP_data_selected_feat = TRP_data_no_neg_values.copy()

features_and_target = ['Date', 'Total Crushed Throughput (tons/day)', 'Total Milled Throughput (tons/day)',
                'Total Merensky Milled Throughput (tons/day)', '6E Stock (tons)',
                'Chromium Content in Concentrate (%)','Primary Rougher Feed +150 µm Size Class',
                'Primary Rougher Feed +106 µm Size Class', 'Primary Rougher Feed +75 µm Size Class',
                'Primary Rougher Feed -75 µm Size Class', 'Secondary Rougher Tailings +150 µm Size Class',
                'Secondary Rougher Tailings +106 µm Size Class', 'Secondary Rougher Tailings +75 µm Size
Class',
                'Secondary Rougher Tailings -75 µm Size Class', 'Secondary Mill Discharge +150 µm Size
Class',
                'Secondary Mill Discharge +106 µm Size Class', 'Secondary Mill Discharge +75 µm Size
Class',
                'Secondary Mill Discharge -75 µm Size Class', 'Merensky (%)', '4E Dispatched Concentrate
Grade (g/t)',
                "4E Product Recovery (%) calc", "4E flow (ton/day)", 'PGM+Au Grade in Primary Rougher
Feed (g/t)',
                'Pt Grade in Primary Rougher Feed (g/t)',
                'Pd Grade in Primary Rougher Feed (g/t)',
                'Rh Grade in Primary Rougher Feed (g/t)',
                'Ir Grade in Primary Rougher Feed (g/t)',
                'Ru Grade in Primary Rougher Feed (g/t)',
                'Au Grade in Primary Rougher Feed (g/t)',
                'Chromia in Primary Rougher Feed (%)', 'Ni in Primary Rougher Feed (%)',
                'Cu in Primary Rougher Feed (%)', 'Co in Primary Rougher Feed (%)',
                'S in Primary Rougher Feed (%)',
                'PGM+Au Grade in Primary Rougher Feed (g/t)', '4E Content in Concentrate (ounce/day)'
]

common_cols = TRP_data_selected_feat.columns.intersection(features_and_target)

TRP_data_selected_feat = TRP_data_selected_feat[common_cols]

TRP_data_selected_feat.shape
```

Removal of Negative values

Entries with negative values in any of the features will be eliminated from the dataset.

In [18]:
```
# Copy dataframe
TRP_data_no_outliers = TRP_data_selected_feat.copy()

# Remove data collected during shutdown
TRP_data_no_outliers = TRP_data_no_outliers[TRP_data_no_outliers['Total Crushed Throughput
(tons/day)'] != 0]
TRP_data_no_outliers = TRP_data_no_outliers[TRP_data_no_outliers['Total Milled Throughput
(tons/day)'] != 0]

TRP_data_no_outliers.shape
```

In [19]:
```
# Remove rows with missing PSD data
psd_cols = [
    'Primary Rougher Feed +150 µm Size Class',
    'Primary Rougher Feed +106 µm Size Class',
    'Primary Rougher Feed +75 µm Size Class',
    'Primary Rougher Feed -75 µm Size Class',
    'Secondary Rougher Tailings +150 µm Size Class',
    'Secondary Rougher Tailings +106 µm Size Class',
    'Secondary Rougher Tailings +75 µm Size Class',
    'Secondary Rougher Tailings -75 µm Size Class',
    'Secondary Mill Discharge +150 µm Size Class',
    'Secondary Mill Discharge +106 µm Size Class',
    'Secondary Mill Discharge +75 µm Size Class',
    'Secondary Mill Discharge -75 µm Size Class'
]

# Iterate through each PSD column and drop rows where the value is zero
for psd_col in psd_cols:
    TRP_data_no_outliers = TRP_data_no_outliers[TRP_data_no_outliers[psd_col] != 0]

# Reset index after dropping rows
TRP_data_no_outliers.reset_index(drop=True, inplace=True)

# Print the shape of the cleaned DataFrame
print("Shape of TRP_data_no_outliers after cleaning:", TRP_data_no_outliers.shape)
```

In [20]:
```
# Copy dataframe
TRP_data_no_noise = TRP_data_no_outliers.copy()
```

In [21]:

```
# Define target columns
target_columns = ['4E Content in Concentrate (ounce/day)', 'Chromium Content in Concentrate (%)',
                  'PGM+Au Grade in Primary Rougher Feed (g/t)',
                  'Pt Grade in Primary Rougher Feed (g/t)',
                  'Pd Grade in Primary Rougher Feed (g/t)',
                  'Rh Grade in Primary Rougher Feed (g/t)',
                  'Ir Grade in Primary Rougher Feed (g/t)',
                  'Ru Grade in Primary Rougher Feed (g/t)',
                  'Au Grade in Primary Rougher Feed (g/t)',
                  'Chromia in Primary Rougher Feed (%)', 'Ni in Primary Rougher Feed (%)',
                  'Cu in Primary Rougher Feed (%)', 'Co in Primary Rougher Feed (%)',
                  'S in Primary Rougher Feed (%)']

r2_threshold = 0.2
```

In [22]:

```
# Function to remove noisy data based on R² score
def remove_noisy_data(data, target_columns, threshold):
    print("Starting data cleaning process...")
    cleaned_data = data.copy()

    while True:
        if cleaned_data.empty:
            print("DataFrame is empty after removing outliers.")
            break

        outlier_indices = set()
        for target in target_columns:
            X = cleaned_data.drop(target_columns + ['Date'], axis=1)
            y = cleaned_data[target]

            # Split the data
            X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

            # Fit RF regression model
            model = RandomForestRegressor()
            model.fit(X_train, y_train)

            # Predict and calculate R²
            y_pred = model.predict(X_test)
            r2 = r2_score(y_test, y_pred)

            print(f"R² for {target}: {r2:.4f}")

            # If R² is below the threshold, mark the points to remove
            if r2 < threshold:
                residuals = y_train - model.predict(X_train)
                outliers = residuals[residuals.abs() > residuals.abs().quantile(0.95)].index
                outlier_indices.update(outliers)

        print(f"Identified outlier indices: {outlier_indices}")

        # If no outliers are found, break the loop
        if not outlier_indices:
            print("No outliers found, exiting loop.")
            break

        # Ensure outlier_indices are valid
        valid_outlier_indices = [i for i in outlier_indices if i in cleaned_data.index]
        print(f"Valid outlier indices to drop: {valid_outlier_indices}")

        # Filter out the outliers using boolean indexing
        cleaned_data = cleaned_data[~cleaned_data.index.isin(valid_outlier_indices)]
        print(f"Cleaned Data Shape: {cleaned_data.shape}")

    return cleaned_data
```

In [23]:
```
# Check the shape of the DataFrame before calling the function
print(f"Initial Data Shape: {TRP_data_no_noise.shape}")

# Remove noisy data
TRP_data_cleaned = remove_noisy_data(TRP_data_no_noise, target_columns, r2_threshold)

# Print the final shape of the cleaned data
print(f'Final Cleaned Data Shape: {TRP_data_cleaned.shape}')
```

In [24]:
```
# Check the shape of the DataFrame before calling the function
print(f"Initial Data Shape: {TRP_data_no_noise.shape}")

# Remove noisy data
TRP_data_cleaned = remove_noisy_data(TRP_data_no_noise, target_columns, r2_threshold)

# Print the final shape of the cleaned data
print(f'Final Cleaned Data Shape: {TRP_data_cleaned.shape}')
```

In [24]:
```
# Copying data post preprocessing
TRP_data_DA = TRP_data_cleaned.copy()
```

APPENDIX B: Python Script for Ore Mineralogy Prediction

This appendix contains the Python code used to develop the ML algorithms used to predict the ore mineralogy parameters, including PGM+Au grade, Pt grade, Pd grade, Rh grade, Ir grade, Ru grade, Au grade, Cr2O3 %, Ni %, Cu %, Co %, and S %.

B1. Feature Selection

In [25]:

```
# Copying data post analysis
TRP_data_min = TRP_data_DA.copy()

# Defining Inputs and Outputs

targets_min = ['PGM+Au Grade in Primary Rougher Feed (g/t)',
               'Pt Grade in Primary Rougher Feed (g/t)',
               'Pd Grade in Primary Rougher Feed (g/t)',
               'Rh Grade in Primary Rougher Feed (g/t)',
               'Ir Grade in Primary Rougher Feed (g/t)',
               'Ru Grade in Primary Rougher Feed (g/t)',
               'Au Grade in Primary Rougher Feed (g/t)',
               'Chromia in Primary Rougher Feed (%)', 'Ni in Primary Rougher Feed (%)',
               'Cu in Primary Rougher Feed (%)', 'Co in Primary Rougher Feed (%)',
               'S in Primary Rougher Feed (%)']

inputs_min = ['Primary Rougher Feed +150 Âµm Size Class',
              'Primary Rougher Feed +106 Âµm Size Class',
              'Primary Rougher Feed +75 Âµm Size Class',
              'Primary Rougher Feed -75 Âµm Size Class',
              'Secondary Rougher Tailings +150 Âµm Size Class',
              'Secondary Rougher Tailings +106 Âµm Size Class',
              'Secondary Rougher Tailings +75 Âµm Size Class',
              'Secondary Rougher Tailings -75 Âµm Size Class',
              'Secondary Mill Discharge +150 Âµm Size Class',
              'Secondary Mill Discharge +106 Âµm Size Class',
              'Secondary Mill Discharge +75 Âµm Size Class',
              'Secondary Mill Discharge -75 Âµm Size Class',
              'Merensky (%)', 'Total Crushed Throughput (tons/day)',
              'Total Milled Throughput (tons/day)', '4E Product Recovery (%) calc',
              '4E Content in Concentrate (ounce/day)', "Chromium Content in Concentrate (%)"]
```

```
# Selection of input and target features from dataframe
X_min = TRP_data_min[inputs_min]
y_min = TRP_data_min[targets_min]

# Remove rows where any of the target variables are zero
non_zero_targets = y_min.loc[~(y_min == 0).any(axis=1)]
X_min = X_min.loc[non_zero_targets.index]
y_min = non_zero_targets
```

In [26]:

```
# Initialize and fit the Random Forest model
rf_model = RandomForestRegressor()
rf_model.fit(X_min, y_min)

# Extract feature importances
feature_importances = rf_model.feature_importances_

# Define the importance threshold
importance_threshold = 0.03

# Select features based on importance
selected_features = [feature for feature, importance in zip(X_min.columns, feature_importances) if
importance > 0.03]

# Print the selected features
print(len(X_min.columns), "Original Features")
print("Selected features:", selected_features)
print(len(selected_features), "Selected Features")

# Train-Test Split
X_train_min, X_test_min, y_train_min, y_test_min = train_test_split(X_min[selected_features], y_min,
test_size=0.3, random_state=42)

# Create a DataFrame for plotting
feature_importance_df = pd.DataFrame({
    'Feature': X_min.columns,
    'Importance': feature_importances
})

# Sort the features by importance
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

# Plot the feature importances
plt.figure(figsize=(10, 6))
plt.barh(feature_importance_df['Feature'], feature_importance_df['Importance'])
plt.xlabel('Importance')
plt.ylabel('Feature')
```

```
# Add a horizontal line to indicate the importance threshold
plt.axvline(x=importance_threshold, color='red', linestyle='--', label='Threshold')
plt.legend()
plt.title('Feature Importance Plot')
plt.gca().invert_yaxis()
plt.savefig(path + 'Ore Min Feature Selection.png', bbox_inches='tight')
plt.show()
```

B2. Model Development

For this section, 4 machine learning models will built: Linear, Random Forest, Extra Tree and k-nearest neighbour.

B2.1. Naïve Benchmark

In [27]:
```
# Conversion
X_train_min_naive = X_train_min.copy()
y_train_min_naive = y_train_min.copy()
X_test_min_naive = X_test_min.copy()
y_test_min_naive = y_test_min.copy()
```

In [28]:
```
# Dictionaries for train-test splits
train_data_dict_min = {}
test_data_dict_min = {}
```

In [29]:
```
for i, target_column in enumerate(y_min.columns):
    # Store train and test data in respective dictionaries
    train_data_dict_min[target_column] = {
        'X_train_min': X_train_min_naive,
        'y_train_min': y_train_min_naive.iloc[:, i]
    }
    test_data_dict_min[target_column] = {
        'X_test_min': X_test_min_naive,
        'y_test_min': y_test_min_naive.iloc[:, i]
    }

naive_rmse_dict_min = {}
```

In [30]:
```
for target_column in y_min.columns:
    y_train_min_naive = train_data_dict_min[target_column]['y_train_min']
    y_test_min_naive = test_data_dict_min[target_column]['y_test_min']

    # Calculate naive benchmark RMSE
    naive_prediction_min = np.full(y_test_min_naive.shape, np.mean(y_train_min_naive))
    naive_rmse_min = np.sqrt(mean_squared_error(y_test_min_naive, naive_prediction_min))

    # Calculate mean of y_train_min
    mean_y_train_min = np.mean(y_train_min_naive)

    # Calculate percentage of naive benchmark RMSE over mean
    naive_percentage_min = (naive_rmse_min / mean_y_train_min) * 100
    naive_rmse_dict_min[target_column] = naive_percentage_min

    print(f"Naive Benchmark RMSE for {target_column} as a percentage of mean:
{naive_percentage_min:.2f}%")
```

B2.2. Linear Regression

In [31]:
```
# Define the parameter grid for grid search
param_grid_lr = {
    'fit_intercept': [True, False]
}

# Create the Linear Regression model
lr_model = LinearRegression()
```

In [32]:
```
# Create the GridSearch object
grid_search_lr = GridSearchCV(estimator=lr_model,
                              param_grid=param_grid_lr,
                              cv=KFold(n_splits=10, shuffle=True),
                              scoring='neg_mean_squared_error',
                              verbose=2)

# Fit the GridSearchCV object to training data
grid_search_lr.fit(X_train_min, y_train_min)

# Print the best parameters found
print("Best parameters found:", grid_search_lr.best_params_)

# Get the best model
best_lr_model = grid_search_lr.best_estimator_
```

In [33]:
```
# Make predictions using cross-validation
y_pred_lr_test_min = cross_val_predict(best_lr_model, X_test_min, y_test_min, cv=KFold(n_splits=10,
shuffle=True))
y_pred_lr_train_min = cross_val_predict(best_lr_model, X_train_min, y_train_min,
cv=KFold(n_splits=10, shuffle=True))
```

In [34]:
```
# Model Evaluation

# Initialize lists to store RMSE and R^2 values for each target variable
rmse_scores_lr_test = []
rmse_scores_lr_train = []
r2_scores_lr_test = []
r2_scores_lr_train = []
```

In [35]:
```
# Loop over each target variable
for i, target_col in enumerate(y_min.columns):
    # Calculate RMSE for the current target variable
    rmse_test_min = (np.sqrt(mean_squared_error(y_test_min[target_col], y_pred_lr_test_min[:, i])) /
TRP_data_min[target_col].mean()) * 100
    rmse_train_min = (np.sqrt(mean_squared_error(y_train_min[target_col], y_pred_lr_train_min[:, i])) /
TRP_data_min[target_col].mean()) * 100
    rmse_scores_lr_test.append(rmse_test_min)
    rmse_scores_lr_train.append(rmse_train_min)
    print(f"Root Mean Squared Error for {target_col} train: {rmse_train_min:.2f} %")
    print(f"Root Mean Squared Error for {target_col} test: {rmse_test_min:.2f} %")

    # Calculate Adjusted R²
    n_train = len(y_train_min)
    n_test = len(y_test_min)
    p = X_train_min.shape[1]

    r2_train_min = r2_score(y_train_min[target_col], y_pred_lr_train_min[:, i])
    r2_test_min = r2_score(y_test_min[target_col], y_pred_lr_test_min[:, i])

    r2_scores_lr_train.append(r2_train_min)
    r2_scores_lr_test.append(r2_test_min)

    print(f"R² Score for {target_col} train: {r2_train_min:.2f}")
    print(f"R² Score for {target_col} test: {r2_test_min:.2f}")

    print(" ")
```

In [36]:
```
# Average RMSE and R^2 over all target variables
avg_rmse_lr_test = np.mean(rmse_scores_lr_test)
avg_rmse_lr_train = np.mean(rmse_scores_lr_train)
avg_r2_lr_test = np.mean(r2_scores_lr_test)
avg_r2_lr_train = np.mean(r2_scores_lr_train)

print("Average RMSE test:", avg_rmse_lr_test, ' %')
print("Average RMSE train:", avg_rmse_lr_train, ' %')
print("Average R^2 test", avg_r2_lr_test)
print("Average R^2 train:", avg_r2_lr_train)
```

In [37]:

```
# Compute RMSE and R² scores
rmse_scores_lr_test = []
rmse_scores_lr_train = []
r2_scores_lr_test = []
r2_scores_lr_train = []

for i, target_col in enumerate(y_min.columns):
    # Calculate RMSE for the current target variable
    rmse_test = np.sqrt(mean_squared_error(y_test_min[target_col], y_pred_lr_test_min[:, i])) /
np.mean(TRP_data_min[target_col]) * 100
    rmse_train = np.sqrt(mean_squared_error(y_train_min[target_col], y_pred_lr_train_min[:, i])) /
np.mean(TRP_data_min[target_col]) * 100
    rmse_scores_lr_test.append(rmse_test)
    rmse_scores_lr_train.append(rmse_train)

    # Calculate R² for the current target variable
    r2_test = r2_score(y_test_min[target_col], y_pred_lr_test_min[:, i])
    r2_train = r2_score(y_train_min[target_col], y_pred_lr_train_min[:, i])
    r2_scores_lr_test.append(r2_test)
    r2_scores_lr_train.append(r2_train)

# Convert lists to arrays
rmse_test_scores = np.array(rmse_scores_lr_test)
rmse_train_scores = np.array(rmse_scores_lr_train)
r2_test_scores = np.array(r2_scores_lr_test)
r2_train_scores = np.array(r2_scores_lr_train)

# Define target variable names
target_variables = y_min.columns  # This should have the same length as your score arrays

# Plot
fig, axs = plt.subplots(1, 2, figsize=(18, 6))

# Plot RMSE scores
bar_width = 0.35
index = np.arange(len(target_variables))

axs[0].bar(index - bar_width/2, rmse_test_scores, bar_width, label='Test RMSE', color='red', alpha=0.5)
axs[0].bar(index + bar_width/2, rmse_train_scores, bar_width, label='Train RMSE', color='blue',
alpha=0.5)

axs[0].set_title('RMSE Model Evaluation Scores')
axs[0].set_xlabel('Target Variables')
axs[0].set_ylabel('Error (%)')
axs[0].set_xticks(index)
axs[0].set_xticklabels(target_variables, rotation=45, ha='right')
axs[0].legend()
```

```
# Plot R² scores
axs[1].bar(index - bar_width/2, r2_test_scores, bar_width, label='Test R²', color='red', alpha=0.5)
axs[1].bar(index + bar_width/2, r2_train_scores, bar_width, label='Train R²', color='blue', alpha=0.5)

axs[1].set_title('R² Model Evaluation Scores')
axs[1].set_xlabel('Target Variables')
axs[1].set_ylabel('R² Scores')
axs[1].set_xticks(index)
axs[1].set_xticklabels(target_variables, rotation=45, ha='right')
axs[1].legend()

plt.tight_layout()
plt.savefig(path + 'Just Save Mineralogy Linear Regression Metric Evaluation.png', bbox_inches='tight')
plt.show()
```

In [38]:

```
# Define the indices of the target variables you want to plot
target_indices = [1, 2, 4, 6, 7]

# Ensure the indices are valid
target_indices = [idx for idx in target_indices if idx < len(y_min.columns)]

# Concatenate the training and testing data for plotting
X_combined = pd.concat([X_train_min, X_test_min])
y_combined = pd.concat([y_train_min, y_test_min])
y_pred_combined = np.concatenate([y_pred_lr_train_min, y_pred_lr_test_min], axis=0)

# Plot
fig, axs = plt.subplots(len(target_indices), 2, figsize=(14, 30))

for i, idx in enumerate(target_indices):
    # Ensure y_pred_combined has the correct column for the current index
    y_pred_combined_col = y_pred_combined[:, idx]

    # Scatter plot with combined regression line
    sns.scatterplot(ax=axs[i, 0], x=X_train_min.index, y=y_train_min.iloc[:, idx], label='Train Data',
color='skyblue')
    sns.scatterplot(ax=axs[i, 0], x=X_test_min.index, y=y_test_min.iloc[:, idx], label='Test Data',
color='lightcoral')
    sns.lineplot(ax=axs[i, 0], x=X_combined.index, y=y_pred_combined_col, label='Prediction',
color='black')
    sns.lineplot(ax=axs[i, 0], x=X_combined.index, y=np.mean(y_pred_combined_col), label='Mean
Prediction', color='green')
    axs[i, 0].set_xlabel('Index')
    axs[i, 0].set_ylabel('\n'.join(textwrap.wrap(y_min.columns[idx], width=30)))
    axs[i, 0].legend()
```

```
    # Scatter plot with x=y line
    sns.scatterplot(ax=axs[i, 1], x=y_combined.iloc[:, idx], y=y_pred_combined_col, label='Combined
Predicted', color='black')
    sns.regplot(ax=axs[i, 1], x=y_combined.iloc[:, idx], y=y_pred_combined_col, scatter=False,
color='blue', label='Regression Line')
    axs[i, 1].plot(y_combined.iloc[:, idx], y_combined.iloc[:, idx], color='green', linestyle='dashdot',
label='x=y')
    axs[i, 1].set_xlabel(f'Actual {y_min.columns[idx]}')
    axs[i, 1].set_ylabel(f'Predicted {y_min.columns[idx]}')
    axs[i, 1].legend()
    axs[i, 1].set_ylabel('\n'.join(textwrap.wrap(f'Predicted {y_min.columns[idx]}', width=30)))

plt.tight_layout()
plt.subplots_adjust(hspace=0.3)
plt.savefig(path + 'Mineralogy Linear Regression.png', bbox_inches='tight')
plt.show()
```

B2.3. Random Forest Regression

In [39]:

```
# Define the parameter grid for grid search
param_grid_rf_min = {
    'n_estimators': [10, 50, 100, 200, 300, 400],
    'max_depth': [1, 5, 9, 12, 14],
    'min_samples_leaf': [1, 2, 3, 4, 5],
}

# Create the Random Forest model
rf_model_min = RandomForestRegressor()

# Create the GridSearchCV object
grid_search_rf_min = GridSearchCV(estimator=rf_model_min,
                          param_grid=param_grid_rf_min,
                          cv=KFold(n_splits=10, shuffle=True),
                          n_jobs=-1,
                          verbose=2)
```

In [40]:
```
# Fit the GridSearchCV object to training data
grid_search_rf_min.fit(X_train_min, y_train_min)

# Print the best parameters found
print("Best parameters found:", grid_search_rf_min.best_params_)

# Get the best model
best_rf_model_min = grid_search_rf_min.best_estimator_
```

In [41]:
```
# Make predictions using cross-validation
y_pred_rf_test_min = cross_val_predict(best_rf_model_min, X_test_min, y_test_min,
cv=KFold(n_splits=10, shuffle=True))
y_pred_rf_train_min = cross_val_predict(best_rf_model_min, X_train_min, y_train_min,
cv=KFold(n_splits=10, shuffle=True))
```

In [42]:
```
# Model Evaluation

# Initialize lists to store RMSE and R^2 values for each target variable
rmse_scores_rf_test = []
rmse_scores_rf_train = []
r2_scores_rf_test = []
r2_scores_rf_train = []
```

In [43]:

```
# Loop over each target variable
for i, target_col in enumerate(y_min.columns):
    # Calculate RMSE for the current target variable
    rmse_test = (np.sqrt(mean_squared_error(y_test_min[target_col], y_pred_rf_test_min[:, i])) /
TRP_data_min[target_col].mean()) * 100
    rmse_train = (np.sqrt(mean_squared_error(y_train_min[target_col], y_pred_rf_train_min[:, i])) /
TRP_data_min[target_col].mean()) * 100
    rmse_scores_rf_test.append(rmse_test)
    rmse_scores_rf_train.append(rmse_train)
    print(f"Root Mean Squared Error for {target_col} train: {rmse_train:.2f} %")
    print(f"Root Mean Squared Error for {target_col} test: {rmse_test:.2f} %")

    r2_train_rf_min = r2_score(y_train_min[target_col], y_pred_rf_train_min[:, i])
    r2_test_rf_min = r2_score(y_test_min[target_col], y_pred_rf_test_min[:, i])

    r2_scores_rf_train.append(r2_train)
    r2_scores_rf_test.append(r2_test)

    print(f"R² Score for {target_col} train: {r2_train_rf_min:.2f}")
    print(f"R² Score for {target_col} test: {r2_test_rf_min:.2f}")

    print(" ")
```

In [44]:

```
# Average RMSE and R^2 over all target variables
avg_rmse_rf_test = np.mean(rmse_scores_rf_test)
avg_rmse_rf_train = np.mean(rmse_scores_rf_train)
avg_r2_rf_test = np.mean(r2_scores_rf_test)
avg_r2_rf_train = np.mean(r2_scores_rf_train)

print("Average RMSE test:", avg_rmse_rf_test, '%')
print("Average RMSE train:", avg_rmse_rf_train, '%')
print("Average R^2 test", avg_r2_rf_test)
print("Average R^2 train:", avg_r2_rf_train)
```

In [45]:

```
# Compute RMSE and R² scores
rmse_scores_rf_test = []
rmse_scores_rf_train = []
r2_scores_rf_test = []
r2_scores_rf_train = []

for i, target_col in enumerate(y_min.columns):
    # Calculate RMSE for the current target variable
    rmse_test = np.sqrt(mean_squared_error(y_test_min[target_col], y_pred_rf_test_min[:, i])) /
np.mean(TRP_data_min[target_col]) * 100
    rmse_train = np.sqrt(mean_squared_error(y_train_min[target_col], y_pred_rf_train_min[:, i])) /
np.mean(TRP_data_min[target_col]) * 100
    rmse_scores_rf_test.append(rmse_test)
    rmse_scores_rf_train.append(rmse_train)

    # Calculate R² for the current target variable
    r2_test = r2_score(y_test_min[target_col], y_pred_rf_test_min[:, i])
    r2_train = r2_score(y_train_min[target_col], y_pred_rf_train_min[:, i])
    r2_scores_rf_test.append(r2_test)
    r2_scores_rf_train.append(r2_train)

# Convert lists to arrays
rmse_test_scores = np.array(rmse_scores_rf_test)
rmse_train_scores = np.array(rmse_scores_rf_train)
r2_test_scores = np.array(r2_scores_rf_test)
r2_train_scores = np.array(r2_scores_rf_train)

# Define target variable names
target_variables = y_min.columns  # This should have the same length as your score arrays

# Plot
fig, axs = plt.subplots(1, 2, figsize=(18, 6))

# Plot RMSE scores
bar_width = 0.35
index = np.arange(len(target_variables))

axs[0].bar(index - bar_width/2, rmse_test_scores, bar_width, label='Test RMSE', color='red', alpha=0.5)
axs[0].bar(index + bar_width/2, rmse_train_scores, bar_width, label='Train RMSE', color='blue',
alpha=0.5)

axs[0].set_title('RMSE Model Evaluation Scores')
axs[0].set_xlabel('Target Variables')
axs[0].set_ylabel('Error (%)')
axs[0].set_xticks(index)
axs[0].set_xticklabels(target_variables, rotation=45, ha='right')
axs[0].legend()
```

```
# Plot R² scores
axs[1].bar(index - bar_width/2, r2_test_scores, bar_width, label='Test R²', color='red', alpha=0.5)
axs[1].bar(index + bar_width/2, r2_train_scores, bar_width, label='Train R²', color='blue', alpha=0.5)

axs[1].set_title('R² Model Evaluation Scores')
axs[1].set_xlabel('Target Variables')
axs[1].set_ylabel('R² Scores')
axs[1].set_xticks(index)
axs[1].set_xticklabels(target_variables, rotation=45, ha='right')
axs[1].legend()

plt.tight_layout()
plt.savefig(path + 'Just save already Mineralogy Random Forest Regression Metric Evaluation.png',
bbox_inches='tight')
plt.show()
```

In [46]:

```
# Define the indices of the target variables you want to plot
target_indices = [1, 2, 4, 6, 7]

# Ensure the indices are valid
target_indices = [idx for idx in target_indices if idx < len(y_min.columns)]

# Concatenate the training and testing data for plotting
X_combined = pd.concat([X_train_min, X_test_min])
y_combined = pd.concat([y_train_min, y_test_min])
y_pred_combined = np.concatenate([y_pred_rf_train_min, y_pred_rf_test_min], axis=0)

# Plot
fig, axs = plt.subplots(len(target_indices), 2, figsize=(14, 30))

for i, idx in enumerate(target_indices):
    # Ensure y_pred_combined has the correct column for the current index
    y_pred_combined_col = y_pred_combined[:, idx]

    # Scatter plot with combined regression line
    sns.scatterplot(ax=axs[i, 0], x=X_train_min.index, y=y_train_min.iloc[:, idx], label='Train Data',
color='skyblue')
    sns.scatterplot(ax=axs[i, 0], x=X_test_min.index, y=y_test_min.iloc[:, idx], label='Test Data',
color='lightcoral')
    sns.lineplot(ax=axs[i, 0], x=X_combined.index, y=y_pred_combined_col, label='Prediction',
color='black')
    sns.lineplot(ax=axs[i, 0], x=X_combined.index, y=np.mean(y_pred_combined_col), label='Mean
Prediction', color='green')
    axs[i, 0].set_xlabel('Index')
    axs[i, 0].set_ylabel('\n'.join(textwrap.wrap(y_min.columns[idx], width=30)))
    axs[i, 0].legend()
```

```
    # Scatter plot with x=y line
    sns.scatterplot(ax=axs[i, 1], x=y_combined.iloc[:, idx], y=y_pred_combined_col, label='Combined
Predicted', color='black')
    sns.regplot(ax=axs[i, 1], x=y_combined.iloc[:, idx], y=y_pred_combined_col, scatter=False,
color='blue', label='Regression Line')
    axs[i, 1].plot(y_combined.iloc[:, idx], y_combined.iloc[:, idx], color='green', linestyle='dashdot',
label='x=y')
    axs[i, 1].set_xlabel(f'Actual {y_min.columns[idx]}')
    axs[i, 1].set_ylabel(f'Predicted {y_min.columns[idx]}')
    axs[i, 1].legend()
    axs[i, 1].set_ylabel('\n'.join(textwrap.wrap(f'Predicted {y_min.columns[idx]}', width=30)))

plt.tight_layout()
plt.subplots_adjust(hspace=0.3)
plt.savefig(path + 'Mineralogy Random Forest Regression.png', bbox_inches='tight')
plt.show()
```

B2.4. Extra Tree Regression

In [47]:

```
# Define the parameter grid for grid search
param_grid_et_min = {
    'n_estimators': [100, 200, 300, 400],
    'max_depth': [10, 15, 20, 25, 30],
    'min_samples_leaf': [1, 2, 5, 10]
}

# Create the ExtraTreesRegressor model
et_model_min = ExtraTreesRegressor(random_state=42)

# Create the GridSearchCV object
grid_search_et_min = GridSearchCV(
    estimator=et_model_min,
    param_grid=param_grid_et_min,
    cv=KFold(n_splits=10, shuffle=True),
    n_jobs=-1,
    verbose=1
)
```

In [48]:
```
# Fit the GridSearchCV object to training data
grid_search_et_min.fit(X_train_min, y_train_min)

# Print the best parameters found
print("Best parameters found:", grid_search_et_min.best_params_)

# Get the best model
best_et_model_min = grid_search_et_min.best_estimator_
```

In [49]:
```
# Make predictions using cross-validation
y_pred_et_test_min = cross_val_predict(best_et_model_min, X_test_min, y_test_min,
cv=KFold(n_splits=10, shuffle=True))
y_pred_et_train_min = cross_val_predict(best_et_model_min, X_train_min, y_train_min,
cv=KFold(n_splits=10, shuffle=True))
```

In [50]:
```
# Model Evaluation

# Initialize lists to store RMSE and R^2 values for each target variable
rmse_scores_et_test = []
rmse_scores_et_train = []
r2_scores_et_test = []
r2_scores_et_train = []
```

In [51]:

```
# Loop over each target variable
for i, target_col in enumerate(y_min.columns):
    # Calculate RMSE for the current target variable
    rmse_test = (np.sqrt(mean_squared_error(y_test_min[target_col], y_pred_et_test_min[:, i])) /
TRP_data_min[target_col].mean()) * 100
    rmse_train = (np.sqrt(mean_squared_error(y_train_min[target_col], y_pred_et_train_min[:, i])) /
TRP_data_min[target_col].mean()) * 100
    rmse_scores_et_test.append(rmse_test)
    rmse_scores_et_train.append(rmse_train)
    print(f"Root Mean Squared Error for {target_col} train: {rmse_train:.2f} %")
    print(f"Root Mean Squared Error for {target_col} test: {rmse_test:.2f} %")

    # Calculate Adjusted R²
    n_train = len(y_train_min)
    n_test = len(y_test_min)
    p = X_train_min.shape[1]

    r2_train_et_min = r2_score(y_train_min[target_col], y_pred_et_train_min[:, i])
    r2_test_et_min = r2_score(y_test_min[target_col], y_pred_et_test_min[:, i])

    r2_scores_et_train.append(r2_train_et_min)
    r2_scores_et_test.append(r2_test_et_min)

    print(f"R² Score for {target_col} train: {r2_train_et_min:.2f}")
    print(f"R² Score for {target_col} test: {r2_test_et_min:.2f}")

    print(" ")
```

In [52]:

```
# Average RMSE and R^2 over all target variables
avg_rmse_et_test = np.mean(rmse_scores_et_test)
avg_rmse_et_train = np.mean(rmse_scores_et_train)
avg_r2_et_test = np.mean(r2_scores_et_test)
avg_r2_et_train = np.mean(r2_scores_et_train)

print("Average RMSE test:", avg_rmse_et_test, '%')
print("Average RMSE train:", avg_rmse_et_train, '%')
print("Average R^2 test", avg_r2_et_test)
print("Average R^2 train:", avg_r2_et_train)
```

In [53]:

```
# Define the RMSE and R2 scores
rmse_test_scores = np.array(rmse_scores_et_test)
rmse_train_scores = np.array(rmse_scores_et_train)
r2_test_scores = np.array(r2_scores_et_test)
r2_train_scores = np.array(r2_scores_et_train)

# Plot
fig, axs = plt.subplots(1, 2, figsize=(18, 6))

# Plot RMSE scores
bar_width = 0.35
index = np.arange(len(target_variables))

axs[0].bar(index - bar_width/2, rmse_test_scores, bar_width, label='Test RMSE', color='red', alpha=0.5)
axs[0].bar(index + bar_width/2, rmse_train_scores, bar_width, label='Train RMSE', color='blue',
alpha=0.5)

axs[0].set_title('RMSE Model Evaluation Scores')
axs[0].set_xlabel('Target Variables')
axs[0].set_ylabel('Error (%)')
axs[0].set_xticks(index)
axs[0].set_xticklabels(target_variables, rotation=45, ha='right')
axs[0].legend()

# Plot R2 scores
bar_width = 0.35

axs[1].bar(index - bar_width/2, r2_test_scores, bar_width, label='Test R^2', color='red', alpha=0.5)
axs[1].bar(index + bar_width/2, r2_train_scores, bar_width, label='Train R^2', color='blue', alpha=0.5)

axs[1].set_title('R^2 Model Evaluation Scores')
axs[1].set_xlabel('Target Variables')
axs[1].set_ylabel(' R^2 Scores')
axs[1].set_xticks(index)
axs[1].set_xticklabels(target_variables, rotation=45, ha='right')
axs[1].legend()

plt.tight_layout()
plt.savefig(path + 'Mineralogy Extra Trees Regression Metric Evaluation.png', bbox_inches='tight')
# plt.close()
plt.show()
```

In [54]:

```
# Define the indices of the target variables you want to plot
target_indices = [1, 2, 4, 6, 7]

# Ensure the indices are valid
target_indices = [idx for idx in target_indices if idx < len(y_min.columns)]

# Concatenate the training and testing data for plotting
X_combined = pd.concat([X_train_min, X_test_min])
y_combined = pd.concat([y_train_min, y_test_min])
y_pred_combined = np.concatenate([y_pred_et_train_min, y_pred_et_test_min], axis=0)

# Plot
fig, axs = plt.subplots(len(target_indices), 2, figsize=(14, 30))

for i, idx in enumerate(target_indices):
    # Ensure y_pred_combined has the correct column for the current index
    y_pred_combined_col = y_pred_combined[:, idx]

    # Scatter plot with combined regression line
    sns.scatterplot(ax=axs[i, 0], x=X_train_min.index, y=y_train_min.iloc[:, idx], label='Train Data',
color='skyblue')
    sns.scatterplot(ax=axs[i, 0], x=X_test_min.index, y=y_test_min.iloc[:, idx], label='Test Data',
color='lightcoral')
    sns.lineplot(ax=axs[i, 0], x=X_combined.index, y=y_pred_combined_col, label='Prediction',
color='black')
    sns.lineplot(ax=axs[i, 0], x=X_combined.index, y=np.mean(y_pred_combined_col), label='Mean
Prediction', color='green')
    axs[i, 0].set_xlabel('Index')
    axs[i, 0].set_ylabel('\n'.join(textwrap.wrap(y_min.columns[idx], width=30)))
    axs[i, 0].legend()

    # Scatter plot with x=y line
    sns.scatterplot(ax=axs[i, 1], x=y_combined.iloc[:, idx], y=y_pred_combined_col, label='Combined
Predicted', color='black')
    sns.regplot(ax=axs[i, 1], x=y_combined.iloc[:, idx], y=y_pred_combined_col, scatter=False,
color='blue', label='Regression Line')
    axs[i, 1].plot(y_combined.iloc[:, idx], y_combined.iloc[:, idx], color='green', linestyle='dashdot',
label='x=y')
    axs[i, 1].set_xlabel(f'Actual {y_min.columns[idx]}')
    axs[i, 1].set_ylabel(f'Predicted {y_min.columns[idx]}')
    axs[i, 1].legend()
    axs[i, 1].set_ylabel('\n'.join(textwrap.wrap(f'Predicted {y_min.columns[idx]}', width=30)))

plt.tight_layout()
plt.subplots_adjust(hspace=0.3)
plt.savefig(path + 'Mineralogy Extra Trees Regression.png', bbox_inches='tight')
plt.show()
```

B2.5. k-Nearest Neighbour Regression

In [55]:
```
# Define the parameter grid for grid search
param_grid_knn_min = {
    'n_neighbors': [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29]
}

# Create the kNN model
knn_model_min = KNeighborsRegressor()

# Create the GridSearchCV object
grid_search_knn_min = GridSearchCV(estimator=knn_model_min,
                          param_grid=param_grid_knn_min,
                          cv=KFold(n_splits=10, shuffle=True)
                         )
```

In [56]:
```
# Fit the GridSearchCV object to training data
grid_search_knn_min.fit(X_train_min, y_train_min)

# Print the best parameters found
print("Best parameters found:", grid_search_knn_min.best_params_)

# Get the best model
best_knn_model_min = grid_search_knn_min.best_estimator_
```

In [57]:
```
# Make predictions using cross-validation
y_pred_knn_test_min = cross_val_predict(best_knn_model_min, X_test_min, y_test_min,
cv=KFold(n_splits=10, shuffle=True))
y_pred_knn_train_min = cross_val_predict(best_knn_model_min, X_train_min, y_train_min,
cv=KFold(n_splits=10, shuffle=True))
```

In [58]:
```
# Model Evaluation

# Initialize lists to store RMSE and R^2 values for each target variable
rmse_scores_knn_test = []
rmse_scores_knn_train = []
r2_scores_knn_test = []
r2_scores_knn_train = []
```

In [59]:
```
# Loop over each target variable
for i, target_col in enumerate(y_min.columns):
    # Calculate RMSE for the current target variable
    rmse_test = (np.sqrt(mean_squared_error(y_test_min[target_col], y_pred_knn_test_min[:, i])) /
TRP_data_min[target_col].mean()) * 100
    rmse_train = (np.sqrt(mean_squared_error(y_train_min[target_col], y_pred_knn_train_min[:, i])) /
TRP_data_min[target_col].mean()) * 100
    rmse_scores_knn_test.append(rmse_test)
    rmse_scores_knn_train.append(rmse_train)
    print(f"Root Mean Squared Error for {target_col} train: {rmse_train:.2f} %")
    print(f"Root Mean Squared Error for {target_col} test: {rmse_test:.2f} %")

    # Calculate Adjusted R²
    n_train = len(y_train_min)
    n_test = len(y_test_min)
    p = X_train_min.shape[1]

    r2_train_knn_min = r2_score(y_train_min[target_col], y_pred_rf_train_min[:, i])
    r2_test_knn_min = r2_score(y_test_min[target_col], y_pred_rf_test_min[:, i])

    r2_scores_knn_train.append(r2_train_knn_min)
    r2_scores_knn_test.append(r2_test_knn_min)

    print(f"R² Score for {target_col} train: {r2_train_knn_min:.2f}")
    print(f"R² Score for {target_col} test: {r2_test_knn_min:.2f}")

    print(" ")
```

In [60]:

```
# Average RMSE and R^2 over all target variables
avg_rmse_knn_test = np.mean(rmse_scores_knn_test)
avg_rmse_knn_train = np.mean(rmse_scores_knn_train)
avg_r2_knn_test = np.mean(r2_scores_knn_test)
avg_r2_knn_train = np.mean(r2_scores_knn_train)

print("Average RMSE test:", avg_rmse_knn_test, '%')
print("Average RMSE train:", avg_rmse_knn_train, '%')
print("Average R^2 test", avg_r2_knn_test)
print("Average R^2 train:", avg_r2_knn_train)
```

In [61]:

```
# Define the target variables and concatenate the scores
target_variables = ['PGM+Au grade', 'Platinum grade', 'Palladium grade',
                    'Rhodium grade', 'Iridium grade', 'Ruthenium grade',
                    'Gold grade', 'Chromium %', 'Nickel %', 'Copper %',
                    "Cobalt %", "Sulphur %"]

# Define the RMSE and R2 scores
rmse_test_scores = np.array(rmse_scores_knn_test)
rmse_train_scores = np.array(rmse_scores_knn_train)
r2_test_scores = np.array(r2_scores_knn_test)
r2_train_scores = np.array(r2_scores_knn_train)

# Plot
fig, axs = plt.subplots(1, 2, figsize=(18, 6))

# Plot RMSE scores
bar_width = 0.35
index = np.arange(len(target_variables))


axs[0].bar(index - bar_width/2, rmse_test_scores, bar_width, label='Test RMSE', color='red', alpha=0.5)
axs[0].bar(index + bar_width/2, rmse_train_scores, bar_width, label='Train RMSE', color='blue',
alpha=0.5)

axs[0].set_title('RMSE Model Evaluation Scores')
axs[0].set_xlabel('Target Variables')
axs[0].set_ylabel('Error (%)')
axs[0].set_xticks(index)
axs[0].set_xticklabels(target_variables, rotation=45, ha='right')
axs[0].legend()

# Plot R2 scores
bar_width = 0.35

axs[1].bar(index - bar_width/2, r2_test_scores, bar_width, label='Test R^2', color='red', alpha=0.5)
axs[1].bar(index + bar_width/2, r2_train_scores, bar_width, label='Train R^2', color='blue', alpha=0.5)

axs[1].set_title('R^2 Model Evaluation Scores')
axs[1].set_xlabel('Target Variables')
axs[1].set_ylabel(' R^2 Scores')
axs[1].set_xticks(index)
axs[1].set_xticklabels(target_variables, rotation=45, ha='right')
axs[1].legend()

plt.tight_layout()
plt.savefig(path + 'Mineralogy kNN Regression Metric Evaluation.png', bbox_inches='tight')
# plt.close()
plt.show()
```

In [62]:

```
# Define the indices of the target variables you want to plot
target_indices = [1, 2, 4, 6, 7]

# Ensure the indices are valid
target_indices = [idx for idx in target_indices if idx < len(y_min.columns)]

# Concatenate the training and testing data for plotting
X_combined = pd.concat([X_train_min, X_test_min])
y_combined = pd.concat([y_train_min, y_test_min])
y_pred_combined = np.concatenate([y_pred_knn_train_min, y_pred_knn_test_min], axis=0)

# Plot
fig, axs = plt.subplots(len(target_indices), 2, figsize=(14, 30))

for i, idx in enumerate(target_indices):
    # Ensure y_pred_combined has the correct column for the current index
    y_pred_combined_col = y_pred_combined[:, idx]

    # Scatter plot with combined regression line
    sns.scatterplot(ax=axs[i, 0], x=X_train_min.index, y=y_train_min.iloc[:, idx], label='Train Data',
color='skyblue')
    sns.scatterplot(ax=axs[i, 0], x=X_test_min.index, y=y_test_min.iloc[:, idx], label='Test Data',
color='lightcoral')
    sns.lineplot(ax=axs[i, 0], x=X_combined.index, y=y_pred_combined_col, label='Prediction',
color='black')
    sns.lineplot(ax=axs[i, 0], x=X_combined.index, y=np.mean(y_pred_combined_col), label='Mean
Prediction', color='green')
    axs[i, 0].set_xlabel('Index')
    axs[i, 0].set_ylabel('\n'.join(textwrap.wrap(y_min.columns[idx], width=30)))
    axs[i, 0].legend()

    # Scatter plot with x=y line
    sns.scatterplot(ax=axs[i, 1], x=y_combined.iloc[:, idx], y=y_pred_combined_col, label='Combined
Predicted', color='black')
    sns.regplot(ax=axs[i, 1], x=y_combined.iloc[:, idx], y=y_pred_combined_col, scatter=False,
color='blue', label='Regression Line')
    axs[i, 1].plot(y_combined.iloc[:, idx], y_combined.iloc[:, idx], color='green', linestyle='dashdot',
label='x=y')
    axs[i, 1].set_xlabel(f'Actual {y_min.columns[idx]}')
    axs[i, 1].set_ylabel(f'Predicted {y_min.columns[idx]}')
    axs[i, 1].legend()
    axs[i, 1].set_ylabel('\n'.join(textwrap.wrap(f'Predicted {y_min.columns[idx]}', width=30)))

plt.tight_layout()
plt.subplots_adjust(hspace=0.3)
plt.savefig(path + 'Mineralogy kNN Regression.png', bbox_inches='tight')
plt.show()
```

APPENDIX C: Python Script for Plant Performance Prediction

This section includes the Python code used to construct the ML algorithms for predicting the MF2 concentrator plant's performance metrics, such as 4E and chromium content in the concentrate.

C1. Feature Selection

In [63]:

```
# Copying data post analysis
TRP_data_FE = TRP_data_DA.copy()

# Define input feature and target variable
targets = ["4E Content in Concentrate (ounce/day)", 'Chromium Content in Concentrate (%)']

inputs = ['Primary Rougher Feed +150 Âµm Size Class',
          'Primary Rougher Feed +106 Âµm Size Class',
          'Primary Rougher Feed +75 Âµm Size Class',
          'Primary Rougher Feed -75 Âµm Size Class',
          'Secondary Rougher Tailings +150 Âµm Size Class',
          'Secondary Rougher Tailings +106 Âµm Size Class',
          'Secondary Rougher Tailings +75 Âµm Size Class',
          'Secondary Rougher Tailings -75 Âµm Size Class',
          'Secondary Mill Discharge +150 Âµm Size Class',
          'Secondary Mill Discharge +106 Âµm Size Class',
          'Secondary Mill Discharge +75 Âµm Size Class',
          'Secondary Mill Discharge -75 Âµm Size Class',
          'Merensky (%)', 'Total Crushed Throughput (tons/day)',
          'Total Milled Throughput (tons/day)']

# Selection of input and target features from dataframe
X = TRP_data_FE[inputs]
y = TRP_data_FE[targets]
```

In [64]:

```
# Initialize grade and chrome features lists
X_4e = []
X_chrome = []

# Initialize dictionaries to hold train-test splits
train_data_dict = {}
test_data_dict = {}
```

In [65]:

```
# Loop through each target column in y DataFrame
for idx, target_column in enumerate(y.columns):
    # Fit Extra Tree model to select features
    rf_model = ExtraTreesRegressor()
    rf_model.fit(X, y[target_column])
    feature_importances = rf_model.feature_importances_

    # Select features based on importance threshold
    selected_features = [feature for feature, importance in zip(X.columns, feature_importances) if
importance > 0.03]

    # Append selected features to corresponding lists
    if target_column == "4E Content in Concentrate (ounce/day)":
        X_4e.extend(selected_features)
        save_features = X_4e.copy()
    elif target_column == 'Chromium Content in Concentrate (%)':
        X_chrome.extend(selected_features)
        save_features = X_chrome.copy()

    print(target_column)
    print(len(X.columns), "Original Features")
    print("Selected features for target", target_column, ":", selected_features)
    print(len(selected_features), "Selected Features\n")

    # Split the data into train and test sets using train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X[selected_features], y[target_column], test_size=0.3,
random_state=42)

    # Store train and test datasets in dictionaries
    train_data_dict[target_column] = {
        'X_train': X_train,
        'y_train': y_train
    }

    test_data_dict[target_column] = {
        'X_test': X_test,
        'y_test': y_test
    }

    # Store train and test datasets as X_train_grade, X_test_grade, X_train_chrome, X_test_chrome
    if target_column == "4E Content in Concentrate (ounce/day)":
        X_train_4e, X_test_4e, y_train_4e, y_test_4e = X_train, X_test, y_train, y_test
    elif target_column == 'Chromium Content in Concentrate (%)':
        X_train_chrome, X_test_chrome, y_train_chrome, y_test_chrome = X_train, X_test, y_train, y_test

    # Fit the Linear Regression model on the selected features
    lr_model = LinearRegression()
    lr_model.fit(X_train, y_train)
```

```
# Extract coefficients for each target variable
coefficients = lr_model.coef_

# Prepare data for tabulate
table_data = []
for feature, coefficient in zip(selected_features, coefficients):
    table_data.append([feature, coefficient])

# Print the feature importance in a table
print(tabulate(table_data, headers=["Feature", "Coefficient"], tablefmt="pretty"))

# Plot feature importance for each target variable
plt.figure(figsize=(10, 6))
bar_width = 0.35
index = range(len(selected_features))

plt.barh(index, coefficients, bar_width, color='blue', label=f'Coefficient for {target_column}')
plt.xlabel('Coefficient Value')
plt.ylabel('Feature')
plt.title(f'Feature Importance Values for {target_column}')
plt.yticks(index, selected_features)
plt.tight_layout()

# Save the plot to a file
save_dir = 'Downloads'
os.makedirs(save_dir, exist_ok=True)
output_path = os.path.join(save_dir, f'{target_column.replace("/", "_")}_coefficients.png')
plt.savefig(output_path, bbox_inches='tight')
plt.show()
```

In [66]:

```
# Set the directory to save the figure
save_dir = 'Downloads'
os.makedirs(save_dir, exist_ok=True)
output_path = os.path.join(save_dir, 'Feature_Importance_All_Targets.png')

# Determine the number of target columns
num_targets = len(y.columns)

# Define the importance threshold
importance_threshold = 0.03

# Create a figure with a subplot for each target variable
fig, axs = plt.subplots(num_targets, 1, figsize=(10, num_targets * 5))

# Loop through each target column in y DataFrame
for idx, target_column in enumerate(y.columns):
    # Fit Extra Trees Regressor to determine feature importances
    rf_model = ExtraTreesRegressor()
    rf_model.fit(X, y[target_column])
    feature_importances = rf_model.feature_importances_

    # Prepare a DataFrame to sort and plot feature importances
    feature_importance_df = pd.DataFrame({
        'Feature': X.columns,
        'Importance': feature_importances
    }).sort_values(by='Importance', ascending=False)

    # Plot the feature importances in the respective subplot
    axs[idx].barh(feature_importance_df['Feature'], feature_importance_df['Importance'])
    axs[idx].set_xlabel('Importance')
    axs[idx].set_ylabel('Feature')
    axs[idx].set_title(f'Feature Importance for {target_column}')
    axs[idx].invert_yaxis()

    # Add a horizontal line to indicate the importance threshold
    axs[idx].axvline(x=importance_threshold, color='red', linestyle='--', label='Threshold')
    axs[idx].legend()

# Adjust layout to prevent overlap and save the figure
plt.tight_layout()
plt.savefig(output_path, bbox_inches='tight')
plt.show()
```

C2. Model Development

For this section, 4 machine learning models will built: Linear, Random Forest, Extra Tree and k-nearest neighbour.

C2.1. Naïve Benchmark

In [67]:
```
# 4E Content in Concentrate

# Calculate naive benchmark RMSE
naive_prediction = np.full_like(y_test_4e, np.mean(y_train_4e))
naive_rmse = np.sqrt(mean_squared_error(y_test_4e, naive_prediction))

# Calculate mean of y_train
mean_y_train_4e = np.mean(y_train_4e)

# Calculate percentage of naive benchmark RMSE over mean
naive_percentage = (naive_rmse / mean_y_train_4e) * 100

print(f"Naive Benchmark RMSE for 4E grade as a percentage of mean: {naive_percentage:.2f}%")
```

In [68]:
```
# Chromium Content in Concentrate

# Calculate naive benchmark RMSE
naive_prediction = np.full_like(y_test_chrome, np.mean(y_train_chrome))
naive_rmse = np.sqrt(mean_squared_error(y_test_chrome, naive_prediction))

# Calculate mean of y_train
mean_y_train_chrome = np.mean(y_train_chrome)

# Calculate percentage of naive benchmark RMSE over mean
naive_percentage = (naive_rmse / mean_y_train_chrome) * 100

print(f"Naive Benchmark RMSE for chromium content as a percentage of mean:
{naive_percentage:.2f}%")
```

C2.2. Linear Regression

In [69]:
```
# Define the parameter grid for grid search
param_grid_lr = {
    'fit_intercept': [True, False]
}

# Create the Lasso model
lr_model = LinearRegression()

# Create the GridSearchCV object
grid_search_lr = GridSearchCV(estimator=lr_model,
                              param_grid=param_grid_lr,
                              cv=KFold(n_splits=10, shuffle=True),
                              scoring='neg_mean_squared_error',
                              verbose=2)
```

In [70]:
```
# Fit the GridSearchCV object to training data
grid_search_lr.fit(X_train_4e, y_train_4e)

# Print the best parameters found by GridSearchCV
print("Best parameters found for 4E Grade: ", grid_search_lr.best_params_)

# Predict using the best Ridge model
best_lr_model_4e = grid_search_lr.best_estimator_
```

In [71]:
```
# Make predictions using cross-validation
y_pred_4e_train_lr = cross_val_predict(best_lr_model_4e, X_train_4e, y_train_4e,
cv=KFold(n_splits=10, shuffle=True))
y_pred_4e_test_lr = cross_val_predict(best_lr_model_4e, X_test_4e, y_test_4e, cv=KFold(n_splits=10,
shuffle=True))
```

In [72]:
```
# Model Evaluation

# Root Mean Squared Error
rmse_train_4e_lr = (np.sqrt(mean_squared_error(y_train_4e, y_pred_4e_train_lr)) / TRP_data_FE['4E Content in Concentrate (ounce/day)'].mean()) * 100
rmse_test_4e_lr = (np.sqrt(mean_squared_error(y_test_4e, y_pred_4e_test_lr)) / TRP_data_FE['4E Content in Concentrate (ounce/day)'].mean()) * 100

print(f"Root Mean Squared Error for 4E Content in Concentrate (ounce/day) train: {rmse_train_4e_lr} %")
print(f"Root Mean Squared Error for 4E Content in Concentrate (ounce/day) test: {rmse_test_4e_lr} %")
```

In [73]:
```
# Model Evaluation

# R2
r2_train_4e_lr = r2_score(y_train_4e, y_pred_4e_train_lr)
r2_test_4e_lr = r2_score(y_test_4e, y_pred_4e_test_lr)

print(f"R-Squared for 4E Content in Concentrate (ounce/day) train: {r2_train_4e_lr}")
print(f"R-Squared for 4E Content in Concentrate (ounce/day) test: {r2_test_4e_lr}")
```

In [74]:
```
# Fit the GridSearchCV object to training data
grid_search_lr.fit(X_train_chrome, y_train_chrome)

# Print the best parameters found by GridSearchCV
print("Best parameters found for Chromium Content: ", grid_search_lr.best_params_)

# Predict using the best Ridge model
best_lr_model_chrome = grid_search_lr.best_estimator_
```

In [75]:
```
# Make predictions using cross-validation
y_pred_chrome_train_lr = cross_val_predict(best_lr_model_chrome, X_train_chrome, y_train_chrome, cv=KFold(n_splits=10, shuffle=True))
y_pred_chrome_test_lr = cross_val_predict(best_lr_model_chrome, X_test_chrome, y_test_chrome, cv=KFold(n_splits=10, shuffle=True))
```

In [76]:

```
# Model Evaluation

# Root Mean Squared Error
rmse_train_chrome_lr = (np.sqrt(mean_squared_error(y_train_chrome, y_pred_chrome_train_lr)) /
TRP_data_FE['Chromium Content in Concentrate (%)'].mean()) * 100
rmse_test_chrome_lr = (np.sqrt(mean_squared_error(y_test_chrome, y_pred_chrome_test_lr)) /
TRP_data_FE['Chromium Content in Concentrate (%)'].mean()) * 100

print(f"Root Mean Squared Error for Chromium Content train: {rmse_train_chrome_lr} %")
print(f"Root Mean Squared Error for Chromium Content test: {rmse_test_chrome_lr} %")

# R2
r2_train_chrome_lr = 1 - (1 - r2_score(y_train_chrome, y_pred_chrome_train_lr) * (len(y_train_chrome)
- 1) / (len(y_train_chrome) - X_train_chrome.shape[1] - 1))
r2_test_chrome_lr = 1 - (1 - r2_score(y_test_chrome, y_pred_chrome_test_lr) * (len(y_test_chrome) - 1) /
(len(y_test_chrome) - X_test_chrome.shape[1] - 1))

print(f"R-Squared for Chromium Content train: {r2_train_chrome_lr}")
print(f"R-Squared for Chromium Content test: {r2_test_chrome_lr}")
```

In [77]:

```
# Define the indices of the target variables you want to plot
target_indices = [0, 1]

# Concatenate the training and testing data for plotting
X_combined_4e = pd.concat([X_train_4e, X_test_4e])
X_combined_chrome = pd.concat([X_train_chrome, X_test_chrome])
y_combined_4e = pd.concat([y_train_4e, y_test_4e])
y_combined_chrome = pd.concat([y_train_chrome, y_test_chrome])
y_pred_combined_4e = np.concatenate([y_pred_4e_train_lr, y_pred_4e_test_lr])
y_pred_combined_chrome = np.concatenate([y_pred_chrome_train_lr, y_pred_chrome_test_lr])

# Plot
fig, axs = plt.subplots(len(target_indices), 2, figsize=(14, 10))

for i, idx in enumerate(target_indices):
    if idx == 0:
        # Combined predictions for 4E Grade
        y_pred_combined_col = y_pred_combined_4e

        # Scatter plot with combined regression line for 4E Grade
        sns.scatterplot(ax=axs[i, 0], x=X_train_4e.index, y=y_train_4e, label='Train Data', color='skyblue')
        sns.scatterplot(ax=axs[i, 0], x=X_test_4e.index, y=y_test_4e, label='Test Data', color='lightcoral')
        sns.lineplot(ax=axs[i, 0], x=X_combined_4e.index, y=y_pred_combined_col, label='Prediction',
color='black')
        sns.lineplot(ax=axs[i, 0], x=X_combined_4e.index, y=y_pred_combined_col.mean(), label='Mean',
color='green')
        axs[i, 0].set_xlabel('Index')
        axs[i, 0].set_ylabel(f'4E Content in Concentrate (ounce/day)')
        axs[i, 0].legend()

        # Scatter plot with x=y line for 4E Grade
        sns.scatterplot(ax=axs[i, 1], x=y_combined_4e, y=y_pred_combined_col, label='Combined
Predicted', color='black')
        sns.regplot(ax=axs[i, 1], x=y_combined_4e, y=y_pred_combined_col, scatter=False, color='blue',
label='Regression Line')
        axs[i, 1].plot(y_combined_4e, y_combined_4e, color='green', linestyle='dashdot', label='x=y')
        axs[i, 1].set_xlabel(f'Actual 4E Content in Concentrate (ounce/day)')
        axs[i, 1].set_ylabel(f'Predicted 4E Content in Concentrate (ounce/day)')
        axs[i, 1].legend()

    elif idx == 1:
        # Combined predictions for Chromium Content
        y_pred_combined_col = y_pred_combined_chrome
```

```
        # Scatter plot with combined regression line for Chromium Content
        sns.scatterplot(ax=axs[i, 0], x=X_train_chrome.index, y=y_train_chrome, label='Train Data',
color='skyblue')
        sns.scatterplot(ax=axs[i, 0], x=X_test_chrome.index, y=y_test_chrome, label='Test Data',
color='lightcoral')
        sns.lineplot(ax=axs[i, 0], x=X_combined_chrome.index, y=y_pred_combined_col, label='Prediction',
color='black')
        sns.lineplot(ax=axs[i, 0], x=X_combined_chrome.index, y=y_pred_combined_col.mean(),
label='Mean', color='green')
        axs[i, 0].set_xlabel('Index')
        axs[i, 0].set_ylabel(f'Chromium Content in Concentrate (%)')
        axs[i, 0].legend()

        # Scatter plot with x=y line for Chromium Content
        sns.scatterplot(ax=axs[i, 1], x=y_combined_chrome, y=y_pred_combined_col, label='Combined
Predicted', color='black')
        sns.regplot(ax=axs[i, 1], x=y_combined_chrome, y=y_pred_combined_col, scatter=False,
color='blue', label='Regression Line')
        axs[i, 1].plot(y_combined_chrome, y_combined_chrome, color='green', linestyle='dashdot',
label='x=y')
        axs[i, 1].set_xlabel(f'Actual Chromium Content in Concentrate (%)')
        axs[i, 1].set_ylabel(f'Predicted Chromium Content in Concentrate (%)')
        axs[i, 1].legend()

# Adjust layout and save the plot
plt.tight_layout()
plt.savefig(path + 'Plant Performance Linear Regression.png', bbox_inches='tight')
plt.show()
```

C2.3. Random Forest Regression

In [69]:

```
# Define the parameter grid for grid search

param_grid_rf = {
    'n_estimators': randint(1, 300),
    'max_depth': randint(1, 15),
    'min_samples_leaf': [25, 30, 35, 40, 50, 100],
    'min_samples_leaf': randint(1, 40),
    'max_features': ['sqrt', 'log2', None]
}

# Create the Random Forest model
rf_model = RandomForestRegressor()

# Create the RandomSearch object
random_search_rf = RandomizedSearchCV(
    estimator=rf_model,
    param_distributions=param_grid_rf,
    n_iter=10,
    cv=KFold(n_splits=10, shuffle=True))
```

In [70]:

```
# Fit the GridSearchCV object to training data
random_search_rf.fit(X_train_4e, y_train_4e)

# Print the best parameters found by GridSearchCV
print("Best parameters found for 4E Grade: ", random_search_rf.best_params_)

# Predict using the best Ridge model
best_rf_model_4e = random_search_rf.best_estimator_
```

In [71]:

```
# Make predictions using cross-validation
y_pred_4e_train_rf = cross_val_predict(best_rf_model_4e, X_train_4e, y_train_4e,
cv=KFold(n_splits=10, shuffle=True))
y_pred_4e_test_rf = cross_val_predict(best_rf_model_4e, X_test_4e, y_test_4e, cv=KFold(n_splits=10,
shuffle=True))
```

In [72]:
```
# Model Evaluation

# Root Mean Squared Error
rmse_train_4e_rf = (np.sqrt(mean_squared_error(y_train_4e, y_pred_4e_train_rf)) / TRP_data_FE['4E
Content in Concentrate (ounce/day)'].mean()) * 100
rmse_test_4e_rf = (np.sqrt(mean_squared_error(y_test_4e, y_pred_4e_test_rf)) / TRP_data_FE['4E
Content in Concentrate (ounce/day)'].mean()) * 100

print(f"Root Mean Squared Error for 4E Grade train: {rmse_train_4e_rf} %")
print(f"Root Mean Squared Error for 4E Grade test: {rmse_test_4e_rf} %")
```

In [73]:
```
# Model Evaluation

# R2
r2_train_4e_rf =  r2_score(y_train_4e, y_pred_4e_train_rf)
r2_test_4e_rf =  r2_score(y_test_4e, y_pred_4e_test_rf)

print(f"R-Squared for 4E Grade train: {r2_train_4e_rf}")
print(f"R-Squared for 4E Grade test: {r2_test_4e_rf}")print(
```

In [74]:
```
# Fit the GridSearchCV object to training data
random_search_rf.fit(X_train_chrome, y_train_chrome)

# Print the best parameters found by GridSearchCV
print("Best parameters found for Chromium Content: ", random_search_rf.best_params_)

# Predict using the best Ridge model
best_rf_model_chrome = random_search_rf.best_estimator_
```

In [75]:
```
# Make predictions using cross-validation
y_pred_chrome_train_rf = cross_val_predict(best_rf_model_chrome, X_train_chrome, y_train_chrome,
cv=KFold(n_splits=10, shuffle=True))
y_pred_chrome_test_rf = cross_val_predict(best_rf_model_chrome, X_test_chrome, y_test_chrome,
cv=KFold(n_splits=10, shuffle=True))
```

In [76]:

```
# Model Evaluation

# Root Mean Squared Error
rmse_train_chrome_rf = (np.sqrt(mean_squared_error(y_train_chrome, y_pred_chrome_train_rf)) /
TRP_data_FE['Chromium Content in Concentrate (%)'].mean()) * 100
rmse_test_chrome_rf = (np.sqrt(mean_squared_error(y_test_chrome, y_pred_chrome_test_rf)) /
TRP_data_FE['Chromium Content in Concentrate (%)'].mean()) * 100

print(f"Root Mean Squared Error for Chromium Content train: {rmse_train_chrome_rf} %")
print(f"Root Mean Squared Error for Chromium Content test: {rmse_test_chrome_rf} %")

# R2
r2_train_chrome_rf =  r2_score(y_train_chrome, y_pred_chrome_train_rf)
r2_test_chrome_rf =  r2_score(y_test_chrome, y_pred_chrome_test_rf)

print(f"R-Squared for Chromium Content train: {r2_train_chrome_rf}")
print(f"R-Squared for Chromium Content test: {r2_test_chrome_rf}")
```

In [77]:

```
# Define the indices of the target variables you want to plot
target_indices = [0, 1]

# Concatenate the training and testing data for plotting
X_combined_4e = pd.concat([X_train_4e, X_test_4e])
X_combined_chrome = pd.concat([X_train_chrome, X_test_chrome])
y_combined_4e = pd.concat([y_train_4e, y_test_4e])
y_combined_chrome = pd.concat([y_train_chrome, y_test_chrome])
y_pred_combined_4e = np.concatenate([y_pred_4e_train_rf, y_pred_4e_test_rf])
y_pred_combined_chrome = np.concatenate([y_pred_chrome_train_rf, y_pred_chrome_test_rf])

# Plot
fig, axs = plt.subplots(len(target_indices), 2, figsize=(14, 10))

for i, idx in enumerate(target_indices):
    if idx == 0:
        # Combined predictions for 4E Grade
        y_pred_combined_col = y_pred_combined_4e

        # Scatter plot with combined regression line for 4E Grade
        sns.scatterplot(ax=axs[i, 0], x=X_train_4e.index, y=y_train_4e, label='Train Data', color='skyblue')
        sns.scatterplot(ax=axs[i, 0], x=X_test_4e.index, y=y_test_4e, label='Test Data', color='lightcoral')
        sns.lineplot(ax=axs[i, 0], x=X_combined_4e.index, y=y_pred_combined_col, label='Prediction',
color='black')
        sns.lineplot(ax=axs[i, 0], x=X_combined_4e.index, y=y_pred_combined_col.mean(), label='Mean',
color='green')
        axs[i, 0].set_xlabel('Index')
        axs[i, 0].set_ylabel(f'4E Content in Concentrate (ounce/day)')
        axs[i, 0].legend()

        # Scatter plot with x=y line for 4E Grade
        sns.scatterplot(ax=axs[i, 1], x=y_combined_4e, y=y_pred_combined_col, label='Combined
Predicted', color='black')
        sns.regplot(ax=axs[i, 1], x=y_combined_4e, y=y_pred_combined_col, scatter=False, color='blue',
label='Regression Line')
        axs[i, 1].plot(y_combined_4e, y_combined_4e, color='green', linestyle='dashdot', label='x=y')
        axs[i, 1].set_xlabel(f'Actual 4E Content in Concentrate (ounce/day)')
        axs[i, 1].set_ylabel(f'Predicted 4E Content in Concentrate (ounce/day)')
        axs[i, 1].legend()

    elif idx == 1:
        # Combined predictions for Chromium Content
        y_pred_combined_col = y_pred_combined_chrome
```

```
        # Scatter plot with combined regression line for Chromium Content
        sns.scatterplot(ax=axs[i, 0], x=X_train_chrome.index, y=y_train_chrome, label='Train Data',
color='skyblue')
        sns.scatterplot(ax=axs[i, 0], x=X_test_chrome.index, y=y_test_chrome, label='Test Data',
color='lightcoral')
        sns.lineplot(ax=axs[i, 0], x=X_combined_chrome.index, y=y_pred_combined_col, label='Prediction',
color='black')
        sns.lineplot(ax=axs[i, 0], x=X_combined_chrome.index, y=y_pred_combined_col.mean(),
label='Mean', color='green')
        axs[i, 0].set_xlabel('Index')
        axs[i, 0].set_ylabel(f'Chromium Content in Concentrate (%)')
        axs[i, 0].legend()

        # Scatter plot with x=y line for Chromium Content
        sns.scatterplot(ax=axs[i, 1], x=y_combined_chrome, y=y_pred_combined_col, label='Combined
Predicted', color='black')
        sns.regplot(ax=axs[i, 1], x=y_combined_chrome, y=y_pred_combined_col, scatter=False,
color='blue', label='Regression Line')
        axs[i, 1].plot(y_combined_chrome, y_combined_chrome, color='green', linestyle='dashdot',
label='x=y')
        axs[i, 1].set_xlabel(f'Actual Chromium Content in Concentrate (%)')
        axs[i, 1].set_ylabel(f'Predicted Chromium Content in Concentrate (%)')
        axs[i, 1].legend()

# Adjust layout and save the plot
plt.tight_layout()
plt.savefig(path + 'Plant Performance Random Forest Regression.png', bbox_inches='tight')
plt.show()
```

C2.4. Extra Tree Regression

In [78]:
```
# Define the parameter grid for grid search
param_grid_et = {
    'n_estimators': randint(1, 300),
    'max_depth': randint(1, 15),
    'min_samples_leaf': [25, 30, 35, 40, 50, 100],
    'min_samples_leaf': randint(1, 40),
    'max_features': ['sqrt', 'log2', None]
}

# Create the ExtraTreesRegressor model
et_model = ExtraTreesRegressor(random_state=42)

# Create the RandomSearch object
random_search_et = RandomizedSearchCV(
    estimator=et_model,
    param_distributions=param_grid_et,
    n_iter=10,
    cv=KFold(n_splits=10, shuffle=True))
```

In [79]:
```
# Fit the GridSearchCV object to training data
random_search_et.fit(X_train_4e, y_train_4e)

# Print the best parameters found by GridSearchCV
print("Best parameters found for 4E Grade: ", random_search_et.best_params_)

# Predict using the best Ridge model
best_et_model_4e = random_search_et.best_estimator_
```

In [80]:
```
# Make predictions using cross-validation
y_pred_4e_train_et = cross_val_predict(best_et_model_4e, X_train_4e, y_train_4e,
cv=KFold(n_splits=10, shuffle=True))
y_pred_4e_test_et = cross_val_predict(best_et_model_4e, X_test_4e, y_test_4e, cv=KFold(n_splits=10,
shuffle=True))
```

In [81]:
```
# Model Evaluation

# Root Mean Squared Error
rmse_train_4e_et = (np.sqrt(mean_squared_error(y_train_4e, y_pred_4e_train_et)) / TRP_data_FE['4E Content in Concentrate (ounce/day)'].mean()) * 100
rmse_test_4e_et = (np.sqrt(mean_squared_error(y_test_4e, y_pred_4e_test_et)) / TRP_data_FE['4E Content in Concentrate (ounce/day)'].mean()) * 100

print(f"Root Mean Squared Error for 4E Content in Concentrate (ounce/day) train: {rmse_train_4e_et} %")
print(f"Root Mean Squared Error for 4E Content in Concentrate (ounce/day) test: {rmse_test_4e_et} %")
```

In [82]:
```
# Model Evaluation

# R2
r2_train_4e_et =  r2_score(y_train_4e, y_pred_4e_train_et)
r2_test_4e_et =  r2_score(y_test_4e, y_pred_4e_test_et)

print(f"R-Squared for 4E Content in Concentrate (ounce/day) train: {r2_train_4e_et}")
print(f"R-Squared for 4E Content in Concentrate (ounce/day) test: {r2_test_4e_et}")
```

In [83]:
```
# Fit the GridSearchCV object to training data
random_search_et.fit(X_train_chrome, y_train_chrome)

# Print the best parameters found by GridSearchCV
print("Best parameters found for Chromium Content: ", random_search_et.best_params_)

# Predict using the best Ridge model
best_et_model_chrome = random_search_et.best_estimator_
```

In [84]:
```
# Make predictions using cross-validation
y_pred_chrome_train_et = cross_val_predict(best_et_model_chrome, X_train_chrome, y_train_chrome, cv=KFold(n_splits=10, shuffle=True))
y_pred_chrome_test_et = cross_val_predict(best_et_model_chrome, X_test_chrome, y_test_chrome, cv=KFold(n_splits=10, shuffle=True))
```

In [85]:

```
# Model Evaluation

# Root Mean Squared Error
rmse_train_chrome_et = (np.sqrt(mean_squared_error(y_train_chrome, y_pred_chrome_train_et)) /
TRP_data_FE['Chromium Content in Concentrate (%)'].mean()) * 100
rmse_test_chrome_et = (np.sqrt(mean_squared_error(y_test_chrome, y_pred_chrome_test_et)) /
TRP_data_FE['Chromium Content in Concentrate (%)'].mean()) * 100

print(f"Root Mean Squared Error for Chromium Content train: {rmse_train_chrome_et} %")
print(f"Root Mean Squared Error for Chromium Content test: {rmse_test_chrome_et} %")

# R2
r2_train_chrome_et =  1 - (1 - r2_score(y_train_chrome, y_pred_chrome_train_et) * (len(y_train_chrome)
- 1) / (len(y_train_chrome) - X_train_chrome.shape[1] - 1))
r2_test_chrome_et =  1 - (1 - r2_score(y_test_chrome, y_pred_chrome_test_et) * (len(y_test_chrome) - 1) /
(len(y_test_chrome) - X_test_chrome.shape[1] - 1))

print(f"R-Squared for Chromium Content train: {r2_train_chrome_et}")
print(f"R-Squared for Chromium Content test: {r2_test_chrome_et}")
```

In [86]:

```
# Define the indices of the target variables you want to plot
target_indices = [0, 1]

# Concatenate the training and testing data for plotting
X_combined_4e = pd.concat([X_train_4e, X_test_4e])
X_combined_chrome = pd.concat([X_train_chrome, X_test_chrome])
y_combined_4e = pd.concat([y_train_4e, y_test_4e])
y_combined_chrome = pd.concat([y_train_chrome, y_test_chrome])
y_pred_combined_4e = np.concatenate([y_pred_4e_train_et, y_pred_4e_test_et])
y_pred_combined_chrome = np.concatenate([y_pred_chrome_train_et, y_pred_chrome_test_et])

# Plot
fig, axs = plt.subplots(len(target_indices), 2, figsize=(14, 10))

for i, idx in enumerate(target_indices):
    if idx == 0:
        # Combined predictions for 4E Grade
        y_pred_combined_col = y_pred_combined_4e

        # Scatter plot with combined regression line for 4E Grade
        sns.scatterplot(ax=axs[i, 0], x=X_train_4e.index, y=y_train_4e, label='Train Data', color='skyblue')
        sns.scatterplot(ax=axs[i, 0], x=X_test_4e.index, y=y_test_4e, label='Test Data', color='lightcoral')
        sns.lineplot(ax=axs[i, 0], x=X_combined_4e.index, y=y_pred_combined_col, label='Prediction',
color='black')
        sns.lineplot(ax=axs[i, 0], x=X_combined_4e.index, y=y_pred_combined_col.mean(), label='Mean',
color='green')
        axs[i, 0].set_xlabel('Index')
        axs[i, 0].set_ylabel(f'4E Content in Concentrate (ounce/day)')
        axs[i, 0].legend()

        # Scatter plot with x=y line for 4E Grade
        sns.scatterplot(ax=axs[i, 1], x=y_combined_4e, y=y_pred_combined_col, label='Combined
Predicted', color='black')
        sns.regplot(ax=axs[i, 1], x=y_combined_4e, y=y_pred_combined_col, scatter=False, color='blue',
label='Regression Line')
        axs[i, 1].plot(y_combined_4e, y_combined_4e, color='green', linestyle='dashdot', label='x=y')
        axs[i, 1].set_xlabel(f'Actual 4E Content in Concentrate (ounce/day)')
        axs[i, 1].set_ylabel(f'Predicted 4E Content in Concentrate (ounce/day)')
        axs[i, 1].legend()

    elif idx == 1:
        # Combined predictions for Chromium Content
        y_pred_combined_col = y_pred_combined_chrome
```

```
        # Scatter plot with combined regression line for Chromium Content
        sns.scatterplot(ax=axs[i, 0], x=X_train_chrome.index, y=y_train_chrome, label='Train Data',
color='skyblue')
        sns.scatterplot(ax=axs[i, 0], x=X_test_chrome.index, y=y_test_chrome, label='Test Data',
color='lightcoral')
        sns.lineplot(ax=axs[i, 0], x=X_combined_chrome.index, y=y_pred_combined_col, label='Prediction',
color='black')
        sns.lineplot(ax=axs[i, 0], x=X_combined_chrome.index, y=y_pred_combined_col.mean(),
label='Mean', color='green')
        axs[i, 0].set_xlabel('Index')
        axs[i, 0].set_ylabel(f'Chromium Content in Concentrate (%)')
        axs[i, 0].legend()

        # Scatter plot with x=y line for Chromium Content
        sns.scatterplot(ax=axs[i, 1], x=y_combined_chrome, y=y_pred_combined_col, label='Combined
Predicted', color='black')
        sns.regplot(ax=axs[i, 1], x=y_combined_chrome, y=y_pred_combined_col, scatter=False,
color='blue', label='Regression Line')
        axs[i, 1].plot(y_combined_chrome, y_combined_chrome, color='green', linestyle='dashdot',
label='x=y')
        axs[i, 1].set_xlabel(f'Actual Chromium Content in Concentrate (%)')
        axs[i, 1].set_ylabel(f'Predicted Chromium Content in Concentrate (%)')
        axs[i, 1].legend()

# Adjust layout and save the plot
plt.tight_layout()
plt.savefig(path + 'Plant Performance Extra Tree Regression.png', bbox_inches='tight')
plt.show()
```

C2.5. k-Nearest Neighbour Regression

In [87]:

```
# Define the parameter grid for grid search
param_grid_knn = {
    'n_neighbors': np.arange(1, 41, 2) # [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29]
}

# Create the kNN model
knn_model = KNeighborsRegressor()

# Create the GridSearchCV object
grid_search_knn = GridSearchCV(estimator=knn_model,
                               param_grid=param_grid_knn,
                               cv=KFold(n_splits=10, shuffle=True)
                              )
```

In [88]:

```
# Fit the GridSearchCV object to training data
grid_search_knn.fit(X_train_4e, y_train_4e)

# Print the best parameters found by GridSearchCV
print("Best parameters found for 4E Grade: ", grid_search_knn.best_params_)

# Predict using the best Ridge model
best_knn_model_4e = grid_search_knn.best_estimator_
```

In [89]:

```
# Make predictions using cross-validation
y_pred_4e_train_knn = cross_val_predict(best_knn_model_4e, X_train_4e, y_train_4e,
cv=KFold(n_splits=10, shuffle=True))
y_pred_4e_test_knn = cross_val_predict(best_knn_model_4e, X_test_4e, y_test_4e,
cv=KFold(n_splits=10, shuffle=True))
```

In [90]:
```
# Model Evaluation

# Root Mean Squared Error
rmse_train_4e_knn = (np.sqrt(mean_squared_error(y_train_4e, y_pred_4e_train_knn)) /
TRP_data_FE['4E Content in Concentrate (ounce/day)'].mean()) * 100
rmse_test_4e_knn = (np.sqrt(mean_squared_error(y_test_4e, y_pred_4e_test_knn)) / TRP_data_FE['4E
Content in Concentrate (ounce/day)'].mean()) * 100

print(f"Root Mean Squared Error for 4E Grade train: {rmse_train_4e_knn} %")
print(f"Root Mean Squared Error for 4E Grade test: {rmse_test_4e_knn} %")
```

In [91]:
```
# Model Evaluation

# R2
r2_train_4e_knn =  r2_score(y_train_4e, y_pred_4e_train_knn)
r2_test_4e_knn =  r2_score(y_test_4e, y_pred_4e_test_knn)

print(f"R-Squared for 4E Grade train: {r2_train_4e_knn}")
print(f"R-Squared for 4E Grade test: {r2_test_4e_knn}")
```

In [92]:
```
# Fit the GridSearchCV object to training data
grid_search_knn.fit(X_train_chrome, y_train_chrome)

# Print the best parameters found by GridSearchCV
print("Best parameters found for Chromium Content: ", grid_search_knn.best_params_)

# Predict using the best Ridge model
best_knn_model_chrome = grid_search_knn.best_estimator_
```

In [93]:
```
# Make predictions using cross-validation
y_pred_chrome_train_knn = cross_val_predict(best_knn_model_chrome, X_train_chrome,
y_train_chrome, cv=KFold(n_splits=10, shuffle=True))
y_pred_chrome_test_knn = cross_val_predict(best_knn_model_chrome, X_test_chrome, y_test_chrome,
cv=KFold(n_splits=10, shuffle=True))
```

In [94]:

```
# Model Evaluation

# Root Mean Squared Error
rmse_train_chrome_knn = (np.sqrt(mean_squared_error(y_train_chrome, y_pred_chrome_train_knn)) /
TRP_data_FE['Chromium Content in Concentrate (%)'].mean()) * 100
rmse_test_chrome_knn = (np.sqrt(mean_squared_error(y_test_chrome, y_pred_chrome_test_knn)) /
TRP_data_FE['Chromium Content in Concentrate (%)'].mean()) * 100

print(f"Root Mean Squared Error for Chromium Content train: {rmse_train_chrome_knn} %")
print(f"Root Mean Squared Error for Chromium Content test: {rmse_test_chrome_knn} %")

# R2
r2_train_chrome_knn =  r2_score(y_train_chrome, y_pred_chrome_train_knn)
r2_test_chrome_knn =  r2_score(y_test_chrome, y_pred_chrome_test_knn)

print(f"R-Squared for Chromium Content train: {r2_train_chrome_knn}")
print(f"R-Squared for Chromium Content test: {r2_test_chrome_knn}")
```

In [95]:

```
# Define the indices of the target variables you want to plot
target_indices = [0, 1]

# Concatenate the training and testing data for plotting
X_combined_4e = pd.concat([X_train_4e, X_test_4e])
X_combined_chrome = pd.concat([X_train_chrome, X_test_chrome])
y_combined_4e = pd.concat([y_train_4e, y_test_4e])
y_combined_chrome = pd.concat([y_train_chrome, y_test_chrome])
y_pred_combined_4e = np.concatenate([y_pred_4e_train_et, y_pred_4e_test_et])
y_pred_combined_chrome = np.concatenate([y_pred_chrome_train_knn, y_pred_chrome_test_knn])

# Plot
fig, axs = plt.subplots(len(target_indices), 2, figsize=(14, 10))

for i, idx in enumerate(target_indices):
    if idx == 0:
        # Combined predictions for 4E Grade
        y_pred_combined_col = y_pred_combined_4e

        # Scatter plot with combined regression line for 4E Grade
        sns.scatterplot(ax=axs[i, 0], x=X_train_4e.index, y=y_train_4e, label='Train Data', color='skyblue')
        sns.scatterplot(ax=axs[i, 0], x=X_test_4e.index, y=y_test_4e, label='Test Data', color='lightcoral')
        sns.lineplot(ax=axs[i, 0], x=X_combined_4e.index, y=y_pred_combined_col, label='Prediction',
color='black')
        sns.lineplot(ax=axs[i, 0], x=X_combined_4e.index, y=y_pred_combined_col.mean(), label='Mean',
color='green')
        axs[i, 0].set_xlabel('Index')
        axs[i, 0].set_ylabel(f'4E Content in Concentrate (ounce/day)')
        axs[i, 0].legend()

        # Scatter plot with x=y line for 4E Grade
        sns.scatterplot(ax=axs[i, 1], x=y_combined_4e, y=y_pred_combined_col, label='Combined
Predicted', color='black')
        sns.regplot(ax=axs[i, 1], x=y_combined_4e, y=y_pred_combined_col, scatter=False, color='blue',
label='Regression Line')
        axs[i, 1].plot(y_combined_4e, y_combined_4e, color='green', linestyle='dashdot', label='x=y')
        axs[i, 1].set_xlabel(f'Actual 4E Content in Concentrate (ounce/day)')
        axs[i, 1].set_ylabel(f'Predicted 4E Content in Concentrate (ounce/day)')
        axs[i, 1].legend()

    elif idx == 1:
        # Combined predictions for Chromium Content
        y_pred_combined_col = y_pred_combined_chrome
```

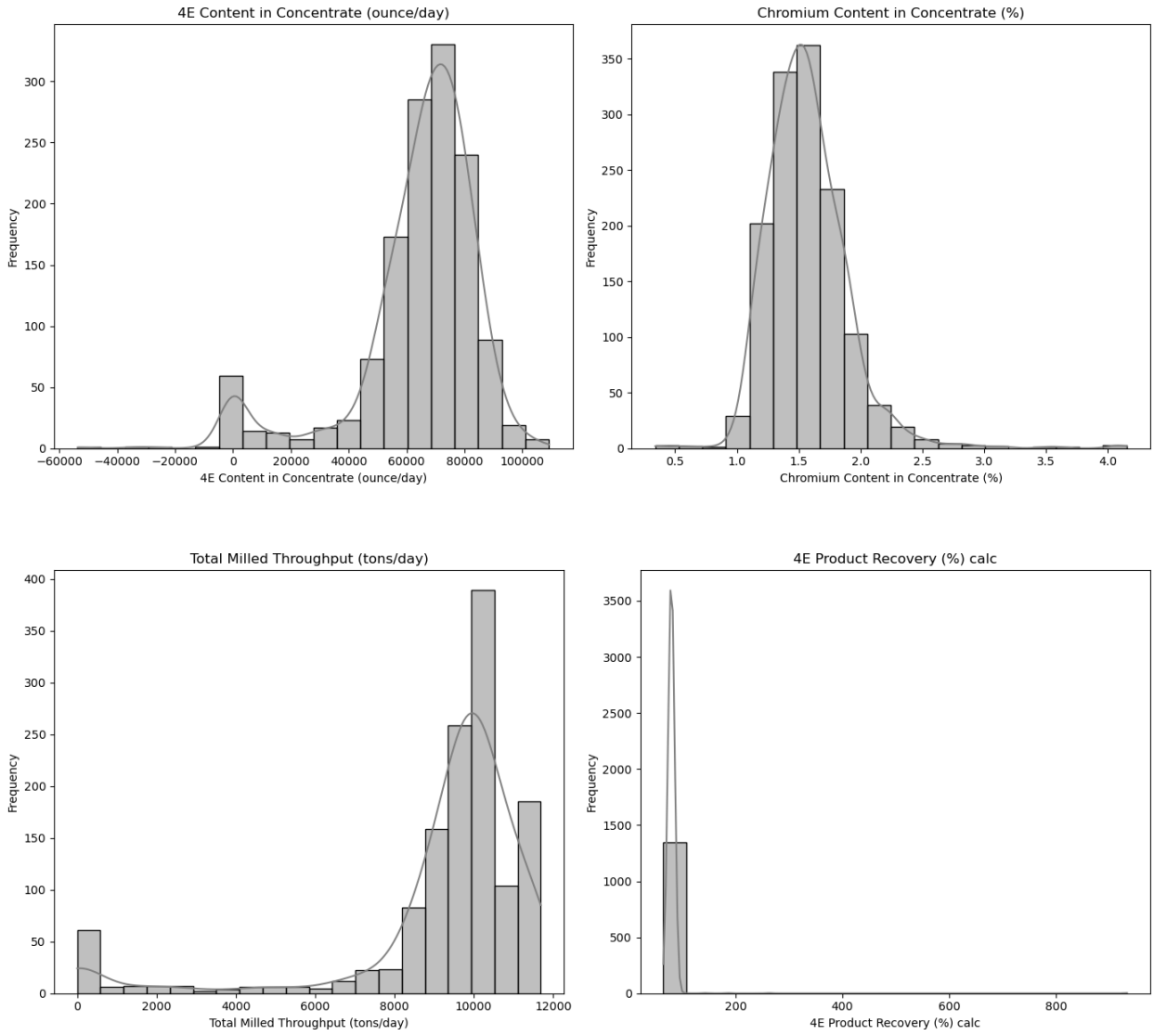
```
        # Scatter plot with combined regression line for Chromium Content
        sns.scatterplot(ax=axs[i, 0], x=X_train_chrome.index, y=y_train_chrome, label='Train Data',
color='skyblue')
        sns.scatterplot(ax=axs[i, 0], x=X_test_chrome.index, y=y_test_chrome, label='Test Data',
color='lightcoral')
        sns.lineplot(ax=axs[i, 0], x=X_combined_chrome.index, y=y_pred_combined_col, label='Prediction',
color='black')
        sns.lineplot(ax=axs[i, 0], x=X_combined_chrome.index, y=y_pred_combined_col.mean(),
label='Mean', color='green')
        axs[i, 0].set_xlabel('Index')
        axs[i, 0].set_ylabel(f'Chromium Content in Concentrate (%)')
        axs[i, 0].legend()

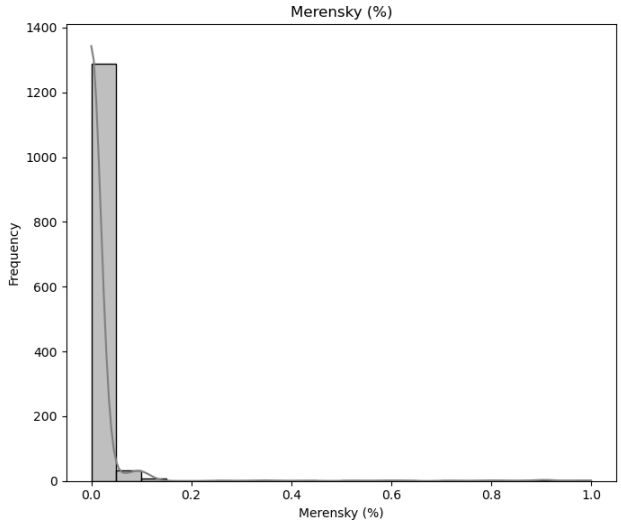
        # Scatter plot with x=y line for Chromium Content
        sns.scatterplot(ax=axs[i, 1], x=y_combined_chrome, y=y_pred_combined_col, label='Combined
Predicted', color='black')
        sns.regplot(ax=axs[i, 1], x=y_combined_chrome, y=y_pred_combined_col, scatter=False,
color='blue', label='Regression Line')
        axs[i, 1].plot(y_combined_chrome, y_combined_chrome, color='green', linestyle='dashdot',
label='x=y')
        axs[i, 1].set_xlabel(f'Actual Chromium Content in Concentrate (%)')
        axs[i, 1].set_ylabel(f'Predicted Chromium Content in Concentrate (%)')
        axs[i, 1].legend()

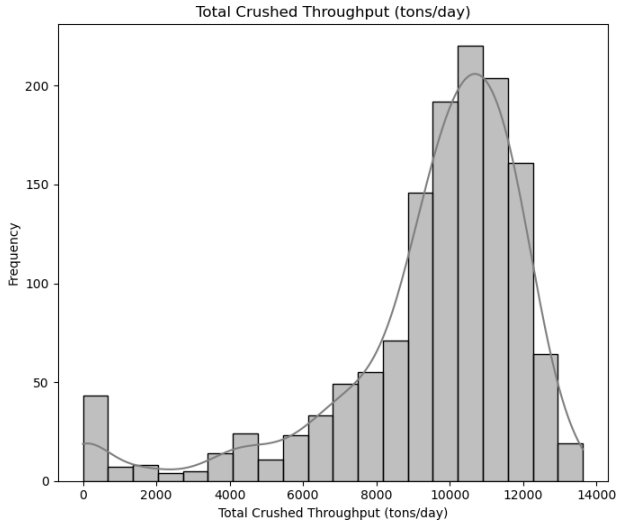
# Adjust layout and save the plot
plt.tight_layout()
plt.savefig(path + 'Plant Performance kNN Regression.png', bbox_inches='tight')
plt.show()
```

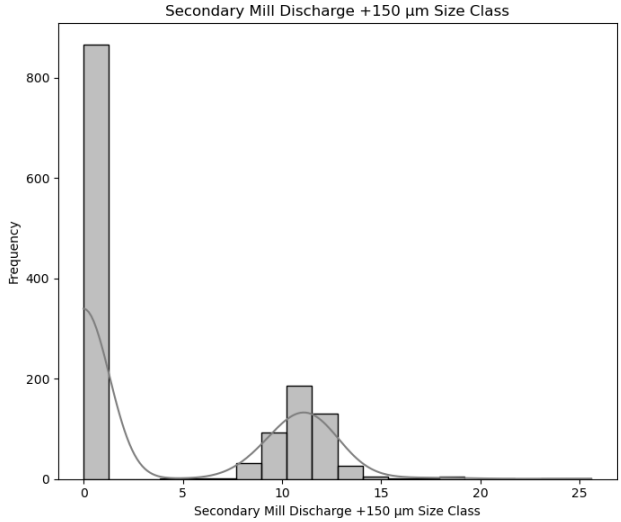
APPENDIX D: Descriptive Statistics of the Database

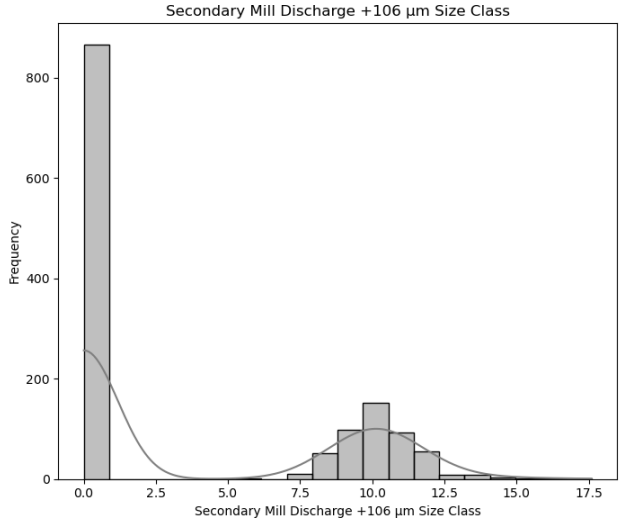
This appendix presents the dataset's descriptive statistics, providing insights into its distribution, central tendency, and variability.

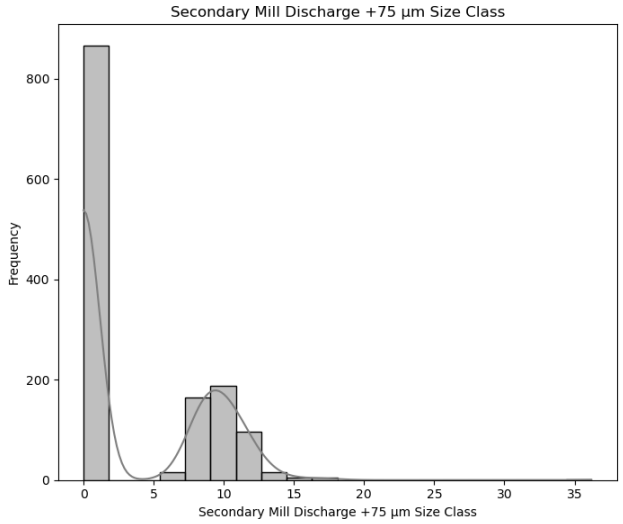


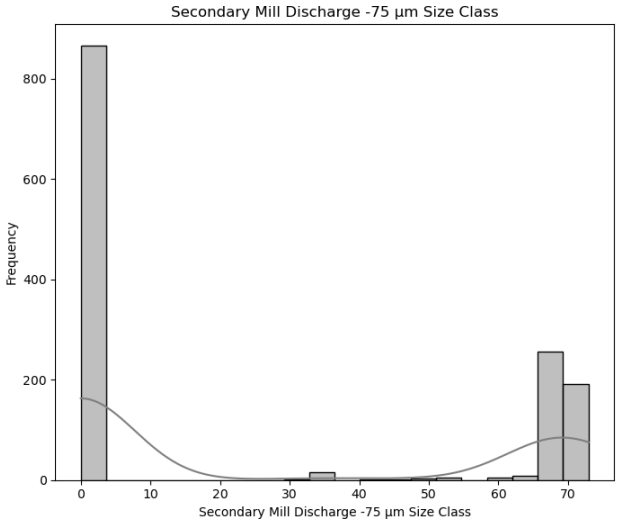

Merensky (%)
Frequency
Merensky (%)
0
200
400
600
800
1000
1200
1400
0.0
0.2
0.4
0.6
0.8
1.0

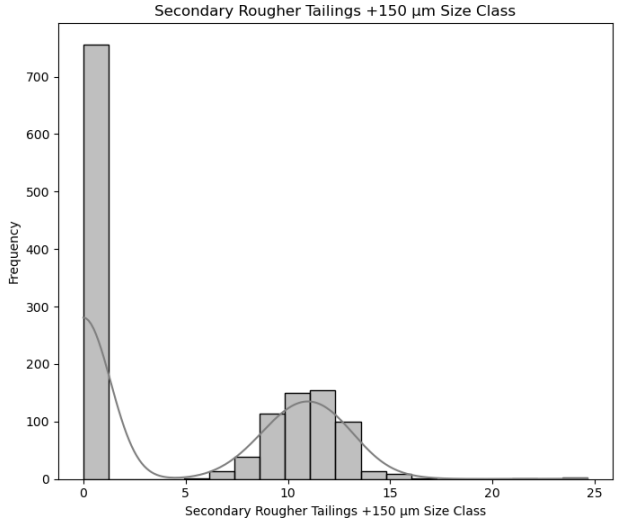

Total Crushed Throughput (tons/day)
Frequency
Total Crushed Throughput (tons/day)
0
50
100
150
200
0
2000
4000
6000
8000
10000
12000
14000

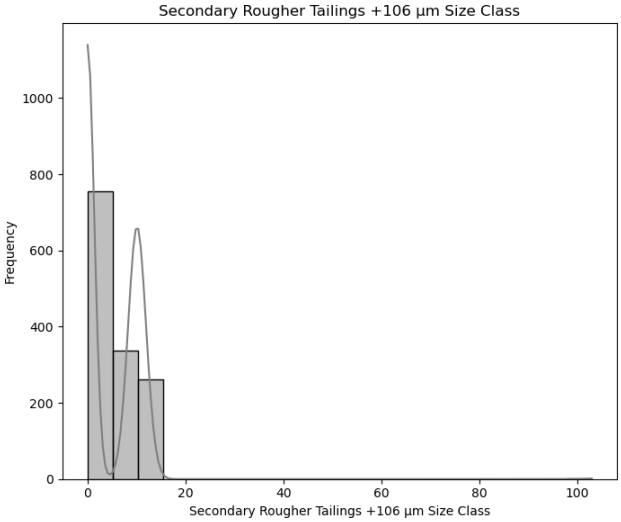

Secondary Mill Discharge +150 µm Size Class
Frequency
Secondary Mill Discharge +150 µm Size Class
0
200
400
600
800
0
5
10
15
20
25

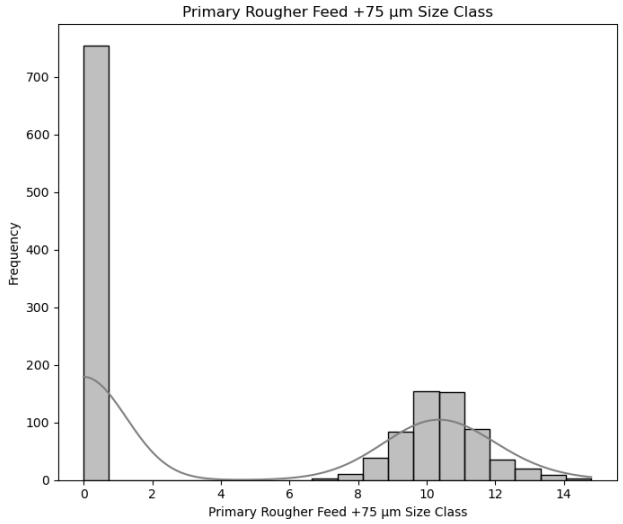

Secondary Mill Discharge +106 µm Size Class
Frequency
Secondary Mill Discharge +106 µm Size Class
0
200
400
600
800
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

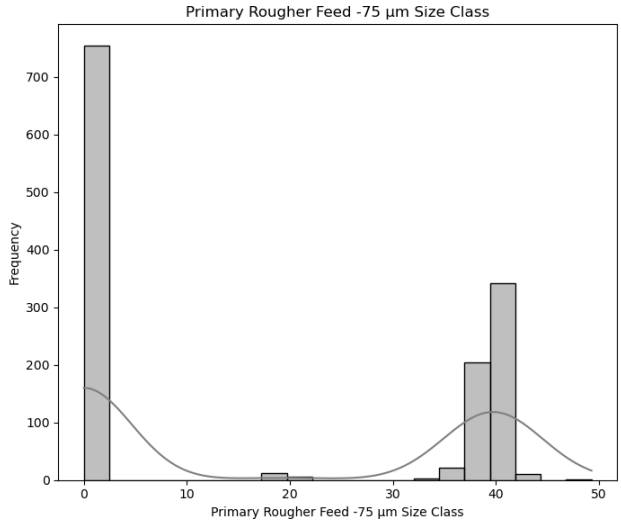

Secondary Mill Discharge +75 µm Size Class
Frequency
Secondary Mill Discharge +75 µm Size Class

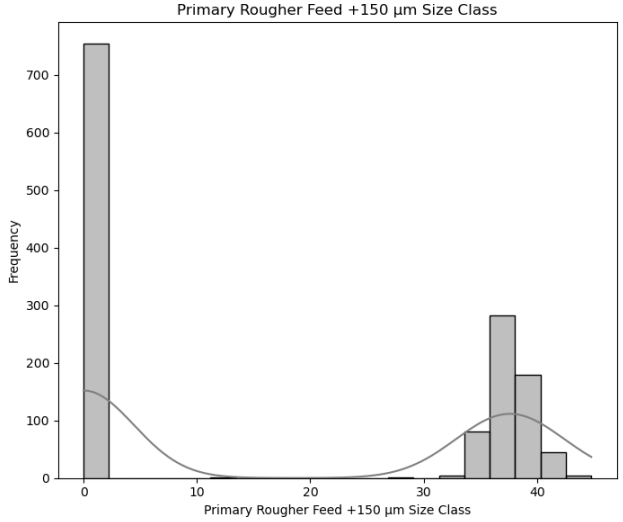

Secondary Mill Discharge -75 µm Size Class
Frequency
Secondary Mill Discharge -75 µm Size Class

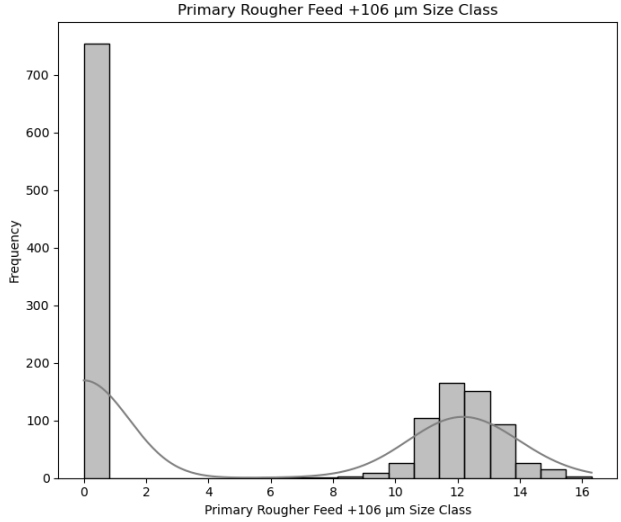

Secondary Rougher Tailings +150 µm Size Class
Frequency
Secondary Rougher Tailings +150 µm Size Class

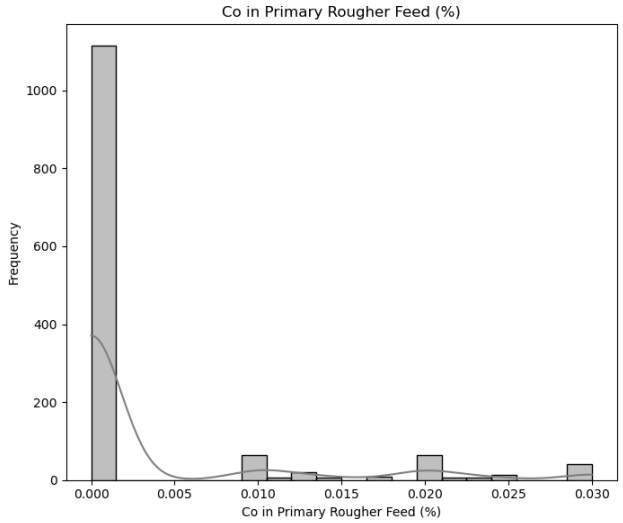

Secondary Rougher Tailings +106 µm Size Class
Frequency
Secondary Rougher Tailings +106 µm Size Class

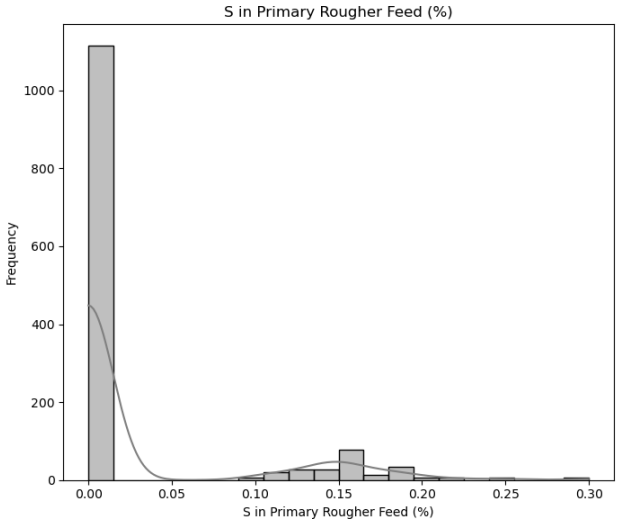

Primary Rougher Feed +75 µm Size Class
Frequency
Primary Rougher Feed +75 µm Size Class

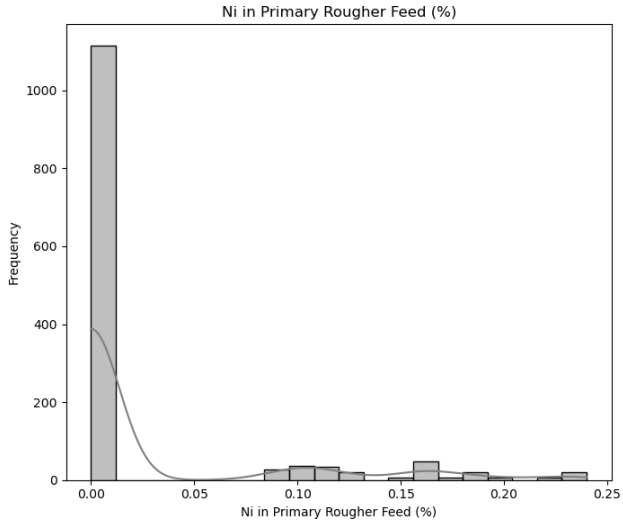

Primary Rougher Feed -75 µm Size Class
Frequency
Primary Rougher Feed -75 µm Size Class

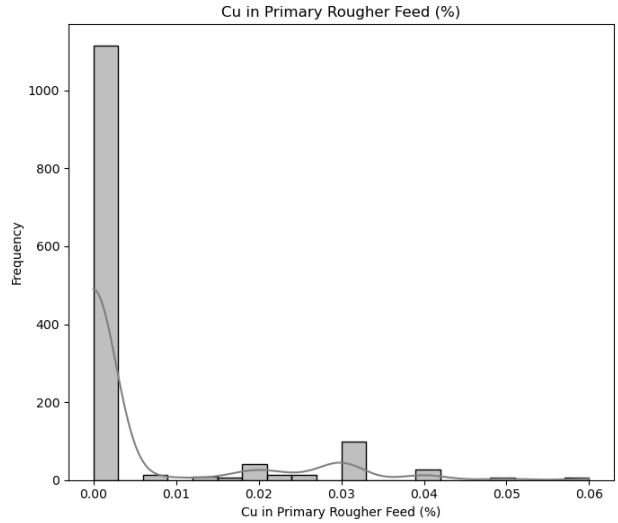

Primary Rougher Feed +150 µm Size Class
Frequency
Primary Rougher Feed +150 µm Size Class

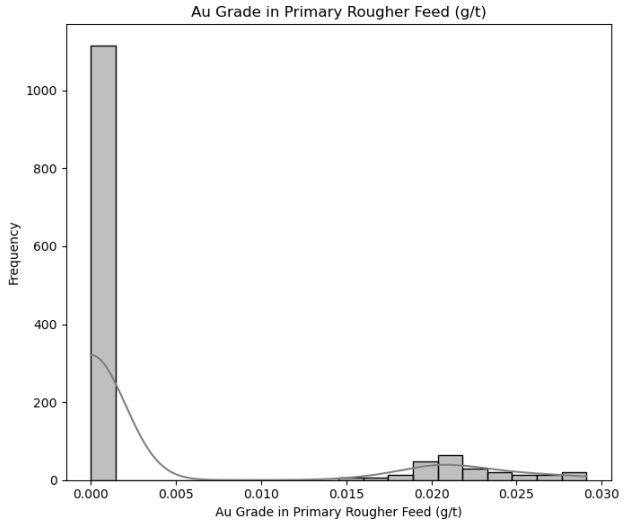

Primary Rougher Feed +106 µm Size Class
Frequency
Primary Rougher Feed +106 µm Size Class

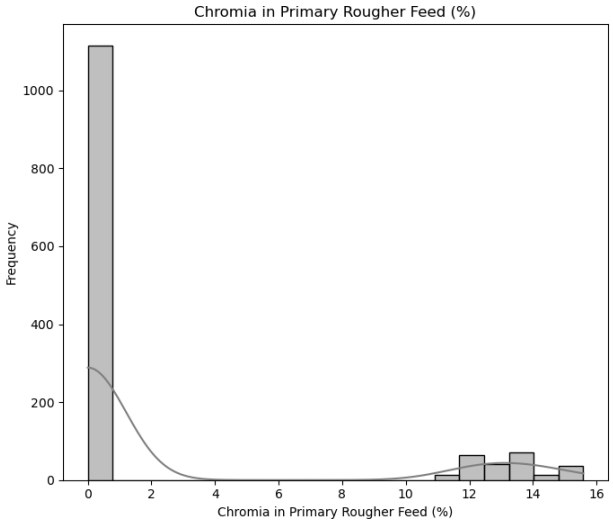

Co in Primary Rougher Feed (%)
Frequency
Co in Primary Rougher Feed (%)

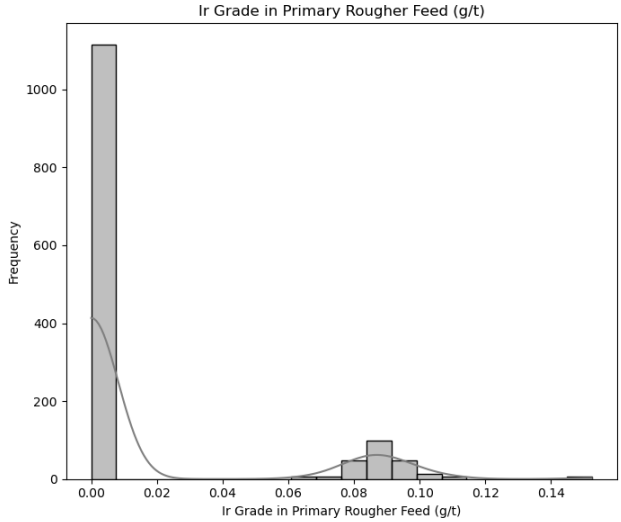

S in Primary Rougher Feed (%)
Frequency
S in Primary Rougher Feed (%)

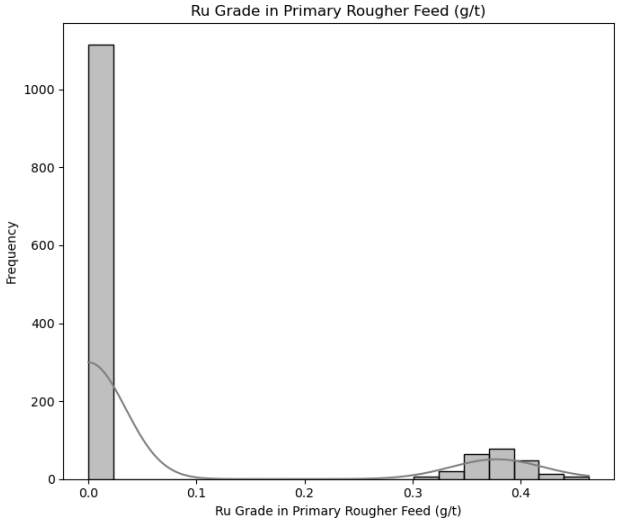

Ni in Primary Rougher Feed (%)
Frequency
Ni in Primary Rougher Feed (%)


Cu in Primary Rougher Feed (%)
Frequency
Cu in Primary Rougher Feed (%)


Au Grade in Primary Rougher Feed (g/t)
Frequency
Au Grade in Primary Rougher Feed (g/t)


Chromia in Primary Rougher Feed (%)
Frequency
Chromia in Primary Rougher Feed (%)


Ir Grade in Primary Rougher Feed (g/t)
Frequency
Ir Grade in Primary Rougher Feed (g/t)


Ru Grade in Primary Rougher Feed (g/t)
Frequency
Ru Grade in Primary Rougher Feed (g/t)

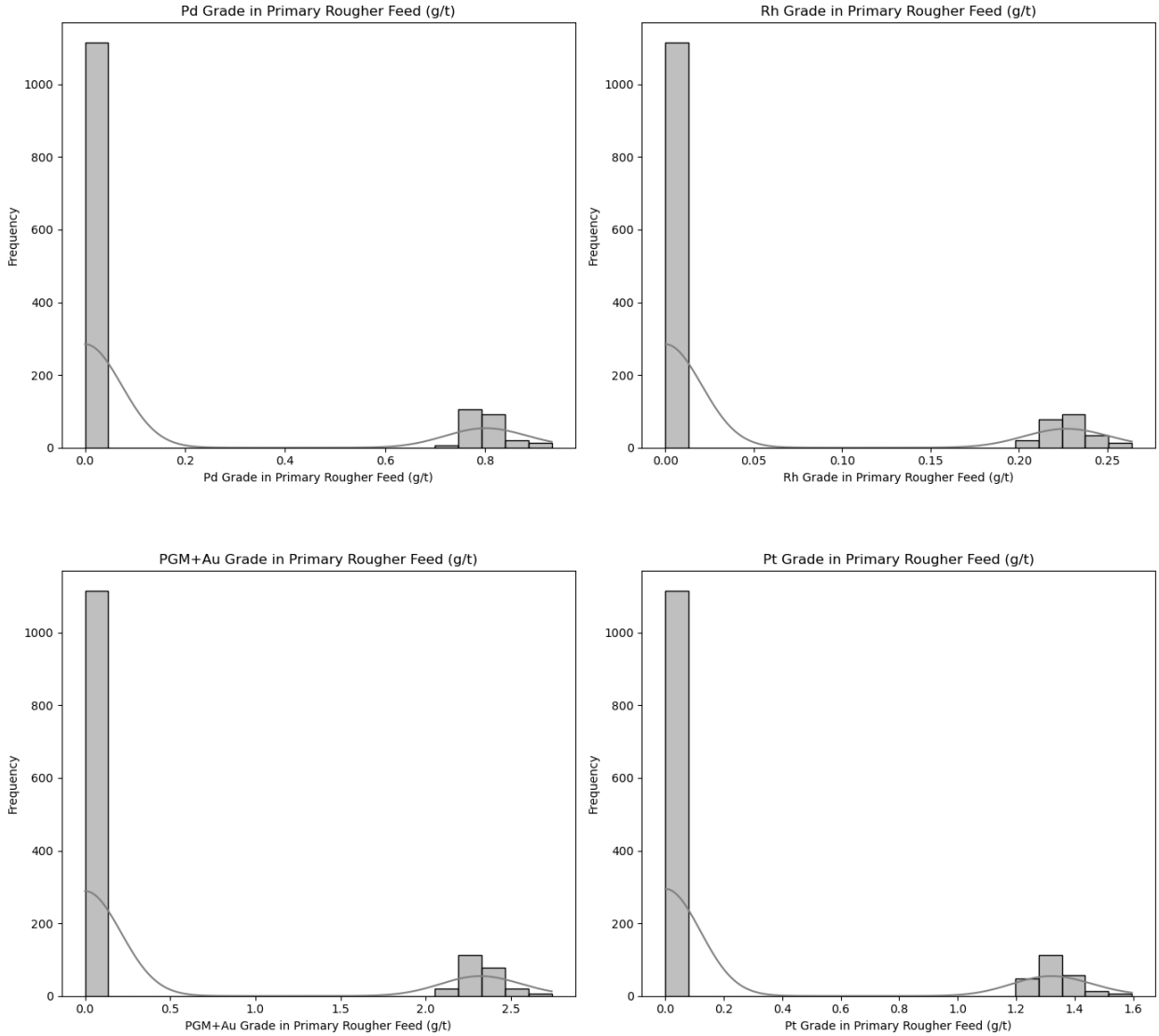


Figure D7.1: Operational and Ore Mineralogy data distribution