BOOSTING
Insights • Impact • Data

RioTinto

AI/Analytics Capstone: The Fueling Project

Project 7: *Implementation of multi-dimensional features and XGBoost algorithms for open-pit mine truck fuel consumption: An Australian case study*

Assignment 3

Gabriel Romeo Vivaldi (14378893) || Katie Nguyen (24504680)

Princess Marvella Liuz (14464440) || Valerie Kusumo (24870572)

Dominic Jason Gunawan (24612184)

Table of Contents

1. Executive Summary

BOOSTING is a data analytics/ machine learning-focused company. It was founded in 2024 by Gabriel Romeo Vivaldi, Katie Nguyen, Princess Marvella Luiz, Valerie Kusumo and Dominic Jason Gunawan. In this project, Boosting will analyse how much fuel various mine trucks use during transportation cycles. This is done by implementing the XGBoost algorithm and multi-dimensional characteristics to propose a prediction model for mine truck fuel consumption. Additionally, alternative algorithms such as Random Forest and Neural Networks were built and tested against the success of the XGBoost model.

According to the Onboard Truck Scale Alternative article, the estimated cost of diesel reaches $1.10/litre(Mine Payload Technologies, 2023). With a 10% overload, which increases the fuel consumption per hour, and other factors such as fleet count and the average operating hours, the price to operate a significant number of mine trucks can reach up to $3,668,500 per annum (Mine Payload Technologies, 2023), taking up a significant portion of up to 22 % of the total operational costs (Wang et al, 2021). Unfortunately, existing research on mine truck fuel consumption faces difficulties due to low monitoring accuracy and unclear consumption patterns.

To increase productivity, safety, and efficiency in the mining process, artificial intelligence has found various applications in the open-pit mining industry. In production planning and scheduling, AI and machine learning have been used to support making tactical decisions for the long- and short-term mining operation plan, such as choosing how to allocate resources and extract ore to meet production and economic goals.

The data for this project was obtained from a reliable third-party source with access to the Western Sydney Mining dataset, an Australian-owned data platform. The initial datasheet is divided into three smaller sheets to explore the dataset more efficiently: CycleData which contains cycle and mining operation-related attributes, LocationData which contains location information of the equipments and DelayData which contains planned and unplanned delay schedules.

Using Microsoft PowerBI, initial findings can be derived from the three datasets. It was found from the cycle dataset that fuel consumption stays around 0.054, payloads weigh around 220 tonnes, and a cycle usually lasts more than 1500. After a thorough analysis of the delay dataset, it appears that none of the delays that occurred were by primary trucks used, as seen in cycle data. The highest delay count of 5646 was by a secondary truck called “CAT 793F CMD”. The location dataset contains the coordinates of the machinery locations, using azure maps in PowerBI, the exact location of the mine was found. The mine is a part of the Jimblebar Hub located in the Pilbara region in Western Australia. It is fully owned and controlled by BHP, one of five mines and four processing hubs that make up Western Australia Iron Ore (BHP, 2024).

The research timeline starts with the collection of the dataset along with our data exploration and initial analysis. Design of Experiment is then conducted to find the correlations between attributes. Data preprocessing comes next followed by training the XGBoost model and the

other alternative algorithms. The models are then evaluated and re-trained if results are poor. After finding the most accurate predictive models, the chosen optimisation algorithm, random search, is then applied to optimise the models. Models are then re-trained again to get the most optimised predictive models which are then the final models we present as the solution.

In data preprocessing, the team has removed columns that have missing values, and uses feature scaling through a standard scaler for data transformation to enhance the accuracy of the model. After data preprocessing, the team builts the models.

For XGBoost, the team uses libraries such as sklearn and XGBoost in python. Data is split into proportions, 80% for training and 20% for testing. The XGBoost model's prediction results demonstrate an extremely positive performance. The Mean Squared Error (MSE) is 2.216283e-05. The better the model in regression analysis, the closer this number is to 0. This assertion is further supported by the Mean Absolute Error (MAE), which has a value of 0.001010013454. The average absolute value difference between the actual and anticipated values is measured by this. Finally, the average size of errors in the projected values is measured by the Root Mean Squared Error (RMSE). The result is 0.0047077 which is very close to the MAE and near 0, indicating excellent predictive accuracy.

To improve the model, random search as mentioned before is applied along with monte carlo simulation. The implementation of the Monte Carlo simulation aims to mitigate the randomness introduced by using random search, with the hope of finding an even better set of hyperparameters. The optimised model has an MSE that is 50.75% better than the original model. This indicates that the optimised model has substantially reduced the error compared to the original model. Additionally, the optimised model is even better at predicting the values that are relatively far from the clump of data points.

For Random Forest, the team uses sklearn library namely train_test_split for data splitting and LabelEncoder for encoding categorical variables. A Random Forest regressor model is then initialised with 100 decision trees (n_estimators) and the same starting point for the same sequence of random numbers (random_state= 42). In this scenario, the MSE is valued at (7.77e-05), indicating that the predicted outcomes are similar to actual values as the value is very close to zero. In addition, the coefficient of determination (r2) of 0.872 implies that the linear regression model has an accuracy rate of 87.2 %. This is considered an excellent r2 value, indicating that the model fits the data relatively well. After applying monte carlo simulation and random search, the average MSE is 5.06e-05, which is an even closer value to zero compared to 7.77e-05(mse value pre-optimisation). This indicates that a higher accuracy is prominent after the model has been optimised.

For Neural Networks, the team implemented keras library from tensorflow in python and other libraries similar to XGBoost and Random Forest implementation such as pandas and sklearn. The mean absolute error for the base model was found to be 0.0416. After implementing random search into the model, the MAE for the predicted values and the real values is calculated to be 0.01906, smaller than the previous model by 2 times. Compared to

the original algorithm without the random search, the MAE is smaller so we can expect a closer result of the predicted values compared with the real values.

During this project, the team has encountered several challenges which ranges from finding related dataset, into having multiple columns with NULL values as mentioned before, and incorrect data types. The team has to deal with these issues before the development of the models hence slowing down the modelling phase.

After the development of the XGBoost model and the alternative models, deployment is discussed in this report for the client to execute after sign-off. Boosting suggests using the Edge deployment strategy when it comes to deployment techniques. The Edge deployment technique enables the direct deployment of machine learning models on the user's device, such as a smartphone. Employees on site can use the model to forecast how much gasoline will be used, which can help with decision-making. To implement Edge Deployment, Rio Tinto needs to decide on the deployment environment, database infrastructure, container for models, and troubleshooting techniques such as alerts and logging.

In conclusion, the BOOSTING team's study shows how to effectively apply the XGBoost algorithm and other machine learning techniques, such as random forest and neural networks, to estimate open-pit mine vehicles' fuel consumption with a high degree of accuracy. The performance measurements, however, showed that the XGboost model generated the maximum accuracy. This not only demonstrates the model's efficacy and efficiency in managing such intricate and sizable datasets with a variety of attributes, but also its potential use in mining operations in the future.

This project offers an optimistic future and benefits in terms of cost reduction, operational efficiency, and environmental protection. It also serves as a benchmark for the integration of machine learning into industrial applications. The project itself may serve as the foundational study for further application in a variety of industries.

The success of using XGBoost algorithm in building fuel consumption prediction for open-pit mining trucks opens several opportunities for future research and development. One of the works BOOSTING is planning to do is enhancing the machine learning models by using a different optimisation algorithm and other machine learning algorithms such as deep learning.

Additionally, a lot of our attributes have missing values which were removed. These additional data would allow the model to take wider range of variables and factors into account and evaluate the effect of them on fuel consumption. Having a more complete dataset in the future would strengthen the model’s prediction power, thus improving its accuracy and reliability.

Furthermore, in the future the team could implement this model in different industries such as logistic, agriculture or construction where heavy machinery/trucks are also used in the operation.

2. Boosting

2.1 About Our Company

BOOSTING is a data analytics/ machine learning-focused company. It was founded in 2024 by Gabriel Romeo Vivaldi, Katie Nguyen, Princess Marvella Luiz, Valerie Kusumo and Dominic Jason Gunawan. Our business proposition revolves around leveraging AI and machine learning to find valuable insights from data to help drive impactful business decisions and performance.

2.2 Mission Statement

BOOSTING's mission is to help businesses make more confident decisions using data-driven insights. By using artificial intelligence and machine learning, BOOSTING can increase the accuracy and relevancy of insights to drive business decisions.

3. The Team

3.1 Meet the Team

3.1.1 Gabriel Romeo Vivaldi

Romeo is a University of Technology Sydney student pursuing a Bachelor of Science in IT, majoring in Data Analytics and Cybersecurity. With a proven track record as a data analyst within the insurance sector, he has demonstrated proficiency in data analysis and visualisation. Romeo is determined to leverage his comprehensive skill set to provide valuable insights and support clients in extracting crucial information from raw data, showcasing his commitment to excellence in the field.

3.1.2 Katie Nguyen

Katie Nguyen is a third-year student at the University of Technology Sydney, majoring in Data Analysis and Cybersecurity. Katie has been actively engaged in academic projects, gaining knowledge and technical experiences through various tasks. One such project involved building a machine learning model using CNN (Convolutional Neural Networks) for sign language recognition which achieves 98% accuracy. This project has demonstrated Katie's technical skills and understanding of data analysis concepts alongside a passion for innovation and problem-solving. Moreover, she has a background in customer service and working with children which possesses strong communication skills. These skill sets will be a valuable asset in facilitating collaboration and providing data-driven insights, assisting the team in completing the project successfully.

3.1.3 Princess Marvella Liuz

Princess Marvella Liuz, a driven individual, is currently pursuing a major in Information Technology with a focus on Data Analytics. With a keen interest in database programming

and a flair for data analysis, Marvella is driven and motivated by her curiosity to discover insights hidden within vast datasets.

Although Marvella may not boast extensive experience, her determination and ambition drive her desire to overcome challenges and thrive in her career. Eager to gain practical experience, she is actively pursuing ways to quickly secure an internship, with the goal of jumping headfirst into the workforce. Marvella's journey epitomises the spirit of continuous growth and learning, as she seeks to build her own path in the realm of data analytics.

3.1.4 Valerie Kusumo

Valerie is a third-year data analytics student at the University of Technology Sydney with a strong focus on Data Structures and Algorithms, Machine Learning, and Cloud Computing. With a keen interest in uncovering machine learning layers and a background in customer service, her passion for pattern recognition and communication skills with clients prove to be a key strength to the contribution of BOOSTING. Additionally, her interest in statistical data analytics helps in aiding clients with more in-depth information about how some data came to be.

Within the realm of data analytics, Valerie often challenges herself and delves into intricate procedures, utilising data to uncover valuable insights and make informed decisions.

3.1.5 Dominic Jason Gunawan

Dominic Jason Gunawan, a motivated individual, is currently pursuing a degree in Information Technology, majoring in Data Analytics. With a significant interest in machine learning and data preprocessing, Dominic is driven by his curiosity to uncover insights hidden within large datasets and find patterns that can be useful from the datasets.

Although Dominic may not possess a lot of experience, his determination and ambition help him overcome challenges and excel in his career. Eager to gain hands-on experience, he actively seeks opportunities to secure an internship swiftly, aiming to dive headfirst into the workforce. Dominic is eager to see what interesting datasets he will be able to dissect in the future.

3.2 Our Roles

The team's success is guaranteed by our expertise in data analysis, strong team dynamic, commitment to weekly check-ins and updates, and ensuring that all members are responsible in areas they are stronger in.

Romeo will be acting as the team manager. This is because he has the strongest communication skills and experience in group projects and overall project management. As a team manager, he will be the primary point of contact with clients, in charge of developing

plans, leading group meetings, delegating tasks, assisting team members, and ensuring everyone is aligned on project goals. He will also aid in providing feedback and guidance.

Katie is the data analyst and machine learning support of the team. She has experience in data processing, building machine learning algorithms, and preparing data for model training, including cleaning, transformation, and feature selection. She will assist Marvella in training and fine-tuning the model to achieve optimal performance. She will also collaborate with Valerie to evaluate the model's performance and interpret the final results for the client.

Marvella is the lead machine learning engineer. She has strong programming skills and can help the team in data preparation and choosing the appropriate model. She will handle programming the algorithms and working with Katie for model training and implementation.

Valerie will be the data analyst. She has strong knowledge of data structure and algorithms and performs data cleaning, exploration, and visualisation to understand data characteristics and identify patterns. She will collaborate with Katie and Marvella on feature engineering for model training. She will also help to evaluate and analyse model performance.

Dominic will be the data analyst and machine learning engineer. Despite his low experience, Dominic is well versed in machine learning and Python programming, allowing him to contribute greatly to building and optimizing machine learning models. He will collaborate with Marvella and Valrie to handle the programming and optimisation of the various machine-learning models.

4. The Project

4.1 Abstract

This report discusses the implementation of XGBoost and other alternative algorithm to solve the problem of low monitoring and unclear patters of mine trucks fuel consumption. Our findings indicate that mining trucks could take up to 22% of operational costs and increasing. To solve this problem, Boosting has implemented XGBoost along with Random Forest and Neural Networks as the alternative algorithms to predict fuel consumption for mining trucks. Random Search has also been implemented to improve the algorithms by using the most suitable attributes. To evaluate the algorithms, metrics such as Mean Squared Error (MSE) and Mean Absolute Error (MAE) are used along with charts to compare predicted results. This report also includes the difficulties faced during this research project along with the strategies used. Deployment of the machine learning models are also discussed along with other future works to be implemented.

4.2 Introduction

In this project, BOOSTING is planned to explore how various mining trucks consume fuel during the mining operations. Our team is equipped with knowledge and skills in data analytics and machine learning engineering. With the skillset, we are aiming to predict fuel

consumption using the XGBoost algorithm as our main approach. XGBoost is a powerful, efficient and widely used machine learning algorithm. It utilises gradient boosting frameworks for building predictive model (NVIDIA, 2024). It is known to be particularly excelling in handling structured data for classification and regression tasks (NVIDIA, 2024). Beside XGBoost, we are also comparing the results and model's performance with other models like Random Forest and Neural Networks to find the most prediction method.

Before the process of model implementation, we are will run several test to ensure input data is clean and relevant to fuel consumption. These including EDA, data visualisation, pattern recognition and data filtering/cleaning. Design of Experience is also use to understand the attributes behaviour and their relationship with fuel consumption (ASQ, 2023). This is a statistical method will assist us in ensuring the data we are using for the model is the best data. Ultimately, the model will be expected to produce the best result. We are looking into using Ordinal Least Squared (OLS) regression to perform the analysis.

As stated, XGBoost, Random Forest and Neural Network will be used as the predictive machine learning models. We wil evaluate our model based on their accuracy and reliability using statistical measures, such as the mean error index and the R-squared values.

4.3 The Business Problem

Clear management of the fuel consumption used in mine trucks is extremely important due to its extensive operational costs, harsh environmental impacts, resource management, equipment maintenance, etc. According to the Onboard Truck Scale Alternative article, the estimated cost of diesel reaches $1.10/litre (Mine Payload Technologies, 2023). With a 10% overload, which increases the fuel consumption per hour, and other factors such as fleet count and the average operating hours, the price to operate a significant number of mine trucks can reach up to $3,668,500 per annum (Mine Payload Technologies, 2023).

The fuel consumption cost of open-pit mine trucks takes up a significant portion of up to 22 % of the total operational costs (Wang et al, 2021). Unfortunately, existing research on mine truck fuel consumption faces difficulties due to low monitoring accuracy and unclear consumption patterns.

In mining operations, numerous factors must be carefully considered. One of which may not be easily regarded is the rapid depletion of surface-level ore, resulting in increased hauling distances. Consequently, larger trucks will usually be operated, thus leading to larger fuel consumption. According to the Fuel Management Opportunities for Open-pit mining operations article, some statistics indicate that fuel consumption for opencast mining has doubled over the past decade (Tavassoli, 2021). Moreover, the geographical location of the mine presents its own challenges. Mostly, mines are located remotely, requiring more effort to extend resources to these areas. Inevitably, this concerns effective planning of fuel storage and distribution logistics, enveloping more than one sector of the mining operation and reducing overall operational efficiency if not carefully managed.

This project's scope emphasises a deep analysis of fuel consumption patterns in open-pit mine trucks within Australia. As per the examination of the 5179 transportation cycles in three types of mines (Wang et al., 2021), comprehensive research into monitoring patterns and prediction models should be implemented specifically to fuel consumption in mine trucks for optimising fuel consumption in discontinuous or semi-continuous mining to ensure long-term viability, competitiveness as well as environmental concerns.

The geographical remoteness of many mines adds another layer of complexity to the issue. They are usually located in areas that are difficult to access, leading to challenges in fuel storage and distribution logistic (McKinsey & Company, 2020). This put pressure on facilitating effective planning with high level off forecasting and coordination with suppliers to ensure the availability of fuel for operations. Thus, any miscalculation or disruption in the operations will affecting greatly not only on the fuel availability but the overal operational efficiency (Creamer Media Reporter, 2022). Additionally, rapid depletion of surface level ore in many mines have also increase the hauling distances. As a result, it leads to a larger fuel consumption. This can be mitigate by detailed planning and accurate fuel consumption estimation.

Mining trucks typically use diesel because it is the most reliable source for heavy loads. Emissions from these engines produce ground-level ozone, typically due to the interaction between nitrogen oxide emissions and volatile organic compounds (VOCs) emitted by vegetation, particularly under sunlight (Labonline, 2023). Especially on particularly hot days, ozone exposure can make breathing very hard, causing coughing and shortness of breath. Other than that, ground-level ozone poses effects that harm the ecosystem. Its detrimental impacts include hindering photosynthesis and reducing plant growth rates, which can ultimately diminish crop yields and forest productivity. Ozone exposure weakens plant's natural defences, rendering them more susceptible to insect pests and diseases (Arizona Department of Environmental Quality, 2021). Food production could also be negatively impacted by diesel exhaust fumes and ozone, which affects pollination in the natural environment (Labonline, 2023). The odour plumes will significantly change due to the above factors, resulting in pollinating insects incapable of recognising the scent they were previously used to, thus constricting the growth of plants and flowers (BHP, 2024).

As mentioned, mining trucks typically use diesel. Diesel is produced from crude oil, which is extracted from fossil fuels. Statistically speaking, the worldwide consumption of fossil fuels has grown over 1,300-fold (Lorenz, 2023). Needless to say, fossil fuels are finite resources, and they have several steps and geological processes before they are formed into coal, oil, etc. Currently, most of these resources are being consumed faster than replenished. With mining being one of the biggest projects in the world and its excessive consumption of different kinds of resources, optimising fuel usage helps minimise waste and ensures the long-term availability of resources (Lorenz, 2023).

Alternatively, there are some fuels such as biodiesel, hydrogen, and electrice power are emerging as potential solution. These potential alternatives also come with some different

challenges, including infrastructure requirements, further research and testing (Russell et al., 2021).

However, as the solutions provided rely greatly on human, human error may cause further affect on the operations(Editorial Team, 2023). Human are prone to bias and subjectivity when interpreting data, at the same time some error may includes misinterpretation, small error detection, mispelt, miscalculate from the available resources. Comparing to machine learning or AI, with a sufficient data input, it will not only predict the fuel consumption with highest accuracy based on multiple factors, but also assist the operation in long term, reducing various cost and improve efficiency(Editorial Team, 2023).

4.4 The Objective

Open-pit mining fuel consumption has long been known for its increasingly high operational costs and its continuous degradation of environmental aspects. This must be carefully managed to solve its inherent problems. However, accurately forecasting the consumption can be tricky as it involves many factors, such as haul road conditions, truckload, speed, elevation changes, and many more. Our research's objective lies in breaking down these parts and analysing the information from a rich dataset by implementing an advanced predictive model incorporating multi-dimensional features and an improved XGBoost model (a combination of XGBoost and optimisation algorithm). This will allow for more accurate predictions, ultimately improving operational efficiency and cost-effectiveness.

5. Literature Review

5.1 Application of AI (Machine Learning) in the Open-Mining Industry

To increase productivity, safety, and efficiency in the mining process, artificial intelligence has found various applications in the open-pit mining industry. In production planning and scheduling, AI and machine learning have been used to support making tactical decisions for the long- and short-term mining operation plan, such as choosing how to allocate resources and extract ore to meet production and economic goals.

One of the earliest efforts was a non-linear programming and genetic research application for production scheduling in coal mines, in which they used Genetic Algorithms (GA) to determine schedules for production, transportation, ore blending, and market selection for several coal mines (Pendharkar et al., 2000). As years passed, more and more machine learning algorithms were implemented into the open-mining industry. In the research of optimising ore-waste discrimination and block sequencing through simulated annealing, Simulated Annealing (SA) was used to perform multiple orebody simulations to evaluate whether a block should be classified as ore or waste (Kumaral, 2013). Production forecasting also makes up an important part of production planning and scheduling. One of the research on Natural Resources Research called Estimating Ore Production in Open-pit Mines Using Various Machine Learning Algorithms Based on a Truck-Haulage System and Support of IoT proposed the implementation of Internet of Things (IoT) devices to gather data, and

supervised machine learning is then used to predict ore production, in which SVM achieved the best performance (Choi et al., 2020).

Another important area of focus for research is grade control and ore delineation. The research on the automated lithological classification using UAV and machine learning on open-cast mines implemented the use of tree-based algorithms (T-B), SVM, and K-Nearest Neighbours (KNN) to divide images of a mining bench into garbage, ore, vegetation, and soil sections (Beretta et al., 2019). Research on the automation creation of mining polygons provided a hierarchical clustering approach to classify mineral blocks into larger units according to similarity in grade and type of rock; a shape control method was then applied to correct for technically viable shapes (Tablesh et al., 2013)

For mining operations to achieve their financial and production goals, equipment allocation and control must be done effectively. Key decisions in the operational planning of mining activities include distributing and sizing truck fleets to shovels and shovels to available mining faces. Data-driven approaches, such as Discrete Event Simulations (DES), have been widely used to evaluate various strategies, and metaheuristics, such as Genetic Algorithms (GA), have been popular in generating equipment allocation and routing plans (Noriega et al., 2022).

As AI technology continues to evolve, we can expect more innovative solutions that will transform the mining industry for better economic viability and environmental sustainability.

5.2 Application of XGBoost Algorithm in Predicting Fuel Consumption

XGBoost (eXtreme Gradient Boosting) is a powerful algorithm that has gained traction in the industry. Like its name, XGBoost was created by Tianqi Chen to use a boosted tree algorithm to their maximum computational capability (Brownlee, 2021). As mentioned in section 3.2. Application of AI in the Open Mining Industry, there are a variety of machine learning models used in research which have shown exceptional performance, including SVM, GA, DES, and T-B. However, for this project, our objective is to produce the highest possible accuracy in which employing an ensemble method like XGBoost will be beneficial due to its effectiveness in handling multi-dimensional data.

According to the paper “Open-pit mine truck fuel consumption pattern and application based on multi-dimensional features and XGBoost”, researchers have properly predicted the fuel and energy consumption of mining trucks using the XGBoost model and a data-driven approach that leverages various data sources, including sensors, GPS tracking, operational data (haulage distance, lifting height, operation time), truck specifications, and environmental factors. Once the model is trained with the data, it learns the relationships between these factors and fuel consumption. It predicts it for new scenarios based on operational parameters and truck specifications. As the research demonstrated, the model achieves an R-squared value of 0.94 and a Root mean square error of 2.23 for its performance (Wang et al, 2021). This indicates a strong correlation between predicted and actual consumption, proving XGBoost's capability to handle the task. Additionally, XGBoost has higher performance compared to other models in similar domains. In a study of energy consumption forecasting

and traffic flow analysis, XGBoost was compared to ANN, MLR, and LightGBM and has the highest R^2, lowest MAE and RMSE (Ullah et al, 2017).

Beyond predicting fuel consumption, the functionality of this model can be integrated into a bigger system where the estimation of fuel consumption can be immediately identified and assists in facilitating adjustments to optimise fuel consumption in real-time. Besides real-time estimation, the company can use the model's predictions to assign trucks and plan routes that minimise fuel usage. As a result, it will reduce costs and improve efficiency. This will also benefit the environment as minimising fuel use contributes to a smaller carbon footprint. From there, mining operations will be more sustainable.

On top of that, this machine learning has the potential to extend beyond this single application. There are various exploration opportunities, such as replicas for mining equipment using sensor data and historical maintenance records to predict and plan maintenance schedules. In addition, AI can enable autonomous trucks to navigate the complex mining environment safely and efficiently by reducing human intervention and associated risks.

6. Scope

As previously mentioned, The fuel consumption cost of open-pit mine trucks takes up a significant portion of up to 22 % of the total operational costs, as mentioned by Wang et al. (2021). This presents a significant business challenge as it is resource-intensive and causes damage to the environment surrounding the area.

Our research's primary goal is to examine and gain insights into various ideas, approaches, and best practices for fuel efficiency optimisation during open-pit mining activities in Australia. For this to be achieved, it requires serious commitment from each team member, ensuring their dedication to implementing efficient practices in the project.

The meeting sessions aim to empower team members to monitor and enhance the fuel consumption performance of the mining trucks by active engagement in material review, structuring an effective machine learning model as well as effective communication to ensure the research's success. Corrections and improvements can be made based on the insights gained during these sessions, ultimately ensuring the quality of fuel consumption outcomes. Our choice of machine learning would be implementing multi-dimensional features and XGBoost algorithms.

Assumptions and constraints play an important role in this project. Team members must be aware of the project's objectives, guidelines, and the scope of fuel consumption reduction efforts. Dedication, cooperation, and enthusiasm are expected from each team member to achieve high-quality fuel efficiency outcomes while adhering to project timelines.

Nevertheless, time constraints arise due to the diverse responsibilities of team members, potentially impacting the project implementation. The diversity in trucks, along with

variations in terrain and operational conditions, may interfere with the uniform performance of fuel-efficient practices. Challenges related to the reliability of fuel monitoring systems may interfere with data quality. Additionally, external factors like fuel quality and availability can influence the overall success of fuel consumption optimisation initiatives. Therefore, meticulous planning is essential to overcome these challenges and effectively reduce fuel consumption in open-pit mining trucks in Australia.

7. Research Methodology

7.1 Background

7.1.1 Introduction to XGBoost

The XGBoost algorithm, also known as eXtreme Gradient Boosting, is an open-source implementation and highly optimised framework of the gradient-boosted trees algorithm. It is a supervised machine learning algorithm that combines estimates from various weaker models to accurately predict a target variable (Amazon, 2024). XGBoost is usually used due to its execution speed and model performance. It works by adding weaker models to existing ones, with the former correcting the errors made by the latter, using gradient descent optimisation to minimise a predefined loss function during training.

7.1.2 Random Search

The team has chosen random search as the optimisation algorithm to improve the XGBoost along with other machine learning models. In the context of machine learning, random search is a type of hyperparameter search technique that uses random sample points to determine the optimal solution. This can help solve problems where there is a lot of domain knowledge that could skew or affect the optimisation technique, leading to the discovery of non-intuitive solutions (Gupta, 2023). Random search is easy to use and can be quite beneficial in high-dimensional and complex hyperparameter spaces. Its effectiveness is contingent upon the nature of the problem and the search space, and it does not ensure the discovery of the best solution.

7.1.3 XGBoost and Random Search

Random search is one of the most popular tuning algorithms for XGBoost, due to the fact that it's fast and effective when it comes to a large search space (Emmanuel, 2023). There are a couple of parameters that could be tuned such as the following:

1. **Max_depth**: The maximum depth of a tree is specified by this option. While lowering max_depth can result in underfitting, increasing it can make the model more complex and cause overfitting.
2. **Learning_rate**: During training, this parameter regulates the step size used to update the weights. Slower learning is achieved using a smaller learning_rate, which can also assist in avoiding overfitting. On the other hand, a learning rate that is too low may

cause a training time that is longer and slower to converge. A higher learning_rate may cause overfitting in addition to accelerating convergence.

3. **N_estimators**: The number of trees in the model is specified by this parameter. Although increasing n_estimators usually enhances the model's performance, it also lengthens the training period and uses more memory.
4. **Subsample**: The percentage of observations for each tree that will be randomly sampled is specified by this option. The model's capacity to generalise to new data may be enhanced by increasing the subsample, but doing so may also cause overfitting. Although it can prevent overfitting, decreasing the subsample can also cause underfitting.
5. **Colsample_bytree**: The percentage of features for each tree that will be randomly sampled is specified by this option. The model's capacity to generalise to new data may be enhanced by increasing colsample_bytree, but doing so may cause overfitting. Although it can prevent overfitting, lowering colsample_bytree can also cause underfitting.

7.2 Research Timeline

The timeline for this research is summarised by *Figure 7.2.* The project starts with the collection of the dataset along with our data exploration and initial analysis. Design of Experiment is then conducted to find the correlations between attributes. Data preprocessing comes next after finding out the optimal attributes to include. The next stage is training the XGBoost model and the other alternative algorithms. The models are then evaluated and re-trained if results are poor. After finding the most accurate predictive models, the chosen optimisation algorithm, random search, is then applied to optimise the models. Models are then re-trained again to get the most optimised predictive models which are then the final models we present as the solution.

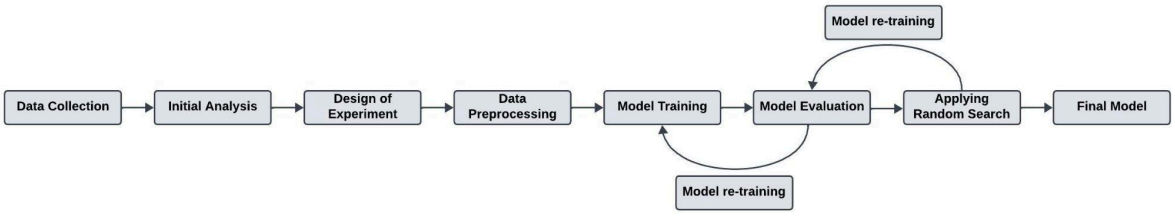


Figure 7.2 Research Timeline.

8. Data Exploration

The data for this project was obtained from a reliable third-party source with access to the Western Sydney Mining dataset, an Australian-owned data platform. The initial datasheet is divided into three smaller sheets to explore the dataset more efficiently: CycleData, LocationData and DelayData.

- Cycle data consists of the movement information of the equipment, the duration of the cycles and mining operation-related attributes
- Location data consists of the location information of the equipment
- Delay data consists of the delay in service time of the on-ground equipment for planned and unplanned delay schedules and other important attributes related to the delay.

The CycleData contains 93 attributes, with 47346 entries in each attribute related closely to mining truck operations on site. With the detailed composition of the data, it will be the primary dataset for model training and testing of this project. However, as the objective is to predict the fuel consumption of open-pit mining trucks, only a few attributes should be considered. Fuel consumption can be easily influenced by the truck machine features, the load and capacity-related features (payload, cycle count, and machine capacity) and related environmental and operational conditions. From these key factors, the best-suited attributes for this project are determined:

1. Machine Features: the attributes focus on the machine/truck features and fuel used when operated.

Attributes	**Description**	**Data Type**
Autonomous	If the vehicle is autonomous or not	Text
Fuel Used	Actual fuel used	Float
IC	Integrated Circuit count for vehicle **If "IC" relates to components in autonomous or semi-autonomous vehicles, these systems could impact fuel efficiency. Autonomous vehicles are often optimised for efficient driving, which could indirectly influence fuel consumption. The function of this is not clearly stated anywhere**	Text
TMPH	Truck speed measured in mph	Float
Primary Machine Name Secondary Machine Name	Name of the primary/secondary machine	Text

Primary Machine Category Name Secondary Machine Category Name	Category of the machine	Text

Table 8.1 Machine Attributes.

2. Load and capacity-related features: the attributes showcase the payload, loading and duration of the process in a cycle.

Attributes	**Description**	**Data Type**
Payload (t)	Payload in tons	Text
Completed Cycle Count	A flag which tells us if the cycle is completed or not	Integer
Cycle Duration	Duration of the cycle in seconds	Float
Cycle Type	Type of cycle: this dataset has 3 types - TruckCycle, LoaderCycle, AuxMobileCycle	Text
Full Travel Duration	Total travel duration	Text
Idle Duration	Idle duration for vehicle or equipment	Integer
Loading Count	Number of counts of loading truck	Text (NULL for AuxMobileCycle)

Table 8.2 Load and Capacity Attributes.

3. Environmental and Operational Conditions: these attributes include many factors, including the location of operation and process of operations, that have a potential effect on fuel consumption.

Attributes	**Description**	**Data Type**
Delay Time	Delay time wasted in	Integer

	waiting	
Dumping Duration	Time required by truck to empty	Float
Empty EFH Distance	Distance truck is travelling empty	Float
Empty Slope Distance	Distance travelled at an inclination	Float
Empty Travel Duration	The actual travel duration for an empty truck	Float
Queuing Duration	Total queue duration	Float
OPERATINGTIME (CAT)	operating time	Integer
Travelling Empty Duration	duration when the truck is travelling empty	Float
Travelling Full Duration	duration when the truck is travelling full	Float
Destination Location Name	Name of destination location	Text
Destination Location Description	Description of Destination location	Text
AT Available Time (iMine)	Availability time in seconds for iMine	Float

Table 8.3 Environmental and Operational Attributes.

Besides the listed attributes, LocationData and DelayData can further explore and give a clearer view of this factor.

The LocationData consists of 4 attributes and 31 rows of entries. This data gives a clearer view of the geographic location where the machines/trucks operate. This data includes:

Attributes	**Description**	**Data Type**

Location_Id	This is the Location ID of the machine	Text
Name	This is the unique ID of the machine used to identify it	Text
Latitude	The latitude of the location of the machine	Float
Longitude	The longitude of the location of the machine	Float

Table 8.4 Location Attributes.

The DelayData provides some insights into all delays that happened during the operation. This includes:

Attributes	**Description**	**Data Type**
Delay OID	Unique ID assigned to each delay	Text
Description	Describes what the delay was for	Text
ECF Class ID	The class ID of the delay	Text
Delay Class Description	Description of the delay class for the delay	Text
Engine Stopped Flag	This flag tells if the engine was stopped in this delay or not	Text
Target Location	The target location of the delay	Numeral
Target Machine Name	Target machine ID/name in which the delay happened	Text
Delay Start Timestamp	The start time of the delay	Time format (text)

(GMT8)		
Delay Finish Timestamp (GMT8)	The time when the delay was resolved	Time format (text)

Table 8.5 Delay Attributes.

After importing all necessary attributes into an Excel sheet and Tableau Workbook, exploratory data analysis (EDA) was conducted to identify trends, outliers, anomalies and correlation analysis to understand the relationships between different attributes, especially those likely to influence fuel consumption.

At first glance, when the datasets are uploaded onto the platforms, the data type is mismatched with its actual value, such as whether fuel consumption should be float or continuous data. To fix this, the datatype of each attribute was manually changed.

From the initial analysis in the Excel sheet, there are a lot of NULL variables. This may be due to data not being recorded correctly, errors in data collection or transmission, or equipment used for collecting data not working as expected. NULL values can greatly affect statistical analysis due to their influence on calculation. Close examination of the NULL data pattern has shown that ‘Job Type’- 98.31%, ‘Source Location Description’- 87.76%, ‘Source Queuing Start Timestamp’- 66.12%, ‘Destination Dumping Start Timestamp’-54.40%, ‘ TRUCKQUEUEATSOURCELOCATION’-53.55%, ‘Loading Efficiency’- 53%, ‘ASSOCPAYLOADNOMINAL’- 53% are the attributes with the highest percentage of the number of NULL values in the dataset. Due to the high volume of NULL in these attributes, resulting in low reliability, they will not be used for further analysis and model training. Besides recognising high NULL percentage columns, NULL occurrence patterns are also noted. The columns below are those that have the same percentage of NULL values, which is 52.29%

- Empty Slope Distance
- Empty Fall Height
- Fuel Used
- Queuing Duration
- TMPH
- Empty Travel Duration
- Empty Target Travel Duration
- Empty Slope Length
- Queuing at Source Duration
- Empty Rise Height
- Empty Plan Length
- Empty EFH Length
- Empty Expected Travel Duration
- Full Travel Duration
- Empty EFH Distance
- Queuing at Sink Duration
- Dumping SMU Duration
- Dumping Duration
- QUEUEATSINKDURATION
- WAITFORDUMPDURATION
- WAITFORLOADDURATION
- iMine Load FCTR Truck
- Full Expected Travel Duration

The rows with NULL values co-occur similarly across all of these columns. This finding suggests a strong correlation in the occurrence of null values, which implies they are likely

interconnected or dependent on similar conditions. In this case, the NULL value will be imputed or removed in small quantities if further analysis can tolerate some degree of missingness.

Looking at the ‘Fuel Used’ and ‘CycleType’ columns specifically, when the ‘Fuel Use’ is null, there are no instances of ‘TruckCycle’ type. **This strongly suggests that the ‘Fuel Used’ data is only recorded for ‘TruckCycle’.**

The correlation heat map below provides further insight into how these variables are interrelated with fuel usage:

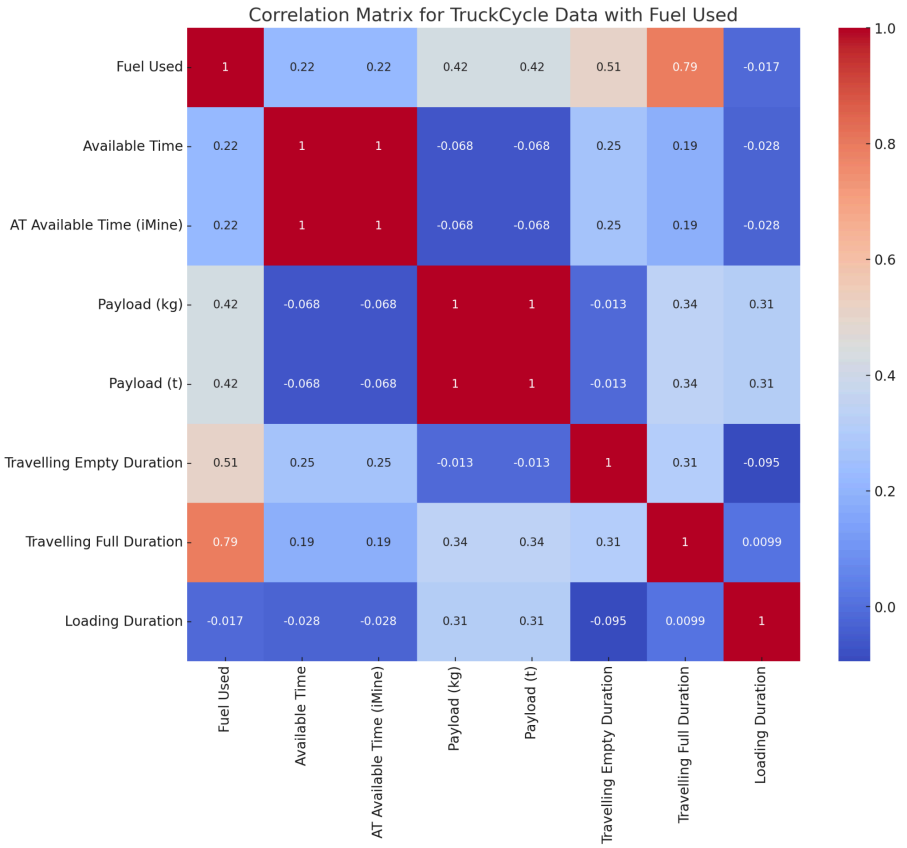


Figure 8.0. Correlation heatmap.

Payload shows a strong correlation (0.42) with 'Fuel Used', suggesting that the weight carried by the trucks significantly impacts fuel consumption.

Travelling Durations (Empty and Full) indicate when trucks travel either empty or with a load. The strong correlation indicates that the amount of time spent in transit, whether full or empty, affects fuel usage.

Available Time and AT Available Time (iMine) reflect the overall time the trucks are operational or available for operation. Strong correlations could suggest that the sheer operational time, regardless of specific activities, is a major determinant of fuel usage.

From this information, further data exploration to provide a clearer understanding and insight of the dataset.

9. Initial Findings

This section provides the initial findings derived from thoroughly examining three different datasets: CycleData, DelayData, and LocationData. The data has been carefully retrieved, processed, and examined these datasets using Microsoft PowerBI's robust analytical tools to gain valuable insights that will assist us in building our machine-learning models. The team has decided to use Microsoft PowerBI since it allows us to build interactive dashboards that allow the team to perform a more effective and accurate data analysis.

The three datasets are imported into PowerBI, and some columns are converted into the desired data type to allow calculations to be made.

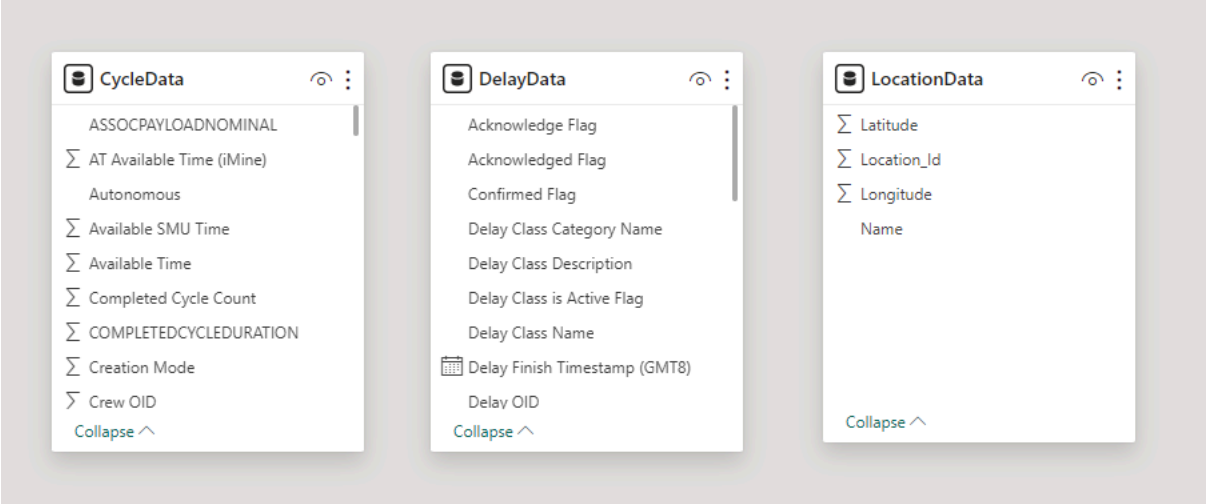


Figure 9.1 Datasets in PowerBI.

9.1 Cycle Data

The cycle dataset is the largest dataset among the three. It comprises numerous columns however, since only truck data is needed, filters have been set for the cycle dashboards in PowerBI, as seen below. Cycle Type is set to "TruckCycle", and null values are removed for the Truck Name.

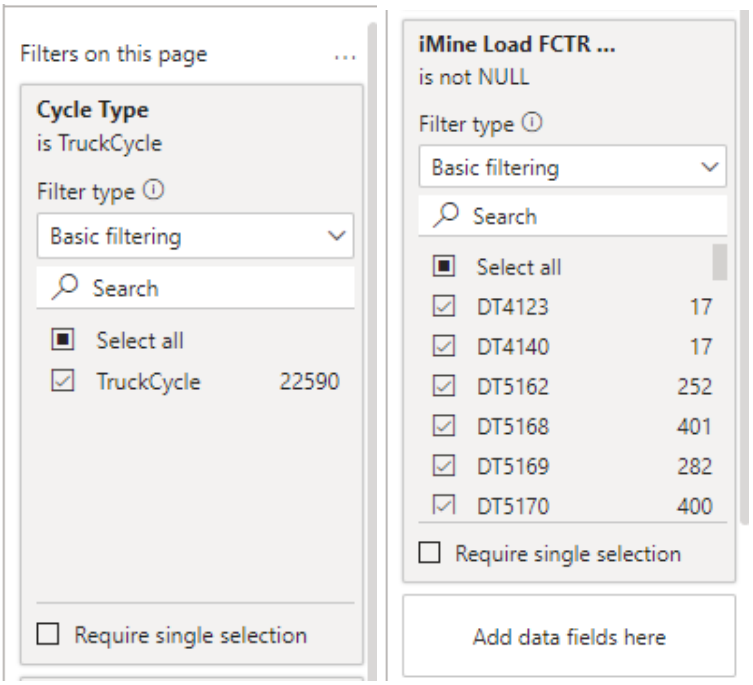


Figure 9.1.1 Filters for Cycle Data.

The first analysis is done on the 4 main attributes, which are chosen due to their direct effect on the fuel consumption of mining trucks. Average, median and standard deviation are calculated and collected in the table below:

	Fuel Used	TMPH	Payload (t)	Cycle Duration
Average	0.052	73784.62	228.34	1785.77
Median	0.054	74361.04	238.20	1489.00
Standard Deviation	0.025	27694.01	49.94	2753.63

Table 9.1 Average, Median, and Standard Deviation Results.

→ From the table, it is shown that

- Higher speed may have an effect in higher fuel consumption
- Fuel consumption stays around 0.054
- Payloads weigh around 220 tonnes
- A Cycle usually lasts more than 1500

The second analysis has been done on the truck name attribute to see which trucks are used most in this dataset. As seen below, the top three most used trucks are “DT5236”, “DT5275”, and “DT5265”, respectively. We can also see the least used trucks are “DT4123” and “DT4140”.

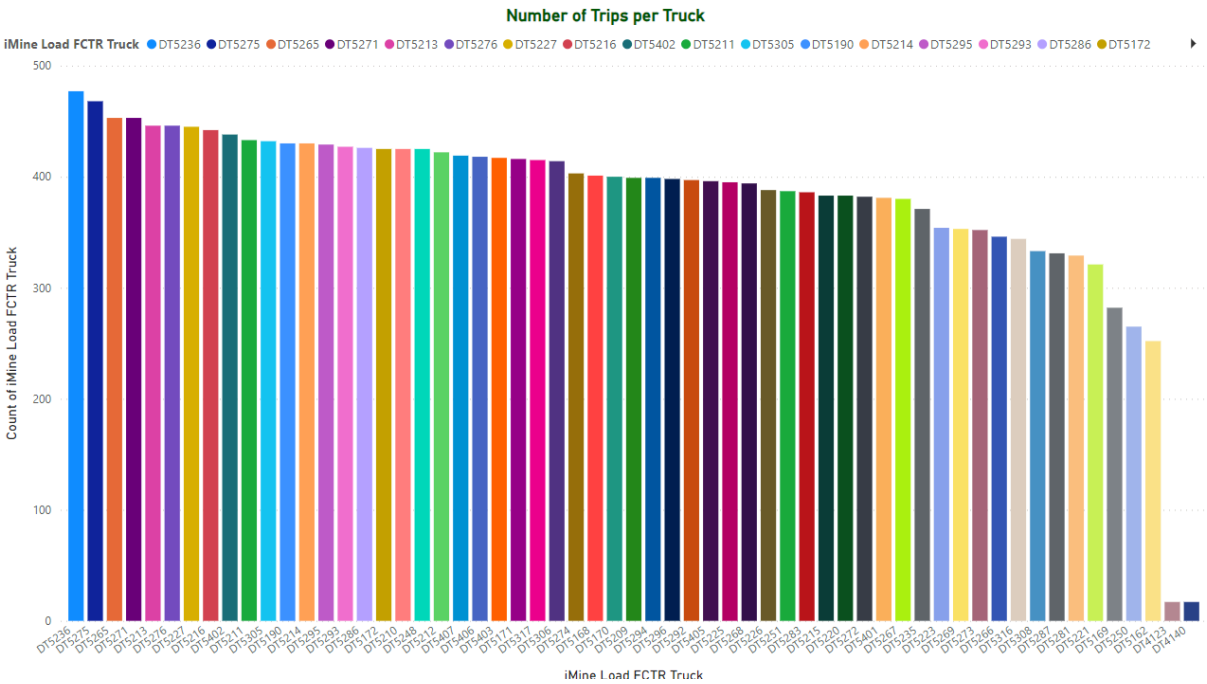


Figure 9.1.2 Count of Trips by Truck Name Bar Chart.

The next analysis is on the truck attributes per trip: the number of trips, cycle duration, fuel used, truck speed (MpH), and truck payload (Tonnes). An average is taken for all these attributes, and an interactive filter is applied to this dashboard. When the filter is applied, the numbers below will change automatically based on the truck name.

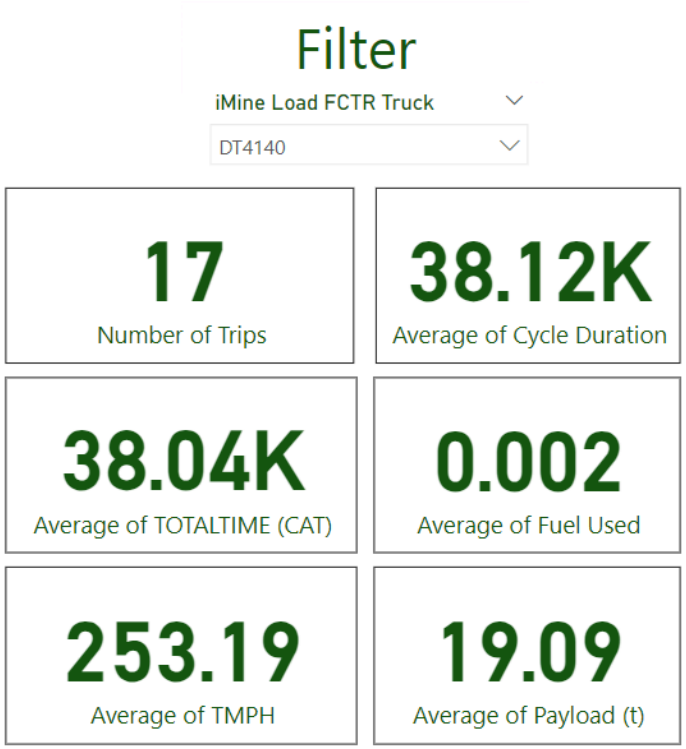


Figure 9.1.3 Interactive Dashboard for Truck Table.

The interactive dashboard above is converted into a table to show all the data for each truck name in a listed format. From the table, the following insights are found,

- DT5250 has the highest average cycle duration, while DT5275 has the lowest average.
- DT5269 has the fastest speed of 83828 MpH on average, while DT4123 has null values for the attribute.
- DT5227, on average, carries the heaviest load of 233.28 tonnes, while DT4123 only carries 17.47 on average.
- DT5269 uses the highest fuel on average (0.071), while DT4140 uses the lowest (0.002).

iMine Load FCTR Truck	Average of Cycle Duration	Average of TMPH	Average of Payload (t)	Average of Fuel Used	Number of Trips
DT4123	38117.65	0.00	17.47	0.017	17
DT4140	38117.65	253.19	19.09	0.002	17
DT5162	2737.35	70832.74	224.53	0.053	252
DT5168	1718.63	72419.62	228.08	0.048	401
DT5169	2370.70	72557.95	222.53	0.053	282
DT5170	1717.85	69983.42	230.01	0.050	400
DT5171	1657.75	79559.24	229.48	0.058	416
DT5172	1620.75	76650.96	229.40	0.052	425
DT5190	1602.05	76869.24	228.58	0.054	430
DT5209	1728.24	71524.86	229.87	0.051	399
DT5210	1620.69	74229.03	227.23	0.049	425
DT5211	1590.56	65187.92	229.77	0.044	433
DT5212	1631.73	67204.94	229.18	0.052	422
DT5213	1545.20	77532.07	232.06	0.051	446
DT5214	1603.11	77820.05	228.87	0.053	430
DT5215	1799.07	70613.87	226.09	0.046	383
DT5216	1561.63	72156.51	227.91	0.049	442
DT5220	1799.90	77593.52	227.96	0.059	383
DT5221	2132.07	78655.35	228.57	0.069	321
DT5223	1947.96	67378.48	228.29	0.049	354
DT5225	1693.62	75286.87	230.91	0.047	395
DT5226	1725.54	75185.91	227.33	0.054	388
DT5227	1548.77	69524.96	233.28	0.049	445
DT5235	1857.47	66719.87	227.06	0.051	371
DT5236	1437.63	69294.42	229.01	0.045	477
DT5248	1620.61	73056.73	225.97	0.048	425
DT5250	2524.09	75414.03	225.95	0.058	265
DT5251	1783.80	72390.00	228.55	0.060	387
DT5265	1521.20	73046.67	229.86	0.050	453
DT5266	1993.87	72080.09	226.49	0.054	346
DT5267	1815.69	76356.17	231.38	0.055	380
DT5268	1748.95	74138.61	225.95	0.053	394
DT5269	1951.23	83828.00	228.51	0.071	353
DT5271	1523.21	69432.37	228.85	0.048	453
DT5272	1807.27	75140.29	226.75	0.049	382
DT5273	1899.30	80406.42	227.35	0.061	352
DT5274	1669.27	77787.34	231.10	0.056	403
DT5275	1432.06	69836.92	229.39	0.048	468
DT5276	1547.75	69316.83	231.41	0.048	446
DT5281	2096.37	73949.19	223.53	0.049	329
DT5283	1788.71	75401.80	229.18	0.052	386
DT5286	1616.65	68063.41	228.85	0.048	426
DT5287	2082.79	79430.13	229.31	0.060	331
DT5292	1730.75	81894.55	227.77	0.056	397
DT5293	1609.24	70956.90	232.19	0.050	427
DT5294	1730.43	82738.30	226.23	0.057	399

DT5295	1609.29	76919.43	227.96	0.054	429
DT5296	1735.49	68272.69	229.55	0.049	398
DT5305	1596.57	79762.62	229.51	0.055	432
DT5306	1667.63	73944.56	227.23	0.052	414
DT5308	1945.95	82031.12	226.54	0.065	333
DT5316	1883.72	68013.74	227.61	0.047	344
DT5317	1654.60	74430.66	230.99	0.055	415
DT5401	1760.53	74987.32	228.17	0.058	381
DT5402	1574.99	74866.96	227.83	0.050	438
DT5403	1653.94	74716.59	228.98	0.051	417
DT5405	1744.22	71912.14	232.50	0.051	396
DT5406	1649.17	76556.97	229.03	0.052	418
DT5407	1644.57	71371.02	229.09	0.051	419
Total	**1785.77**	**73784.62**	**228.34**	**0.052**	**22590**

Figure 9.1.4 Truck Name Table.

9.2 Delay Data

The delay dataset contains every single delay that occurred during the time period. This dataset contains more text-based data, such as the description of the delay, the description of the machine class, and the person recording the delay.

The first analysis performed on the dataset is on the delay count by machine class name. Null values are removed to provide a clear bar chart. From the bar chart, it can be seen that “CAT 793F CMD” has the highest delay count (5646), far ahead of the next machine “LBH R9400” (1424).

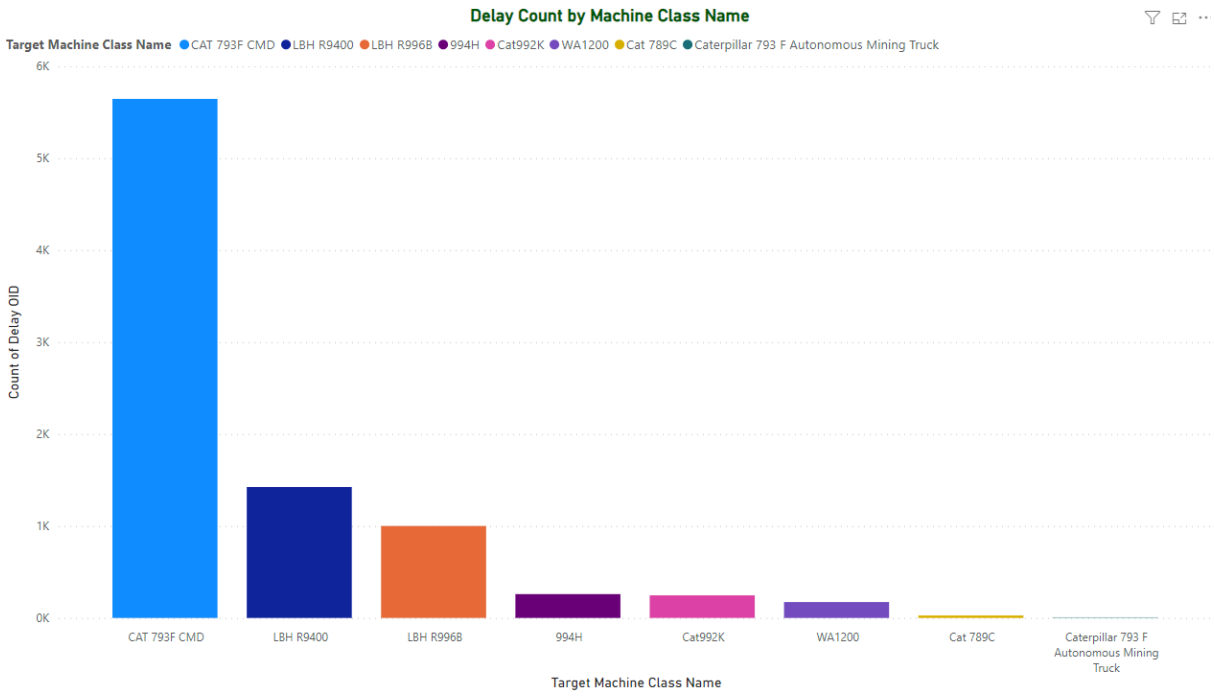


Figure 9.2.1 Delay Count by Machine Class Name.

After thorough analysis, it appears that none of the delays that occurred were by primary trucks used, as seen in cycle data. These delays occurred due to machinery that assists these trucks on every trip, which are labelled “Primary Machine Name” and “Secondary Machine Name” in the cycle dataset.

This may mean that the delay dataset is unnecessary and can be removed in data pre-processing since it will not affect predicting fuel usage for mine trucks.

9.3 Location Data

The location dataset only contains four columns which are Location_Id, Name, Latitude, and Longitude. Using the azure map visual in PowerBI, the latitude and longitude of each row of data to pinpoint the location of these coordinates. As seen below, the coordinates pinpoint the mines, which are part of the Jimblebar Hub located in the Pilbara region in Western Australia. It is fully owned and controlled by BHP, and it is one of five mines and four processing hubs that make up Western Australia Iron Ore (BHP, 2024).

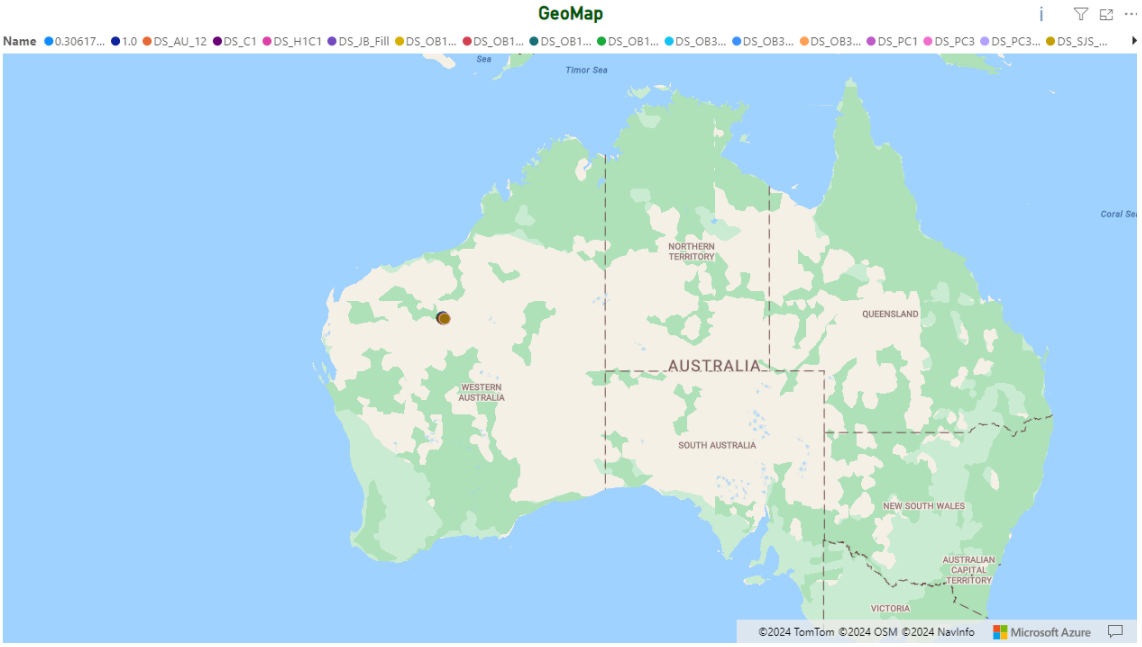


Figure 9.3.1 GeoMaps of Truck/Machineries Location Wide-Angle.

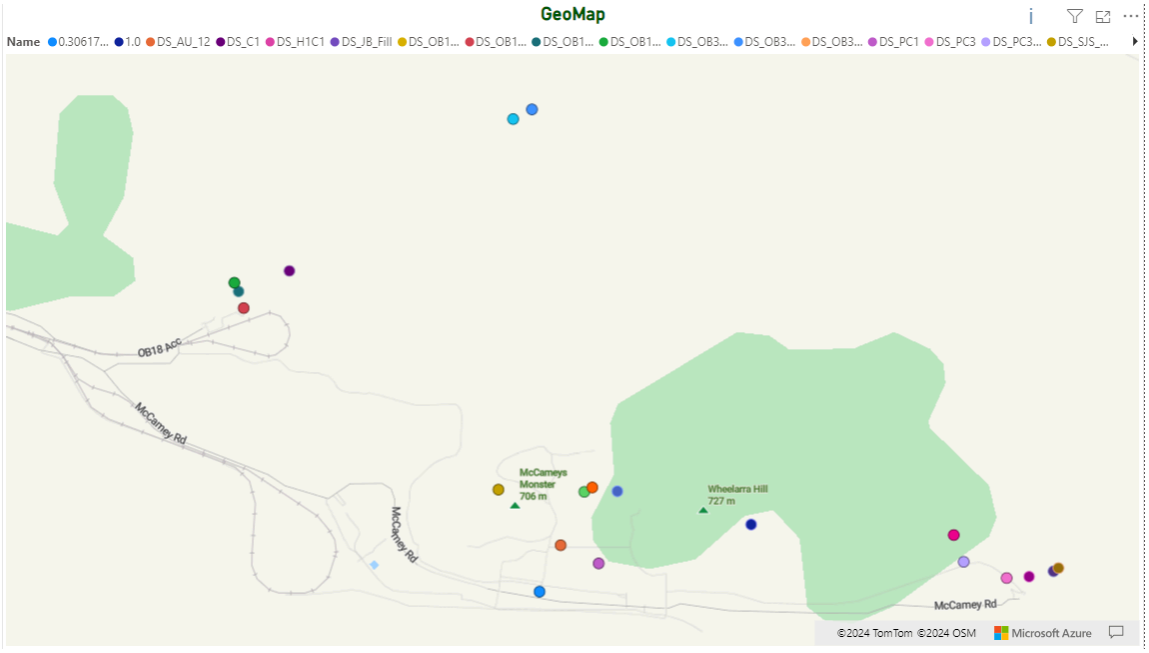


Figure 9.3.2 GeoMaps of Truck/Machineries Location Zoomed-In.

After thorough exploration, the names of these locations correspond to “Source Location Name” and “Destination Location Name” in the cycle dataset. This means that the names of these locations correspond to the source and destination these mine trucks are coming from and going to, respectively. These locations could be potentially used as inputs for our machine-learning models.

10. Design of Experiment

Design of Experience, also known as DOE, is a statiscal method used for planning, conducting, analysing and interpreting controlled tests to evaluate the factors that control the value of a parameter or group of parameter (ASQ, 2023). In this project, DOE will be used to understand the influence and relationship of various operational factors on fuel consumption of mining trucks. The process of DOE will be conducted through regression analysis.

Before starting the analysis, the dataset is filtered based on the EDA and initial findings process earlier. This ensures the dataset only contains key variables that potentially affect fuel consumption and minimise inaccurate outputs. Firstly, attributes with more than 60% of NULL value occurrence are removed as high percentage of NULL value will influence the statistical analysis greatly. ‘60%’ is used as a safe estimate number in this case due to a group of attributes with the same number of NULL values. After that, rows where the CycleType is not ‘TruckCycle’ are removed, based on the pattern found during EDA process. From the initial findings, unrelated attributes such as Job Type, Job Description, and so on will be filtered out. Since there are 90 columns, to quickly filter the data, key variables are picked and included in a new datasheet. These variables are:

- **Fuel Used:** The amount of fuel used, our primary target for prediction.
- **COMPLETEDCYCLEDURATION**: The duration of a completed cycle, which could influence fuel consumption.
- **Payload (kg or t):** The weight of the payload carried, impacting fuel efficiency.
- **Autonomous**: Indicates whether the truck operation was autonomous.
- **Available SMU Time:** Service Meter Unit time when the truck was available.
- **iMine Operating Hours**: Total operating hours as recorded by the iMine system.
- **AT Available Time (iMine)**: Available time as recorded by the iMine system, which might reflect overall operational efficiency.
- **Travelling Empty Duration:** The time spent traveling without a load, which affects fuel consumption differently compared to loaded travel.
- **Travelling Full Duration:** The time spent traveling with a full load, directly relevant to understanding how load impacts fuel usage.
- **Idle Duration:** Time when the truck is running but not actively engaged in primary transport or loading activities, still consuming fuel.
- **Primary Machine Class Name:** Truck Model Name

As this is a statistical analysis, one-hot encoding is used to transform the ‘Primary Machine Class Name’ categorical data into numerical format. The process is shown in the *Figure 10.1.*

```
# Calculate the percentage of NULL values in each column
null_percentage = data.isnull().mean()

# Filter out columns where the null percentage is greater than 60%
columns_to_keep = null_percentage[null_percentage < 0.6].index.tolist()

# Keep only the columns that have less than 60% NULL values
data = data[columns_to_keep]

# Filter to include only rows where the cycle type is 'TruckCycle'
# Ensure that the 'Cycle Type' column was not removed due to NULL values before filtering rows
if 'Cycle Type' in data.columns:
    data = data[data['Cycle Type'] == 'TruckCycle']
else:
    print("Filtered dataset does not include 'Cycle Type' due to high null values or it does not exist.")

# Save the filtered data to a new CSV file
data.to_csv('FilteredData_TruckCycle.csv', index=False)
```

Figure 10.1 Data Filtering process.

```
# Initial filtered attributes focusing on key aspects for fuel consumption prediction
filtered_data = data[['Fuel Used', 'COMPLETEDCYCLEDURATION', 'Payload (kg)', 'Autonomous',
                      'Available SMU Time', 'iMine Operating Hours', 'AT Available Time (iMine)',
                      'Travelling Empty Duration', 'Travelling Full Duration', 'Idle Duration', 'Primary Machine Class Name']]

# One-hot encode the truck models
filtered_data = pd.get_dummies(filtered_data, columns=['Primary Machine Class Name'])
```

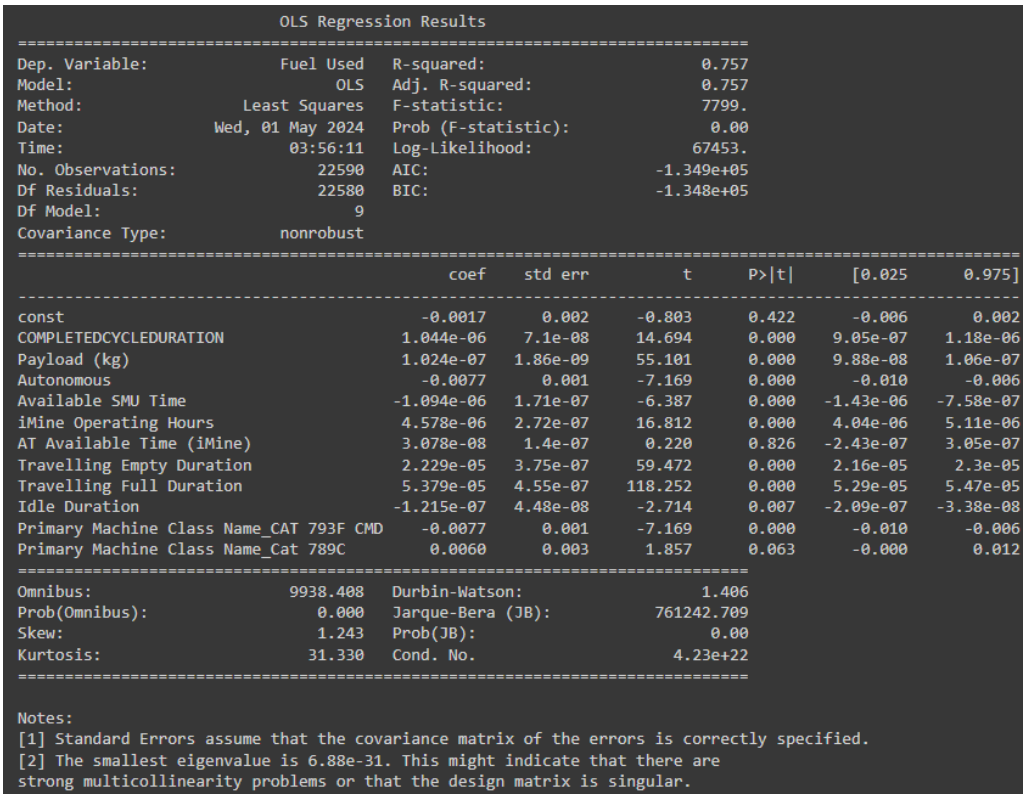
Figure 10.2. Filtered Data.

After filtering the data, regression analysis is performed, specifically, Ordinary Least Squares regression (OLS). OLS is a common and straightforward technique for estimating the parameters in a linear regression model. It provides clear interpretation of the relationship between independent and dependent variables (XLSTAT, 2023). The code below provides the structure, data preparation for model implementation.

```
import pandas as pd
import statsmodels.api as sm
import numpy as np
# Load the dataset
file_path = '/content/FilteredData_TruckCycle.csv'
data = pd.read_csv(file_path)

# One-hot encode the truck models
data = pd.get_dummies(data, columns=['Primary Machine Class Name'])

# Convert all columns to float, coercing errors and filling NaNs
for column in data.columns:
    data[column] = pd.to_numeric(data[column], errors='coerce')
data.fillna(data.mean(), inplace=True)

# Convert boolean columns to integer to avoid any Statsmodels issues
data['Primary Machine Class Name_CAT 793F CMD'] = data['Primary Machine Class Name_CAT 793F CMD'].astype(int)
data['Primary Machine Class Name_Cat 789C'] = data['Primary Machine Class Name_Cat 789C'].astype(int)

# Ensure no infinite values exist
data.replace([np.inf, -np.inf], np.nan, inplace=True)
data.fillna(data.mean(), inplace=True)

# Define predictors and add a constant for intercept
predictors = ['COMPLETEDCYCLEDURATION', 'Payload (kg)', 'Autonomous', 'Available SMU Time',
              'iMine Operating Hours', 'AT Available Time (iMine)', 'Travelling Empty Duration',
              'Travelling Full Duration', 'Idle Duration',
              'Primary Machine Class Name_CAT 793F CMD', 'Primary Machine Class Name_Cat 789C']
X = sm.add_constant(data[predictors])  # Adding a constant for the intercept

# Dependent variable
y = data['Fuel Used']

# Fit the regression model
model = sm.OLS(y, X).fit()

# Print out the summary of the regression model
print(model.summary())
```

Figure 10.3. OLS regression model implementation.

After running the regression model successfully, the R-squared value is shown to be 0.757, which means 75.5% of the variability in fuel consumption is explained by the model. COMPLETECYCLEDURATION and Payload have significant positive coefficients, indicating that as cycle duration and payload increases, so does fuel consumption. As expected, travelling (empty and full) duration have a significant positive effects on fuel consumption. Unexpectedly, while SMU time has a negative effect, operation hours have a positive effect. This may due to the value of SMU does not match its description, led to misinterpretation. Autonomous also has a negative coefficient, suggesting autonomous truck consumes less fuel. Idle duration has a very little effect on fuel consumption based on its low negative coefficient. The full result can be seen in the *Figure 10.4.*



OLS Regression Results

Dep. Variable:	Fuel Used	R-squared:	0.757
Model:	OLS	Adj. R-squared:	0.757
Method:	Least Squares	F-statistic:	7799.
Date:	Wed, 01 May 2024	Prob (F-statistic):	0.00
Time:	03:56:11	Log-Likelihood:	67453.
No. Observations:	22590	AIC:	-1.349e+05
Df Residuals:	22580	BIC:	-1.348e+05
Df Model:	9		
Covariance Type:	nonrobust		

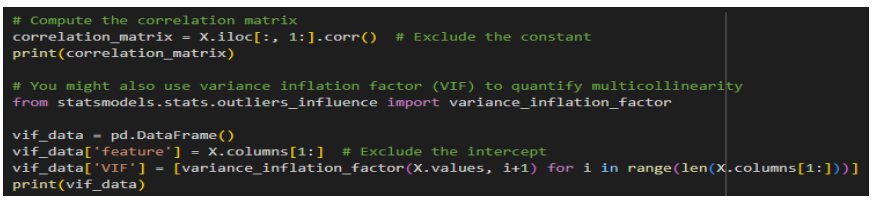
	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.0017	0.002	-0.803	0.422	-0.006	0.002
COMPLETEDCYCLEDURATION	1.044e-06	7.1e-08	14.694	0.000	9.05e-07	1.18e-06
Payload (kg)	1.024e-07	1.86e-09	55.101	0.000	9.88e-08	1.06e-07
Autonomous	-0.0077	0.001	-7.169	0.000	-0.010	-0.006
Available SMU Time	-1.094e-06	1.71e-07	-6.387	0.000	-1.43e-06	-7.58e-07
iMine Operating Hours	4.578e-06	2.72e-07	16.812	0.000	4.04e-06	5.11e-06
AT Available Time (iMine)	3.078e-08	1.4e-07	0.220	0.826	-2.43e-07	3.05e-07
Travelling Empty Duration	2.229e-05	3.75e-07	59.472	0.000	2.16e-05	2.3e-05
Travelling Full Duration	5.379e-05	4.55e-07	118.252	0.000	5.29e-05	5.47e-05
Idle Duration	-1.215e-07	4.48e-08	-2.714	0.007	-2.09e-07	-3.38e-08
Primary Machine Class Name_CAT 793F CMD	-0.0077	0.001	-7.169	0.000	-0.010	-0.006
Primary Machine Class Name_Cat 789C	0.0060	0.003	1.857	0.063	-0.000	0.012

Omnibus:	9938.408	Durbin-Watson:	1.406
Prob(Omnibus):	0.000	Jarque-Bera (JB):	761242.709
Skew:	1.243	Prob(JB):	0.00
Kurtosis:	31.330	Cond. No.	4.23e+22

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 6.88e-31. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

Figure 10.4. OLS Result Table.

The note section of the result also stated that there are strong multicollinearity problems. This can distort the regression coefficients and make the estimates extremely sensitive to changes. To resolve this, we need to look at the correlation among variables and VIF to quantify multicollinearity. The code use for the process can be seen in the *Figure 10.5.*



```
# Compute the correlation matrix
correlation_matrix = X.iloc[:, 1:].corr()  # Exclude the constant
print(correlation_matrix)

# You might also use variance inflation factor (VIF) to quantify multicollinearity
from statsmodels.stats.outliers_influence import variance_inflation_factor

vif_data = pd.DataFrame()
vif_data['feature'] = X.columns[1:]  # Exclude the intercept
vif_data['VIF'] = [variance_inflation_factor(X.values, i+1) for i in range(len(X.columns[1:]))]
print(vif_data)
```

Figure 10.5. VIF Implementation.

The results has shown further insight into issues related to multicollinearity. VIF result for Autonomous and the truck model suggest near-perfect to perfect multicollinearity. This happens due to autonomous variable is predictable from the truck models. The correlation matrix values also show that autonomous is perfectly negative correlate to truck model CAT 789C and perfectly positive for the other. This makes one of the variables redundant, in this case, 'Autonomous' will be dropped and OLS wil be re-run without the stated variable.

After the adjustment, the next step is to validate the model's predictive accuracy and analysing the residuals, and performing analysis the residuals and diagnostic checks. The results are showcased in the figures below.

- Model Validation:

  Root Mean Squared Error is approximately 0.01193 which is not too bad of the result.

- Distribution of Residuals:

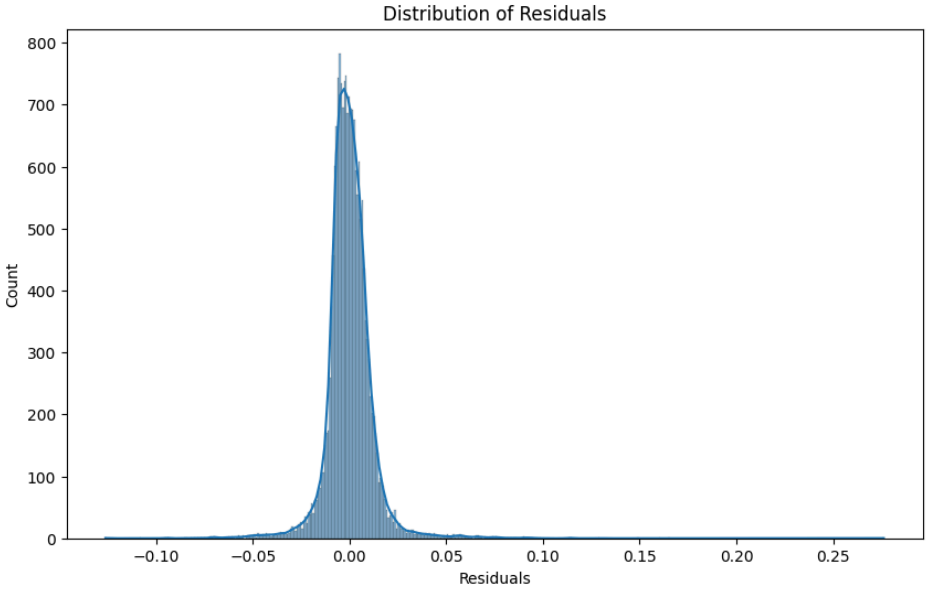


Figure 10.6. Distribution of Residuals.

A peak around 0 suggests that many residuals are small, however, long tail on the right hand side indicates that there are outliers with positive residuals.

- Q-Q Plot of Residuals:

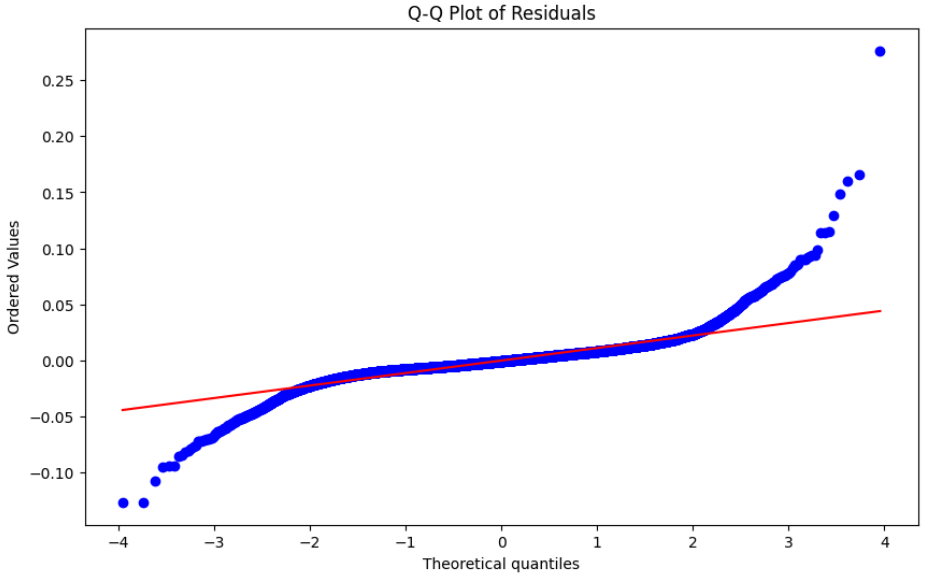


Figure 10.7. Q-Q Plot of Residuals.

The significance on both ends of the distribution shows that the residuals do not follow a normal distribution, further verify the outliers effect on the model prediction.

- Influence Plot:

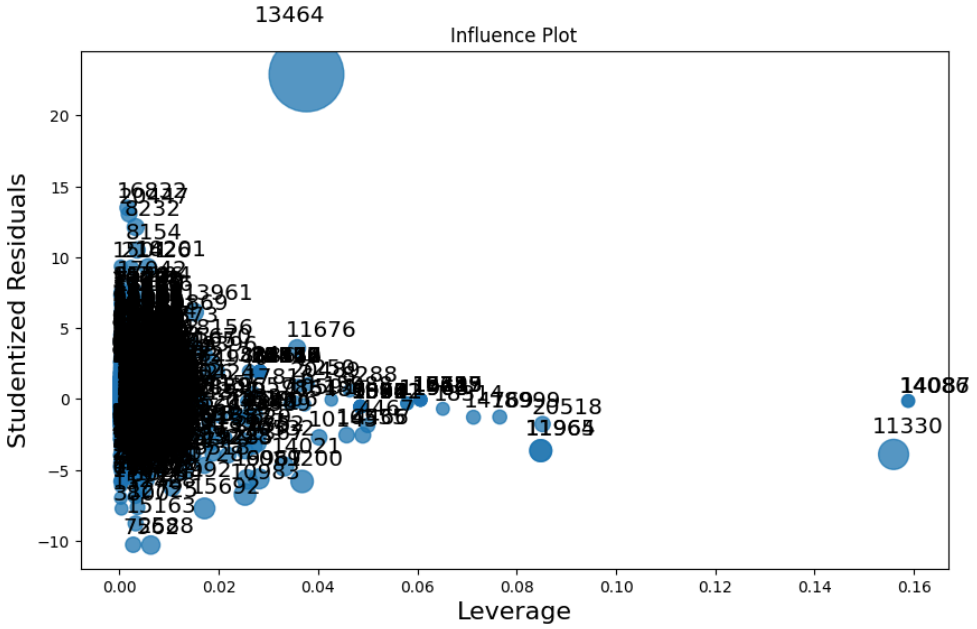


Figure 10.8. Influence Plot.

The influence plot shows several points that are far from the cluster of the rest of the data. This indicates the points that influence the regression result.

Overall, the analysis yields a model with good explanatory power as indicated by R-squared value. However, the diagnosis checks revealed areas where the model could be improved and

will be helpful for the next process of the project, including outlier management, adding more predictors/attributes and experiments with other complex models including XGBOOST as proposed.

11. Data Preprocessing

In the dataset, it can be observed that a lot of missing values were present. A solution to rectify this issue would be to clear missing values through deletion. Additionally, the Excel files provided by the source had major header misplacements that created an obstacle to interpreting data. Therefore, the team decided to use the CSV files supplemented instead.

```
df_delay= pd.read_csv('/content/drive/Shareddrives/Ai Capstone Dataset/DelayData.csv')
df_cycle= pd.read_csv('/content/drive/Shareddrives/Ai Capstone Dataset/CycleData.csv')
df_location= pd.read_csv('/content/drive/Shareddrives/Ai Capstone Dataset/LocationData.csv')

# Check for missing values
print("\n\nMissing Values of CycleData:\n")
print(df_cycle.isnull().sum())
print("\n\nMissing Values of DelayData:\n")
print(df_delay.isnull().sum())
print("\n\nMissing Values of LocationData:\n")
print(df_location.isnull().sum())

# Remove missing values
df_cycle = df_cycle.dropna(axis=1)
df_delay = df_delay.dropna(axis=1)
df_location = df_location.dropna(axis=1)
```

Figure 11.1 Data Preprocessing: Deletion of Missing Values.

To further prove our previous statement, feature scaling through a standard scaler was utilised for data transformation. The standard scaler standardises features by subtracting the mean and dividing by the standard deviation with a mean of the distribution around zero and a standard deviation of 1, enhancing accuracy for the model.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

X = df_cycle.drop('Fuel Used', axis=1)  # Features
y = df_cycle['Fuel Used']  # Target variable
# Take the object var only and change to int type
object_columns = X.select_dtypes(include=['object']).columns
label_encoders = {}
for col in object_columns:
    label_encoders[col] = LabelEncoder()
    X[col] = label_encoders[col].fit_transform(X[col])
#print(X.dtypes)
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
print(y.head())
```

Figure 11.2 Data Preprocessing: Feature Scaling.

Finally, the Overall Equipment Effectiveness (OEE) is calculated to define fuel consumption by evaluating each equipment's on-ground performance. The Overall Equipment Effectiveness is obtained from the product of Availability, Performance, and Quality.

OEE, or Overall Equipment Effectiveness, is widely used in mining and manufacturing. According to 3AG (n.d.), OEE provides a pointer of the proportion of manufacturing time used effectively. The higher the OEE score, the higher the efficiency.

The ratio of high-quality products produced is specified by the OEE by an organisation within a specific timeframe relative to the maximum potential output achievable if all machinery and processes are flawlessly operated as per 3AG's statement.

Availability is defined by the proportion of an equipment's operational time by comparing the ratio of net available time minus the downtime multiplied by 100. Performance is defined by the equipment's efficiency during operational periods, calculated by the ratio of operating time minus the speed losses multiplied by 100. Lastly, Quality is defined by the equipment's effectiveness by retrieving the ratio of operating time minus speed losses multiplied by 100.

```
[5]  ##OEE Calculation
     oee = df_cycle[['AT Available Time (iMine)', 'Down Time', 'iMine Operating Hours', 'OPERATINGTIME (CAT)', 'Idle Duration']]
     oee['Availability'] = ((oee['AT Available Time (iMine)'] - oee['Down Time']) / oee['AT Available Time (iMine)']) * 100
     oee['Performance'] = ((oee['OPERATINGTIME (CAT)'] - oee['Idle Duration']) / oee['OPERATINGTIME (CAT)']) * 100
     oee['Quality'] = ((oee['iMine Operating Hours'] - oee['Down Time']) / (oee['Down Time'] + tee['Idle Duration'])) * 100
     oee['TEE'] = oee['Availability'] * oee['Performance'] * oee['Quality']

     ##OEE Calculations
     def calculate_oee():
         return oee[['AT Available Time (iMine)', 'Down Time', 'iMine Operating Hours', 'OPERATINGTIME (CAT)',
                     'Idle Duration', 'Availability', 'Performance', 'Quality', 'OEE']]

     ##Print OEE Calculations
     print(calculate_oee())
```

Figure 11.3 Data Preprocessing: Overall Equipment Efficiency Calculation

12. XGBoost

12.1 Implementation

The model was built using Python due to the abundance of libraries available for constructing and evaluating the XGBoost model.

Several Python libraries were employed in our implementation. Among these, Pandas stands out as a well-known tool for data importation, manipulation, and preprocessing. In our approach, Pandas were utilised to import the dataset and perform basic data preprocessing tasks.

Another noteworthy library we utilised is Scikit-learn (sklearn), which offers a plethora of useful functionalities for model development. With sklearn.model, the dataset is divided into training and testing sets. Additionally, the Label Encoders feature has been employed from Scikit-learn, which assigns integers to the unique values of a column. This facilitates the use

of columns with integer data types in models. Scikit-learn also provided various evaluation methods, which were instrumental in assessing the model's performance.

The final and perhaps most crucial library utilised was the XGBoost library. This library serves as the foundation of our regression model, enabling the team to construct an XGBoost regression model using XGBRegressor.

The data is split using Scikit-learn's ‘train_test_split’ function from the ‘sklearn.model_selection’ module. This function divides the data into training and testing sets, following an 80:20 split where 80% of the data is allocated for training and the remaining 20% for testing the model.

```
from sklearn.preprocessing import LabelEncoder

X = df_cycle.drop('Estimated Fuel Used', axis=1)  # Features
y = df_cycle['Estimated Fuel Used']  # Target variable
# Take the object var only and change to int type
object_columns = X.select_dtypes(include=['object']).columns
label_encoders = {}
for col in object_columns:
    label_encoders[col] = LabelEncoder()
    X[col] = label_encoders[col].fit_transform(X[col])
#print(X.dtypes)
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (37875, 47)
X_test shape: (9469, 47)
y_train shape: (37875,)
y_test shape: (9469,)
```

Figure 12.1 Data Split and Preprocessing .

12.2 Initial Findings

As seen in *Figure 12.2*, the team split the data with test_size=0.2, indicating that 20% of the data is designated as the test dataset. The training data retained 37,875 rows, while the testing dataset contained 9,468 rows.

An essential step in the data preprocessing phase involves the utilisation of label encoders. These encoders are instrumental in converting textual data within the data frames into numerical int64 data. Since XGBoost models cannot process object-type columns directly, we must employ label encoders to transform these columns. The label encoder assigns a unique integer to each distinct data entry in the column, effectively replacing the text with integers. This conversion ensures that the column comprises only integers, maintaining its classification properties intact. Given that multiple columns contain object data types, we isolate them and iterate through them using a for loop, applying the label encoder to each selected column.

```
import matplotlib.pyplot as plt
# Initialize and train the XGBoost regressor
xgb_regressor = XGBRegressor(objective='reg:squarederror', eval_metric='rmse', random_state=42)
xgb_regressor.fit(X_train, y_train)
print(y_test.head())
# Predict
y_pred = xgb_regressor.predict(X_test)
print(y_pred)
```

Figure 12.2 Model Building and Training Code.

Following data preprocessing, the model is constructed, trained, and ultimately evaluated using the test dataset to yield accuracy metrics. As previously mentioned, the model is constructed using the XGBoost Python library, utilising the XGBRegressor function to build a regression model. The objective of this model is to minimise the squared error, as we are performing regression to predict the fuel usage of trucks based on multiple input features.

12.3 Evaluation

```
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)

mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error:", mae)

rmse = mean_squared_error(y_test, y_pred, squared=False)
print("Root Mean Squared Error:", rmse)
```

Figure 12.3.1 Model Evaluation Methods Code.

The model is evaluated using several different methods. The first and arguably most important is the mean squared error of the model because the mean squared error is the aim set for the model. A lower mean squared error means the model predicts values closer to the actual values.

```
Mean Squared Error: 2.2162830756948045e-05
Mean Absolute Error: 0.001010013454107246
Root Mean Squared Error: 0.004707741577120397
```

Figure 12.3.2 MSE, MAE, RMSE Results.

The results predicted by the XGBoost model show a very encouraging performance. For instance, the Mean Squared Error (MSE) is 2.216283e-05. In regression analysis, the closer this value is to 0, the better the model. This measures the average squared difference between the actual and predicted values. A value close to 0 suggests that the model's predictions are very close to the actual values. The Mean Absolute Error (MAE) also supports this statement as it has a value of 0.001010013454. This measures the average absolute value difference between actual and predicted values. Lastly, the Root Mean Squared Error (RMSE) measures the average magnitude of errors in the predicted values. Likewise, it has a value of 0.0047077 which is very close to the MAE and near 0, indicating excellent predictive accuracy.

Other evaluation methods include calculating the accuracy, precision, recall, F1-score, and ROC AUC for classification. However, this may not be as applicable to predicting fuel

consumption. Classification models assign instances to classes that require binary variables and discrete categories. To predict fuel consumption, it is more efficient to have precise estimates rather than assigning trucks to broad categories. Regression models allow for continuous predictions rather than classification, which would classify truck usage into 3-4 categories, such as low, medium, and high, leading to some loss of information. Regression, on the other hand, will allow us to examine fuel consumption as it is affected by other factors like speed, payload, load, etc.

12.4 Improvement

Overfitting commonly occurs when training a machine learning model. This usually happens due to too much detail and noise in the training data. One technique to evaluate a model's performance is k-fold cross-validation. It involves splitting the data into multiple folds, using one fold as a validation set and the rest to train the model. This process will be done multiple times, using different folds as the validation sets, resulting in a more realistic performance.

```python
#k-cross validation

from sklearn.model_selection import KFold, cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Define the number of folds for cross-validation
k = 5

# Initialize the cross-validation splitter
kf = KFold(n_splits=k, shuffle=True, random_state=42)

# Initialize your regression model (e.g., Linear Regression)
model = LinearRegression()

# Perform k-fold cross-validation and calculate regression metrics
mse_scores = -cross_val_score(xgb_regressor, X_train, y_train, cv=kf, scoring='neg_mean_squared_error')
r2_scores = cross_val_score(xgb_regressor, X_train, y_train, cv=kf, scoring='r2')

print("Mean Squared Error (MSE) Scores for Each Fold:", mse_scores)
print("R-squared Scores for Each Fold:", r2_scores)
average_mse = mse_scores.mean()
print("Average Mean Squared Error (MSE):", average_mse)
average_r2 = r2_scores.mean()
print("Average R-squared:", average_r2)
```

Figure 12.4.1 K-Fold Cross Validation Methods Code.

In the code above, the team first split the data into k folds of 5. The folds are then trained with one fold reserved to be the validation set. The ‘cross_val_score’ function from scikit-learn automatically splits the data into k folds according to the specified cv parameter and will use the remaining k-1 folds as the training set. The scoring metrics are then calculated for each fold, being the MSE, r2, and their average.

```
Mean Squared Error (MSE) Scores for Each Fold: [1.10035438e-05 8.13861881e-06 5.56417922e-06 7.10598865e-06
 3.37614875e-05]
R-squared Scores for Each Fold: [0.98115772 0.98787407 0.99079437 0.98820484 0.94467711]
Average Mean Squared Error (MSE): 1.3114763608374002e-05
Average R-squared: 0.9785416223048493
```

Figure 12.4.2 MSE, R-squared, Average MSE, Average R-squared results.

The MSE calculated by the k-cross validation for each fold is shown in the picture above. Comparing it with the MSE calculated in the original model, most of the values are higher, thus furthering away from zero, proving that the original model might have overfitted the dataset. Taking the average of the MSE for each fold results in the Average MSE of 1.3114e-05, which contradicts the previous statement of overfitting as it is even less than the MSE calculated by the original model. This may result from outliers or inconsistent data in the dataset affected by external factors. Since the k-cross validation separates the dataset into different folds for validation testing, we can look at the MSE scores for each fold and compare those with significantly different results. The second and the 4th fold, for example, show a significant increase in MSE as compared to the rest. However, since the average MSE shows that overall the model is great, it can be concluded that the machine learning model we have created is reliable since k-cross validation typically generalises better to unseen data due to being exposed to different parts of the dataset during training.

The other metric used is the R-squared value, which measures the proportion of the variance from 0 to 1, a value closer to 1 that shows a better model. The model above has an average R-squared value of 0.9785, which indicates that a large proportion of the dependent variables is explained by the independent variables. This suggests that the model successfully encaptures the underlying relationships within the dataset.

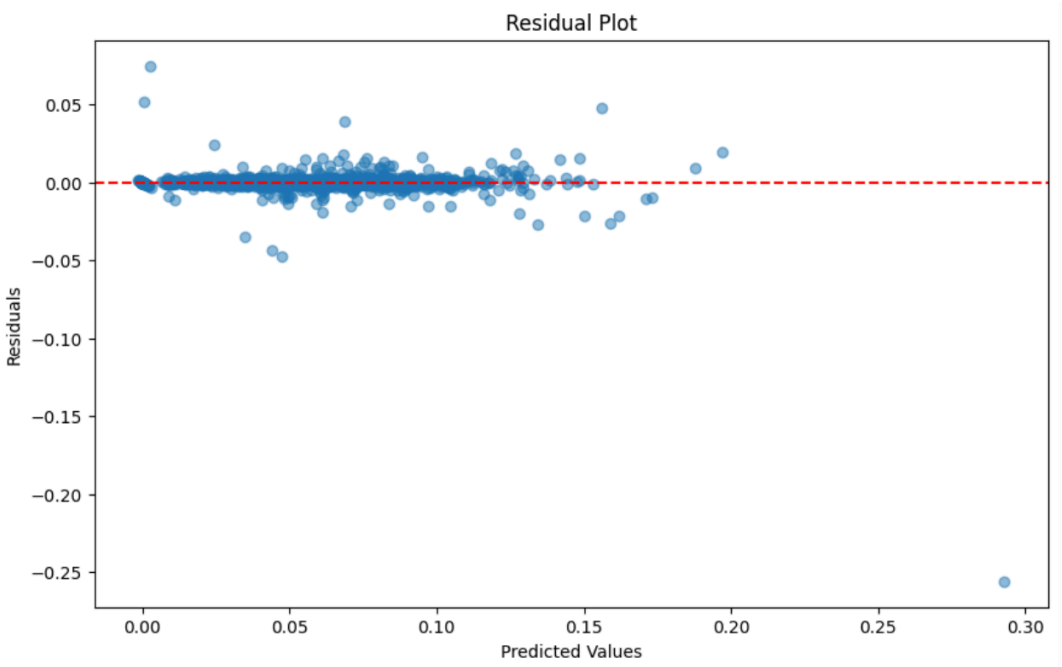


Figure 12.4.3 Residual Plot for XGBoost Model

The residual plot for the model shows a generally random pattern of dots, indicating that the model's predictions are largely unbiased and effectively capture the underlying relationship between the input features and the target variable. However, there is a noticeable clump of data located around the 0-0.15 predicted values. This concentration of residuals corresponds to a high density of actual target values within this range, reflecting the natural distribution of the data rather than a model issue.

The residuals within this clump do not show any discernible pattern and are close to the zero line, suggesting that the model performs well for these predictions. Overall, the residual plot supports the conclusion that the optimised model provides accurate and unbiased predictions, with the observed clustering aligning with the expected distribution of the target variable.

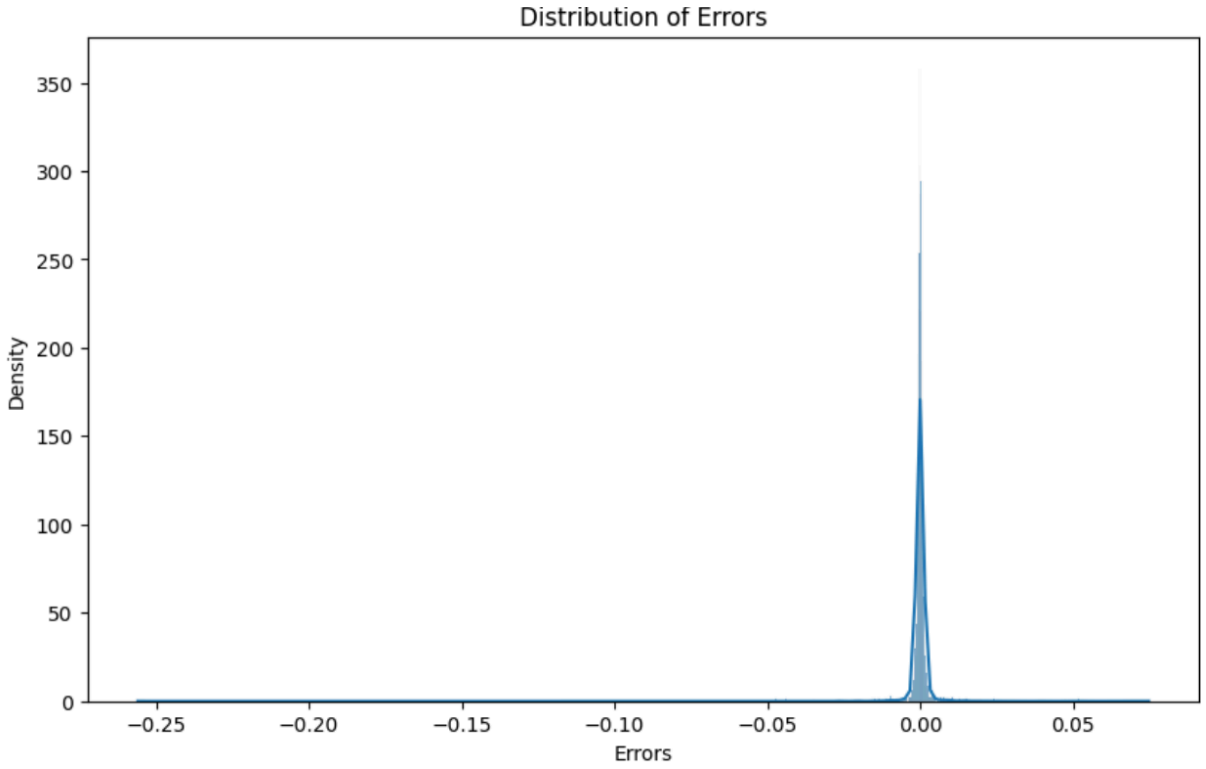


Figure 12.4.4 Distribution of errors for XGBoost Model

The distribution of errors plot depicts the density of errors observed in the model's predictions. A large density around 0 on the x-axis indicates that a significant portion of the errors is centered close to zero, implying that the model tends to make predictions that are very close to the actual values. This concentration of errors around zero suggests that the model is generally accurate, as it indicates minimal deviation between predicted and actual values for a large portion of the dataset. Overall, the distribution of errors plot shows that the model is highly accurate, with almost no error exceeding the -0.5 and 0.5 marks.

While the model is producing great results, the utilization of other optimisation methods, such as random search and Monte Carlo Simulation can further improve the model. This will be covered below.

12.5 Applying Random Search Algorithm

Random search is a hyperparameter optimisation technique used to enhance machine learning models. Random search works as a hyperparameter optimisation technique that aims to find the best combination of hyperparameters for a machine learning model by randomly sampling different sets of hyperparameters from a pre-defined range. Unlike grid search, which evaluates all possible combinations of specified hyperparameters, random search randomly selects combinations within these ranges and evaluates them.

This makes the random search approach particularly useful because it can more efficiently cover a broader search space, often finding good hyperparameter configurations with fewer iterations.

For XGBoost regression models, which have numerous hyperparameters such as learning rate, max depth, and number of estimators, and many more, random search can significantly improve performance by identifying optimal hyperparameter settings for the model.

```
param_grid = {
    'n_estimators': [100, 200, 300, 400, 500],
    'learning_rate': [0.01, 0.05, 0.1, 0.2, 0.3],
    'max_depth': [3, 4, 5, 6, 7],
    'min_child_weight': [1, 2, 3, 4, 5],
    'gamma': [0, 0.1, 0.2, 0.3, 0.4],
    'subsample': [0.6, 0.7, 0.8, 0.9, 1.0],
    'colsample_bytree': [0.6, 0.7, 0.8, 0.9, 1.0],
    'reg_alpha': [0, 0.1, 0.5, 1, 2],
    'reg_lambda': [0, 0.1, 0.5, 1, 2]
}

# Initialize XGBRegressor
xgb_regressor = XGBRegressor(objective='reg:squarederror', eval_metric='rmse', random_state=42)

# Initialize RandomizedSearchCV
random_search = RandomizedSearchCV(estimator=xgb_regressor, param_distributions=param_grid, n_iter=100,
                                   scoring='neg_mean_squared_error', cv=5, verbose=2, random_state=42, n_jobs=-1)

# Fit the random search to the data
random_search.fit(X_train, y_train)

# Best hyperparameters
print("Best hyperparameters:", random_search.best_params_)
```

Figure 12.5.1 Random Search Hyperparameter optimiser for XGBoost Regression Model.

Figure 12.5.1 above shows the implementation of Random Search on our XGBoost regressor model. Random search is implemented using the function `RandomizedSearchCV` from the `sklearn.model_selection` library, which facilitates the process of hyperparameter tuning by sampling random combinations of hyperparameters from a pre-defined search space. The `param_grid` variable is responsible for storing this search space, specifying the range or distribution of values for each hyperparameter we need to explore, such as learning rate, max depth, and number of estimators.

The metrics used for evaluating the performance of the hyperparameters during the random search are passed as arguments to the `RandomizedSearchCV` function. In this case, the primary metric is mean squared error (MSE), which measures the average of the squares of the errors, providing a sense of how well the model's predictions match the actual values. As seen in *Figure 12.5.1*, the random search is configured to run for 100 iterations, meaning it will evaluate 100 different combinations of hyperparameters to find the optimal set that minimises the MSE. This ensures a thorough exploration of the search space, increasing thelikely hood of finding the best set of hyperparameter combinations.

12.5.1 Application of Monte Carlo Simulation With Random Search

To further improve the model, we implement the Monte Carlo simulation along with the Random Search hyperparameter optimisation technique. Monte Carlo simulation is a widely used technique in mathematics, computing, and engineering fields to better understand uncertainty and probabilistic attributes. The implementation of the Monte Carlo simulation aims to mitigate the randomness introduced by using random search, with the hope of finding an even better set of hyperparameters.

```
#Monte Carlo Simulation Parameters
num_simulations = 5
sample_size = 0.8

results = []
for i in range(num_simulations):
    X_sample, _, y_sample, _ = train_test_split(X, y, train_size=sample_size, random_state=i)
    xgb_regressor = XGBRegressor(objective='reg:squarederror', eval_metric='rmse', random_state=42)
    #Random Search
    random_search = RandomizedSearchCV(estimator=xgb_regressor, param_distributions=param_grid, n_iter=50,
                                       scoring='neg_mean_squared_error', cv=5, verbose=2, random_state=42, n_jobs=-1)
    random_search.fit(X_sample, y_sample)

    #Append Results
    results.append({
        'best_params': random_search.best_params_,
        'best_score': random_search.best_score_
    })
#convert results to dataframe
results_df = pd.DataFrame(results)
```

Figure 12.5.1.1 Application of Monte Carlo and Random Search

The integration of Monte Carlo simulation with random search presents a powerful tool for hyperparameter optimisation in our XGBoost model. Monte Carlo simulation operates by executing multiple iterations of the random search technique, each iteration producing a distinct set of best hyperparameters. These optimal hyperparameter configurations are systematically recorded and stored within a structured data frame for further analysis. Subsequently, we analyse this collection of best parameters to identify the mode, which represents the hyperparameter configuration that appears most frequently across the simulations. This mode serves as our ultimate selection for the optimal hyperparameters. By using this combination technique, we not only harness the benefits of random search's large search space but also employ Monte Carlo simulation to filter and distil these diverse results into a single ultimate hyperparameter.

```
model_best_params = results_df['best_params'].mode().iloc[0]
for key, value in model_best_params.items():
    print(f"{key}: {value}")
```

```
subsample: 0.7
reg_lambda: 0.5
reg_alpha: 0.1
n_estimators: 500
min_child_weight: 4
max_depth: 4
learning_rate: 0.3
gamma: 0
colsample_bytree: 0.9
```

Figure 12.5.1.2 Best Optimised Hyperparameters

Figure 12.5.1.2 shows the results of the Monte Carlo and Random Search optimisation. These hyperparameters are highly likely to be the best possible for the model. This is because these hyperparameters have consistently shown to be the most optimal set of values throughout multiple iterations.

Overall, while powerful, the combination of Random Search and Monte Carlo simulation is not foolproof. There still lies a possibility that the multiple iterations of random search did not find the best set of values. However, after running the simulation several times, we are confident that these are the best hyperparameters. Thus, we can proceed to build the optimised model with these newly optimised hyperparameters.

12.5.2 Model with Optimised Hyperparameters

With the newly found hyperparameters, we are going to build a new model. This model will be tested thoroughly and compared to the original XGBoost model to assess the improvements made by the optimised hyperparameters. We aim to determine if the optimiser has significantly improved the accuracy of the model's predictions.

```
best_params = {
    'subsample': 0.7,
    'reg_lambda': 0.5,
    'reg_alpha': 0.1,
    'n_estimators': 500,
    'min_child_weight': 4,
    'max_depth': 4,
    'learning_rate': 0.3,
    'gamma': 0,
    'colsample_bytree': 0.9
}
# Initialize XGBRegressor with best hyperparameters
xgb_regressor_best = XGBRegressor(objective='reg:squarederror', eval_metric='rmse', random_state=42, **best_params)
xgb_regressor_best.fit(X_train, y_train)
# Predict
y_pred = xgb_regressor_best.predict(X_test)
```

Figure 12.5.2.1 Model With Optimised Hyperparameters

Hyperparameter	Value
Sub-sample (subsample)	0.7
Regression lambda (reg_lambda)	0.5
Regression alpha (reg_alpha)	0.1
n_estimators	500
Minimum child weight(min_child_weight)	4
Maximum depth (max_depth)	4
learning rate (learning_rate)	0.3
gmma	0
colsample_bytree	0.9

Table 12.5.2.1 Hyperparameters of Optimized XGBoost

The new optimised model is made using the list of best hyperparameters the random search optimiser provides. The hyperparameters used and their significance are as follows:

- subsample (0.7): This parameter specifies the fraction of training data to be randomly sampled for each tree. The 0.7 means that 70% of the training data will be used to grow each tree, reducing the chance of overfitting.
- reg_lambda (0.5): This parameter helps control the model's complexity by penalizing large weights. A value of 0.5 means a moderate level of regularization and reduces overfitting.
- reg_alpha (0.1): Known also as Lasso regularization. A value of 0.1 indicates a small amount of Lasso regularization on the weights.
- n_estimators (500): This parameter sets the number of trees to be built. Having the right number of trees will reduce overfitting.
- min_child_weight (4): This parameter specifies the minimum sum of instance weight needed in a child. A value of 4 means that a node will not be split if the resulting nodes have fewer than four instances.
- max_depth (4): This parameter determines the maximum depth of a tree. A value of 4 means that the tree can grow up to 4 levels.
- learning_rate (0.3): This parameter scales the contribution of each tree. A learning rate of 0.3 is relatively high, resulting in faster learning at the risk of overfitting.
- gamma (0): This parameter specifies the minimum loss reduction required to partition a leaf node of the tree further. A value of 0 means no minimum loss reduction is required, allowing the trees to grow freely.
- colsample_bytree (0.9): This parameter specifies the fraction of features to be randomly sampled for each tree. Using a large portion of the features helps prevent overfitting.

12.5.3 Optimised Model Evaluation

The optimised model is evaluated using the same metrics as the original XGBoost model. This approach makes it easier to compare the two models and assess the improvements made by the optimised model. Overall, the optimised model significantly enhances the performance of the already well-performing model, making it an even better regressor.

```
Mean Squared Error: 1.091423778488167e-05
Mean Absolute Error: 0.0011831180896541916
Root Mean Squared Error: 0.003303670350516478
```

Figure 12.5.3.1 Optimised Model MSE, MAE, and RMSE

Figure 12.5.3.1 shows the MSE, MAE, and RMSE of the optimised model. They show a very encouraging performance. For instance, the Mean Squared Error (MSE) is 1.091424e-05. In regression analysis, the closer this value is to 0, the better the model. A value close to 0 suggests that the model's predictions are very close to the actual values. The Mean Absolute Error (MAE) also supports this statement as it has a value of 0.00118311. This measures the average absolute value difference between actual and predicted values. Lastly, the Root Mean Squared Error (RMSE) measures the average magnitude of errors in the predicted values. Likewise, it has a value of 0.00330367 which is very close to the MAE and near 0, indicating excellent predictive accuracy of the optimised model. Now, we will compare this result to the original xgboost regression model and see the improvements.

```
#Improvements to the mse (percent)
mse_percent= (mse - mse_best) / mse * 100
print("MSE improvement: {:.2f}%".format(mse_percent))
```

Figure 12.5.3.2 MSE Improvement Calculation.

From the calculation done using the snippet of python code above, we can figure out the improvement of MSE of the optimised model compared to the original model in terms of percentages.

```
MSE improvement: 50.75%
```

Figure 12.5.3.3 MSE Result.

The improvement in results shows that the optimised model has an MSE that is 50.75% better than the original model. This indicates that the optimised model has substantially reduced the error compared to the original model. While MSE is an excellent metric to judge the improvement of the model, other comparisons will also be performed. These include comparisons of the residual plot and the error distribution graph.

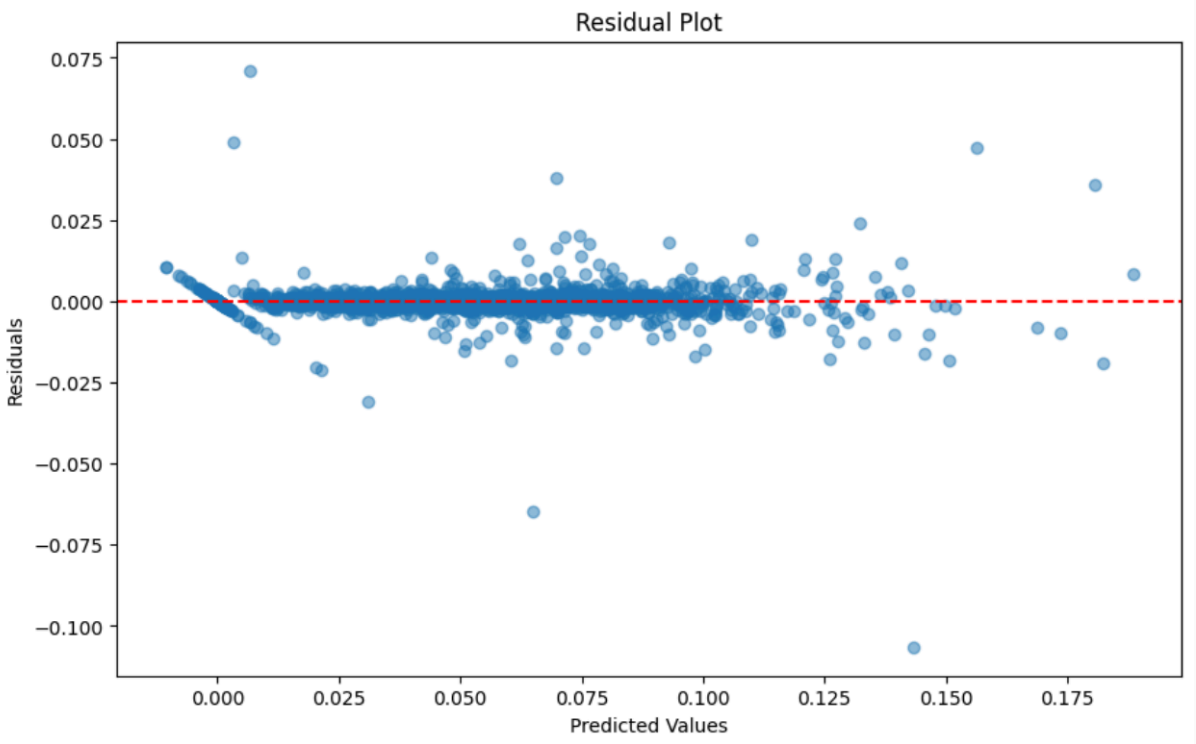


Figure 12.5.3.4 Residual Plot for Optimised Model

The residual plot for the optimised model follows a similar pattern to the original model. The plot shows a generally random pattern of dots, indicating that the model's predictions are largely unbiased and effectively capture the underlying relationship between the input features and the target variable. The clump of data between 0.0 and 0.1 still exists, which is logical since the real values of the data are around that range. The improvement in the optimised model can be clearly seen by looking at the x-axis values, where the values increase by 0.025 increments compared to the 0.05 increments in the original plot. This shows that the optimised model is even better at predicting the values that are relatively far from the clump of data points between 0.0 and 0.1

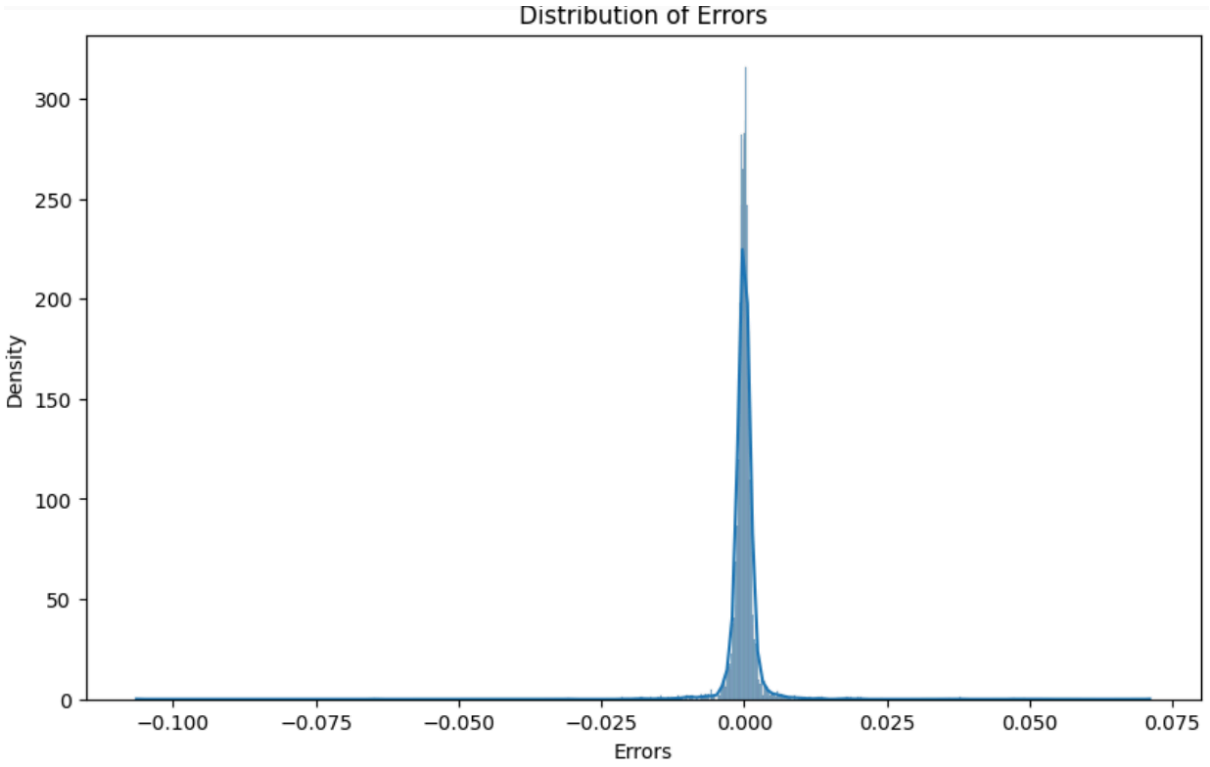


Figure 12.5.2.5 Distribution of Errors for Optimised Model

The distribution of errors plot for the optimised model follows a similar pattern to the original distribution of errors plot. It shows a large density around 0 on the x-axis, indicating that a significant portion of the errors is centered close to zero. This implies that the optimised model tends to make predictions that are very close to the actual values. The main difference between the optimised model plot and the original plot is the increments on the x-axis. The graph for the optimised model shows an increment of 0.025, while the original graph shows an increment of 0.05. This demonstrates that the optimised model is much more accurate than the original model, effectively eliminating the larger errors that the original model made.

In conclusion, the optimised model showed significant improvements in multiple areas compared to the base model. The main improvement made by the optimised model is in eliminating the predicted values with large errors. This is evidenced by the shrinking of the x-axis in both graphs, indicating that the errors are generally closer to 0 and that the larger error values have been eliminated. Another substantial improvement is in the Mean Squared Error (MSE) of the model, with a 50.75% improvement, which is significant and proves that the optimisation has effectively enhanced the model.

12.6 Benefits and Limitations

The benefits of XGBoost Regression model are as follows:

- XGBoost is known for its exceptional predictive accuracy, this is proven by the fact that our model excels in finding relationships in the data, making it ideal for regression tasks that demand precision.

- XGBoost has multitude of tunable hyperparameters that are carefully engineered to strike a balance between complexity and generalization. These hyperparameter when tuned, can significantly reduce overfitting of the model, improving the predictive accuracy segnificatly.

The benefits of Random Search and Monte Carlo simulation in optimizing the model are as follows:

- Random search explores a wide range of hyperparameter values, which can help in finding a better combination that might be missed when using a smaller range of hyperparametes.
- Random search can be computationally more efficient than grid search, especially when only a subset of hyperparameter is large. This is because unlike grid search random search does not try all the different combination of haperparameters.
- Leveraging Monte Carlo simulation further refines random search results. Monte Carlo simulation ensures that multiple iteration of random search come up with the same hyper parameter reducing the random nature of Random search.

The limitations of XGBoost Regression are as follows:

- XGBoost can be challenging to interpret due to its complexity. Understanding how the model arrives at its predictions will require additional effort.
- XGBoost performance is sensitive to hyperparameters, and tuning them effectively require significant computational resources and the right optimisation methods
- .XGBoost may not perform as well with small datasets compared to larger ones. It requires a sufficient amount of data to generalize effectively and may overfit when trained on small datasets.

The limitations of Random Search and Monte Carlo simulation in optimizing the model are as follows:

- While more Random Search efficient than grid search, it can still be computationally expensive, especially with a large hyperparameter search space and high-dimensional data.
- While the combination of random search and Monte Carlo simulation is beneficial, it does not guarantee the most optimal prediction of the hyperparameters due to the inherent randomness of random search.

Despite the limitations listed above, the benefits of XGBoost still outweigh its drawbacks. This is because its limitations can be mitigated by using the random search hyperparameter optimiser and having a sufficient amount of data.

13. Application of Other Machine Learning Algorithms

13.1 Random Forest

13.1.1 Introduction

IBM (n.d.) defines a random forest as a machine learning algorithm that combines many decision trees to produce a single output. The approach requires three major hyperparameters: node size, number of trees, and number of sample features. The benefit of this approach is that its classifiers may be used for both regression and classification, according to IBM (n.d.).

Our study focuses on supervised learning, and we noticed that Random Forest is one of the most recommended machine learning algorithms. In this sense, Random Forest can manage big datasets while still producing reliable findings even when the data is noisy or missing.

Furthermore, Random Forest specialises in detecting non-linear correlations and interactions in data, making it appropriate for applications with complex or numerous patterns. Random Forest is extremely adaptable and performs well in a variety of supervised learning applications, such as spam email classification, customer churn prediction, and medical diagnosis.

13.1.2 Implementation and Initial Findings

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

# Initialize the Random Forest regressor

rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)
rf_regressor.fit(X_train, y_train)

# Make predictions on the test data
predictions = rf_regressor.predict(X_test)
```

```
from sklearn.linear_model import LinearRegression
# X_train, X_test, y_train, y_test are already defined so we train our model
model= LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

#Create random samples of n =1000
sample_n = np.random.choice(len(y_test), size=1000, replace=False)
```

Figure 13.1.2 Random Forest Implementation.

As seen in the figure above, the machine learning model is implemented to make predictions by first preparing the data and then splitting it into training and testing data. The library used to import the data splitting is known as scikit-learn, namely train_test_split for data splitting and LabelEncoder for encoding categorical variables.

Further discussing the above, features x and target variable y are defined based on the DataFrame df_cycle, with 'Fuel Used' being the target variable. Any categorical values identified (.select_dtypes(include=['object'])) are encoded using LabelEncoder to convert them into integer values to match the numerical inputs in the dataset.

By implementing train_test_split(), the dataset is split into training and testing. The proportion of dataset assigned to the test dataset is defined by the test_size parameter and random_state ensures that the split is consistent.

A Random Forest regressor model is then initialised with 100 decision trees (n_estimators) and the same starting point for the same sequence of random numbers (random_state= 42). The model is then trained on the training dataset and predictions are then made based on the given trained dataset.

Finally, the Mean Squared Error (MSE) is calculated to evaluate the performance of the Random Forest model by finding the average squared difference between actual and predicted values, providing a measure of prediction accuracy.

13.1.3 Evaluation

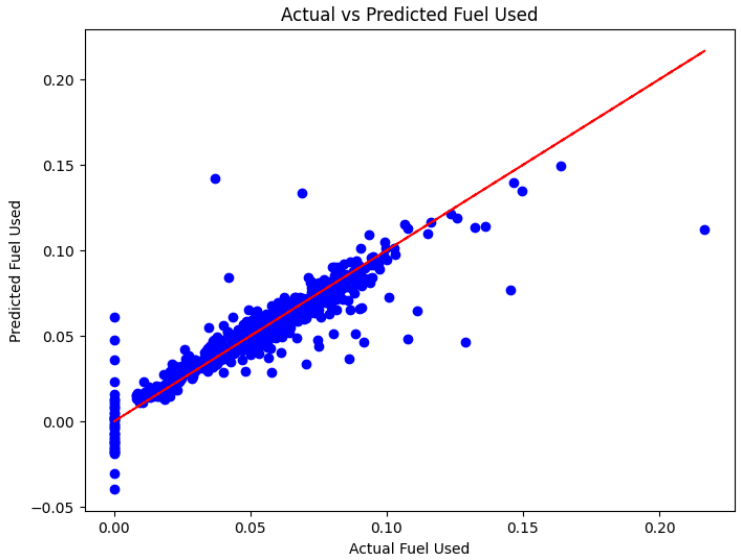


Figure 13.1.3.1 Random Forest Actual vs Predicted Fuel Used Plot.

```
from sklearn.metrics import mean_squared_error, r2_score

# Calculate MSE
mse_linear = mean_squared_error(y_test, y_pred)

# Calculate R^2 score
r2_linear = r2_score(y_test, y_pred)

print("Linear Regression Model:")
print("MSE:", mse_linear)
print("R^2 Score:", r2_linear)
```

```
Linear Regression Model:
MSE: 7.769146158023225e-05
R^2 Score: 0.8720771038298945
```

Figure 13.1.3.2 Random Forest Actual vs Predicted Fuel Used Results.

The graph represents a linear relationship between predicted and actual values as the predicted values grow with actual values (directly proportional). However, it is important to highlight that consistency of the model's performance is still questionable as a greater

difference is observed with even larger values. This implies that while the model works well at first, there may be certain outliers that go undetected.

Furthermore, the MSE takes into account the differences of squared values to measure the compatibility between predicted and actual values, serving as a model accuracy indicator. In this scenario, the MSE is valued at (7.77e-05), indicating that the predicted outcomes are similar to actual values as the value is very close to zero.

In addition, the coefficient of determination (r2) of 0.872 implies that the linear regression model has an accuracy rate of 87.2 %. This is considered an excellent r2 value, indicating that the model fits the data relatively well.

```
Mean Squared Error: 2.2162830756948045e-05
Mean Absolute Error: 0.001010013454107246
Root Mean Squared Error: 0.004707741577120397
```

Figure 13.1.3.4 Random Forest Results vs XGBoost Results (Pre Random Search).

The XGBoost algorithm is seen to have an MSE value of (2.22e-05), which is closer to zero compared to (7.77e-05), indicating that the XGBoost model has a higher accuracy than the Random Forest model. In this case, the XGBoost is more preferable.

13.1.4. Improvement

```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import RandomizedSearchCV, train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

# Define a smaller parameter grid
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [5, 10],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
}

# Reduce the number of iterations
n_iterations = 5

# Define the sample size
sample_size = 500

mse_results = []
feature_importances = []

for iteration in range(n_iterations):
    # Randomly sample the data
    random_indices = np.random.choice(len(X), size=sample_size, replace=False)
    X_sampled = X.iloc[random_indices]
    y_sampled = y.iloc[random_indices]

    # Split the sampled data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X_sampled, y_sampled, test_size=0.2, random_state=iteration)

    # Initialize the Random Forest regressor
    rf_regressor = RandomForestRegressor(random_state=42)
```

Figure 13.1.4 Random Forest Parameter Grids and Iterations.

A parameter grid, essentially a collection of hyperparameters, is used to optimise the current Random Forest model. The number of decision trees have values of both 100 and 200 The values will be tested and the better value is chosen for optimisation. The maximum depth of the trees (in this case, set to 5 and 10 respectively) is also tested and compared. Finally, the

minimum number of samples required to split an internal node and form a leaf node are also compared. This parameter grid compares multiple values within each parameter and combines them to showcase best performance of the Random Forest machine learning model.

The sample size is reduced to 500 and an iteration of 5 is set in order to speed up machine learning time consumption.

13.1.5 Applying Optimisation Algorithm

```
    # Fit the RandomizedSearchCV
    random_search.fit(X_train, y_train)

    # Get the best model
    best_rf_regressor = random_search.best_estimator_

    # Make predictions on the test data
    predictions = best_rf_regressor.predict(X_test)

    # Calculate mean squared error
    mse = mean_squared_error(y_test, predictions)
    mse_results.append(mse)

    print(f"Iteration {iteration + 1}/{n_iterations} - MSE: {mse}")

       # Get feature importances
    feature_importances.append(best_rf_regressor.feature_importances_)

# Print average MSE over all iterations
print("Average MSE:", np.mean(mse_results))
```

Figure 13.1.5.1 Random Forest Optimisation by Random Search.

A Monte Carlo simulation within random search is utilised for this model's optimization. A random search is deployed to search for the best combination of hyperparameters. The random search is initialised with the regressor as an estimator and the parameter grid to set boundaries for the search for the hyperparameters. Furthermore, the search is fitted to the training data sets and the hyperparameters that yield the highest performance are then shown by using the best estimator based on the parameter grid.

Preceding the statement above, the test data sets are then used to determine predictions using the best random forest regressor and the MSE of both target values are then calculated per iteration. Finally, the feature importances of the best Random Forest model is extracted and appended to a list to further analyse the model's accuracy.

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
Iteration 1/5 - MSE: 0.00013163825727515057
Fitting 5 folds for each of 4 candidates, totalling 20 fits
Iteration 2/5 - MSE: 2.673757628593712e-05
Fitting 5 folds for each of 4 candidates, totalling 20 fits
Iteration 3/5 - MSE: 3.163901555716143e-05
Fitting 5 folds for each of 4 candidates, totalling 20 fits
Iteration 4/5 - MSE: 2.5869468795271217e-05
Fitting 5 folds for each of 4 candidates, totalling 20 fits
Iteration 5/5 - MSE: 3.7349228247777386e-05
Average MSE: 5.0646709232259547e-05
```

Figure 13.1.5.2 Random Forest Optimisation Results.

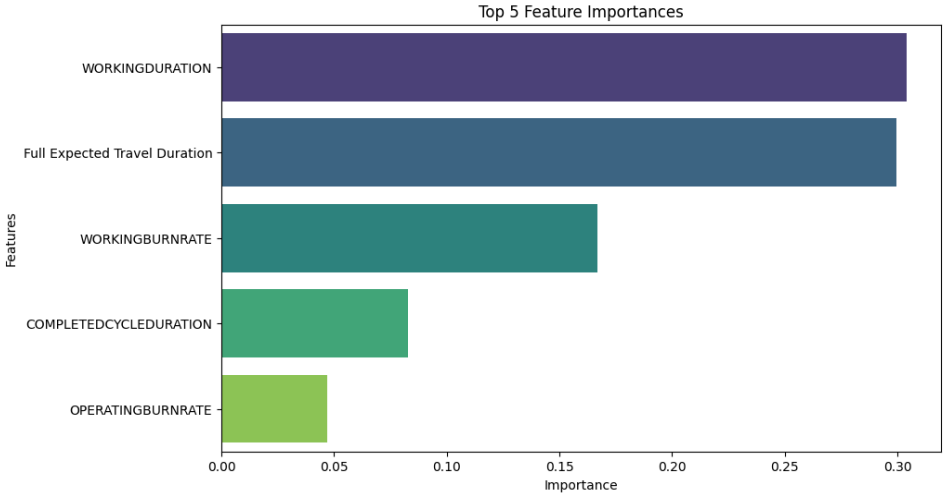


Figure 13.1.5.3 Random Forest Top 5 Feature Importances.

As observed in the figures above, the average MSE is 5.06e-05, which is an even closer value to zero compared to 7.77e-05(mse value pre-optimisation). This indicates that a higher accuracy is prominent after the model has been optimised.

Furthermore, the most significant feature for the model is ‘WORKINGDURATION’ as seen in *Figure 13.1.5.3.*.

Result of Random Forest After Optimization:

```
Iteration 1/5 - R^2: 0.9303501517365073
Iteration 2/5 - R^2: 0.9303501517365073
Iteration 3/5 - R^2: 0.9303501517365073
Iteration 4/5 - R^2: 0.9303501517365073
Iteration 5/5 - R^2: 0.9303501517365073
Average R^2: 0.9303501517365073
Average MSE: 5.0646709232259547e-05
```

Result of XGBoost After Optimization:

```
Mean Squared Error (MSE) Scores for Each Fold: [1.10035438e-05 8.13861881e-06 5.56417922e-06 7.10598865e-06
 3.37614875e-05]
R-squared Scores for Each Fold: [0.98115772 0.98787407 0.99079437 0.98820484 0.94467711]
Average Mean Squared Error (MSE): 1.3114763608374002e-05
Average R-squared: 0.9785416223048493
```

Figure 13.1.9 Random Forest vs XGBoost After Optimisation.

As seen in the figures above, the XGBoost learning model after optimisation has an MSE value of 1.311 e-05, a closer value to 0 compared to the Random Forest after optimisation and r2 value of 97.8% , which indicates a higher accuracy. Therefore, this implies that the XGBoost algorithm is more preferred.

13.1.6 Benefits and Limitations

This section will mostly be referred from GeeksForGeeks (2024) and my personal experience utilising the random forest model:

Benefits:

1. **High Accuracy:** As observed in the figures above, the accuracy for the random forest is almost always nearly accurate, though not as accurate as XGBoost. According to GeeksForGeeks (2024), this is due to the utilisation of multiple decision trees instead of a single decision tree.
2. **Handles Both Numerical and Categorical Data:** Specifically for this dataset, there were mixed up values between numerical and categorical data that are required for analysis. Random forest has the ability to utilise both regression and classification, whichever is more beneficial for the machine learning model, in this case regression.
3. **Estimating Feature Importance:** The Random Forest machine learning model takes into account the features that contribute the most to predict the model with high accuracy and result in less noise , as mentioned by GeeksForGeeks (2024).

Limitations:

1. **Memory Usage:** As mentioned by GeeksForGeeks (2024), the training data, feature splits, and leaf node predictions are stored in each decision tree, which consumes a lot of memory. This may serve as a limitation for people with limited hardware specifications.
2. **Prediction Time:** While running the optimization for the datasets, the first prediction time took over an hour to fully load which resulted in implementation of very small sample sizes and parameter values. Despite efforts to decrease the parameters, it still took around 10 full minutes to load it completely.
3. **Overfitting:** As the model is very good for generalisation, it cannot be used to handle very specific values. As mentioned by GeeksForGeeks (2024), The algorithm's ability to reproduce the training data excessively can result in lower performance on unexpected occurrences.

13.2 Neural Networks

13.2.1 Introduction

Neural network is a machine learning model that is designed to be able to encapture and learn from datasets as a human brain could. Every neural network consists of an input layer, hidden layers, and an output layer. In these layers are artificial neurons which are usually the main factor in making the model accurate with its predictions, as it involves tweaking the weights and biases in order to achieve the most optimal parameter. Each neuron can have different weights and biases, therefore really mimicking the human brain. As the number of neurons reaches a couple of hundreds, it is unlikely that we will fine tune each neuron's parameters on our own. When observing one of the decisions made by the neural network model, it makes the next decisions based on the output of the previous decisions or layers (IBM, 2024). This makes neural networks a very powerful algorithm when wanting to solve complex problems.

13.2.2 Implementation

We used the keras library imported from tensorflow in order to create the neural network model. Similar to the XGBoost algorithm, we imported several other libraries such as pandas and sklearn. The most important library needed for this model is the keras library, as we need to import packages like Sequential, Dense, and EarlyStopping in order to build the NN model.

Using the pre-processed data, the dataset is split into the training and testing set. The X training and testing set will then be scaled for the model to balance the impact of the variables well.

```
#Neural Network (NN) Algorithm
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from sklearn.preprocessing import LabelEncoder
from keras.utils import to_categorical

from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.callbacks import EarlyStopping

#Performing Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Figure 13.2.2 Library Imports and Data Scaling Code.

13.2.3 Initial Findings

After scaling the dataset, the model is now ready to be initialised. Using the Sequential package, the artificial neural network is created, and since the whole premise of the neural network model is to have the algorithm learn by passing through layers, the first and second hidden layer will be created. The number of layers is determined through trial and error. Typically, a larger dataset will require more layers or more neurons, or vice versa. Finally, the output layer will be created with only one neuron since we are expecting only one output or result.

All of these will then be compiled using the adam optimiser, once again decided through trial and error to see which works the best, the mean squared error loss function which typically works best under regression, and the mean absolute error metric so we can plot it into a graph and see the convergence of the validation tests.

```
#Initialising ANN
ann = tf.keras.models.Sequential()

#Adding first and second hidden layer
ann.add(tf.keras.layers.Dense(units=128,activation="relu"))
ann.add(tf.keras.layers.Dense(units=64,activation="relu"))

#Adding output layer
ann.add(tf.keras.layers.Dense(units=1,activation="linear"))

#Compile mae
ann.compile(optimizer="adam",loss="mean_squared_error",metrics=['mean_absolute_error'])

es = keras.callbacks.EarlyStopping(monitor='val_loss', mode='min', patience=10, restore_best_weights=True)

history1 = ann.fit(X_test, y_test, callbacks=[es], epochs=10, batch_size=10, shuffle=True, validation_split=0.2, verbose=1)
```

Figure 13.2.3 Neural Network Model Building.

The number of epochs chosen for this model is only 10 for investigative reasons. We wish to compare the results with an improved version of this model hence we chose a decent number to prevent taking too much time to run the code. We could also experiment with a huge number of epochs while using Early Stopping so that the model will be fitted once the best weights are found.

13.2.4. Evaluation

```
#plot
loss = history1.history['loss']
val_loss = history1.history['val_loss']
epochs = range(1,len(loss)+1)
plt.plot(epochs, loss, label='Training Loss')
plt.plot(epochs, val_loss, 'r', label="Validation loss")
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

acc = history1.history['mean_absolute_error']
val_acc = history1.history['val_mean_absolute_error']
plt.plot(epochs, acc, 'y', label='Training MAE')
plt.plot(epochs, val_acc, 'r', label='Validation MAE')
plt.title('Training and validation MAE')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

Figure 13.2.4.1 Code for plotting line graph.

The model is evaluated using the mean absolute error metric plotted into a graph. The first graph plots the loss function and how it decreases after a number of epochs. The second graph will show the convergence of the training and validation MAE which signifies the reduction in errors as the epochs increase.

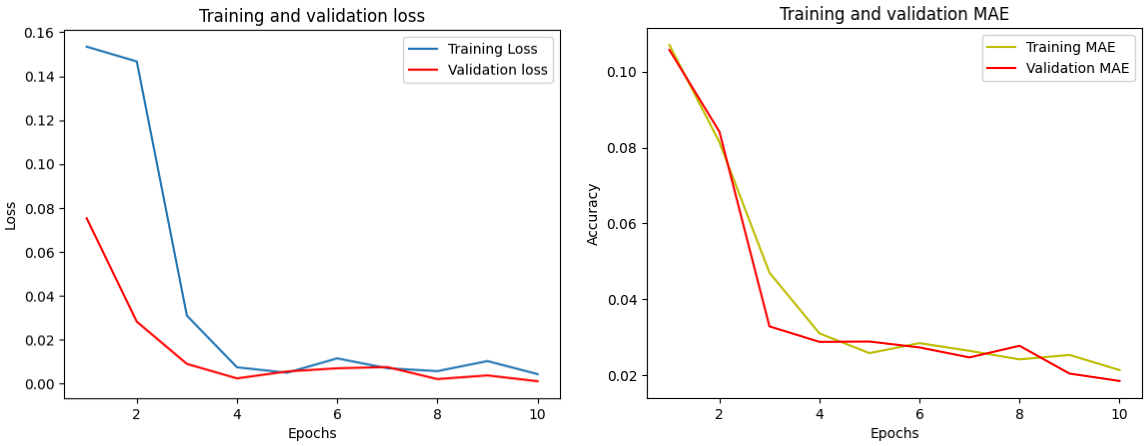


Figure 13.2.4.2 Training and Validation Loss and MAE Line Graph.

Looking at the graph, we can see that the validation and training loss and mae have not completely converged by the 10th epoch. But overall, the model converges well.

```
#predict on test data
predictions = ann.predict(X_test[:5])
print("Predicted values are: ", predictions)
print("Real values are: ", y_test[:5])
mae = np.mean(np.abs(predictions - y_test.values))
print("mae: ", mae)
```

Figure 13.2.4.3 Printing predicted values, real values, and MAE.

```
Predicted values are:  [[ 0.04079099]
 [ 0.02043917]
 [-0.0330502 ]
 [ 0.04994342]
 [ 0.00441722]]
Real values are:  2040     0.052617
22440    0.037930
10477    0.053034
46386    0.059507
17833    0.017337
Name: Fuel Used, dtype: float64
mae:  0.04164584716821207
```

Figure 13.2.4.4 Predicted Values, Real values, and MAE.

For the final result, we can compare the values predicted by the algorithm and the real values. The difference isn't quite large as we can tell from the small value of 0.0416.

13.2.5. Improvement

To further improve the model, we implemented a random search function to fine-tune the parameters grid, making the model find its own most optimal parameters. The random search in machine learning involves generating a random input over a number of iterations and plugging it into the function. The result will then be kept and stored to be compared with the results gained from the other iterations.

13.2.6 Applying Optimisation Algorithm

```
#Optimised code with random search
#Neural Network (NN) Algorithm
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from sklearn.preprocessing import LabelEncoder
from keras.utils import to_categorical
from sklearn.metrics import mean_squared_error
import random

from keras.models import Sequential
from keras import layers
from keras.layers import Dense, Dropout
from keras.callbacks import EarlyStopping

#Performing Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

def create_model(hidden_layers, learning_rate):
    model = keras.Sequential()
    model.add(layers.Dense(units=hidden_layers[0], activation='relu', input_dim=X_train.shape[1]))
    for units in hidden_layers[1:]:
        model.add(layers.Dense(units=units, activation='relu'))
    model.add(layers.Dense(3, activation='linear'))
    optimizer = keras.optimizers.Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss='mean_squared_error', metrics=['mean_absolute_error'])
    return model
```

Figure 13.2.6.1 Initial Model Creation with Randomised Parameters.

Similar to the original model, all the necessary libraries are imported and the training and testing set are scaled. Following that, a “create_model” function is created with “hidden_layers” and “learning_rate” as the parameters we are accounting for. In the function, the first input layer is created, only that the next layer or layers will be added if it meets the condition of the random search, by the “for loop” accordingly.

The optimiser used is still the adam optimiser, however, the learning rate will be dependent on the random search. Meanwhile, the loss function is still calculated with the “mean_squared_error” and the metric being “mean_absolute_error”.

```python
# Define hyperparameter search space
param_grid = {
    'hidden_layers': [(32,), (64,), (32,32), (64,64)],
    'learning_rate': [0.001, 0.01, 0.1]
}

# Perform random search
best_mae = float('inf')
best_params = None
##best_mse = float('inf')
for _ in range(10):  # Number of random searches
    params = {k: random.choice(v) for k, v in param_grid.items()}
    model = create_model(params['hidden_layers'], params['learning_rate'])
    es = keras.callbacks.EarlyStopping(monitor='val_loss', mode='min', patience=10, restore_best_weights=True)
    history = model.fit(X_test, y_test, callbacks=[es], epochs=10, batch_size=10, shuffle=True, validation_split=0.2, verbose=1)

    predictions = model.predict(X_test)
    y_test_reshaped = np.tile(y_test.values, (3, 1)).T

    mse = mean_squared_error(y_test_reshaped, predictions)
    mae = np.mean(np.abs(predictions - y_test_reshaped))
    if mae < best_mae:
        best_mae = mae
        best_params = params
        ##best_mse = mse

    print()

print("Best MAE:", best_mae)
##print("Best MSE:", best_mse)
print("Best parameters:", best_params)
```

Figure 13.2.6.2 Algorithm for the application of Random Search.

The hyperparameter search space is defined first so that the random search function doesn't search blindly which may make the training process a whole lot slower. Then the random search is performed. Here, the number of iterations shouldn't be too high, otherwise the data will be too generalised and the model will start memorising the data instead of learning from it. This will result in a reduction in accuracy. Completing the algorithm, the model will be fitted with the "best_mae" returned in order for the model to learn which parameters are the most best suited.

```
Best MAE: 0.01906809974512391
Best parameters: {'hidden_layers': (32, 32), 'learning_rate': 0.1}
```

Figure 13.2.6.3 Results of Random Search.

The figure above shows how the best MAE is close to nil as well as which hidden layers and learning rate parameters are kept. Compared to the original algorithm without the random search, the MAE is smaller so we can expect a closer result of the predicted values compared with the real values.

```python
#plot
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()

plt.plot(history.history['mean_absolute_error'], label='Training MAE')
plt.plot(history.history['val_mean_absolute_error'], label='Validation MAE')
plt.title('Training and validation MAE')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

Figure 13.2.6.4 Code for plotting line graph.

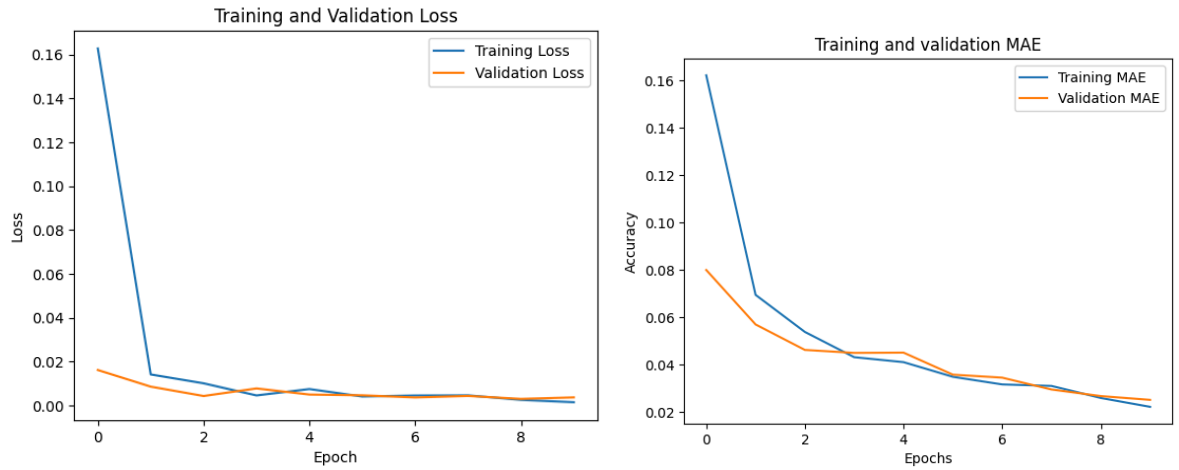


Figure 13.2.6.5 Training and Validation Loss and MAE Line Graph.

The figure above depicts the convergences of the training and validation loss and MAE. Different from the original model, the model converges faster, finishing the fitting by the 8th epoch.

```
#predict on test data
predictions = model.predict(X_test[:5])
print("Predicted values are: ", predictions)
print("Real values are: ", y_test.values[:5])
print("MAE: ", best_mae)
```

Figure 13.2.6.6 Printing predicted values, real values, and MAE.

```
Predicted values are:  [[0.06492357]
 [0.0233904 ]
 [0.0799921 ]
 [0.02979262]
 [0.04588769]]
Real values are:  [0.05261722 0.03792983 0.05303362 0.05950667 0.01733719]
MAE:  0.01906809974512391
```

Figure 13.2.6.7 Predicted values, Real values, and MAE.

Finally, to sum the result, the MAE for the predicted values and the real values is calculated to be 0.01906, smaller than the previous model by 2 times.

13.2.7 Benefits and Limitations

Neural networks are often preferred to deal with large datasets with complex and possible underlying relationships. This is due to the fact that the algorithm works by creating layers to make decisions, simulating a human brain. Neural networks are able to learn from previous mistakes to improve their accuracy over time, having different weights and biases for each neuron.

However, a common problem found when implementing this algorithm is how it is quite difficult to interpret the output (Donges, 2023). Neural networks come into a decision output based on what the computers learn from the dataset. For example, the model may predict an image of a computer to a chair, and we would not have any idea how the model came to that conclusion. This issue is applicable in some domains where interpretability is critical (Donges, 2023), like making business decisions. Neural Networks may also take a long time

to develop at a more professional setting, unlike the example given above which used imported libraries like Keras. This does not allow for flexibility thus the lack of control of the algorithm. Lastly, this algorithm does not do well with little data most of the time, which causes overfitting. The fundamentals of neural networks is that layers are stacked on top of each other, each layer with hidden features where complex aspects of the data are learned. Dealing with small data won't allow the algorithm to understand the underlying relationships present and the model generally won't perform well with new unseen data. It has a higher chance of memorising the training data, including noise and outliers, instead of learning it.

14. Difficulties, Risks Identified, and Strategies

When the team first started the project, it was quite difficult to find any related dataset that could be useful for this project. The team tried to look through all the past papers on this project to see if they provided the dataset or stated where they obtained it. Besides the articles/papers, the search for the dataset also goes through official dataset collection websites, including Google Dataset, the Australian Bureau Of Statistics, and the Australia Data Archive. However, nothing was found.

The team tried to contact the authors of the articles, but they have not responded yet. As a result, the search was expanded for third-party data collection, and the needed dataset was finally obtained. Despite that, the dataset was quite messy, attributes have different formats, repeated columns under different naming conventions, a lot of missing data and unclear attributes' descriptions/dictionaries. This made it difficult to do the dataset's initial analysis in the project's first phase.

A correlation map for similar columns to find correlations between attributes. If 1 was the correlation value between attributes with the same measure, then it is likely to have the same value. This means we remove one of the columns.

The percentage of NULL values is calculated for each column. A high percentage column will not be used due to its unreliable nature. During the analysis, an occurrence pattern of NULL values was derived. The NULL value appears simultaneously across all the columns that have 52.29%. Further exploration also gave an insight into the NULL value appearance depending on the Cycle TYPE, specifically AuxMobile and Loading Cycle.

Furthermore, the Excel files supplemented by the source had major header misplacements that created an obstacle to interpreting data. Therefore, the team decided to use the CSV files supplemented instead.

During initial data analysis, due to the sheer number of attributes in the cycle dataset, it took the team quite a while to understand the data. Additional team meetings must occur to get a unanimous opinion on which attributes the team will use for the initial data analysis, preprocessing, and modelling.

Additionally, when importing the data into PowerBI and Tableau, several attributes were in an incorrect format e.g. fuel data in text format instead of decimal numbers. This means the

team had to do some preprocessing before importing the data, which slowed down the initial analysis of the datasets. SQL and DAX expressions in PowerBI assist the team in this matter.

During the implementation of the XGBoost algorithm, the large number of attributes in the dataset caused several issues. The column containing the names of trucks being used in particular caused an issue in building the regression model. The initial data type of the iMine Load FCTR Truck column was recognised as Object, this caused several issues as the regression model could not accept Objects as one of the variables.

This issue is solved by changing the datatype of the intended column to int. This is done with the help of label encoder, label encoders make it so that each unique value in a column is replaced with a certain number, effectively removing the issue of datatypes.

15. Deployment

Deployment stage comes after the development of the machine learning models. Although this stage is not in Boosting's scope of the project, the team has decided to include recommendations on future deployment of the models which can be integrated with Rio Tinto's business information systems.

After the development of machine learning models, Rio Tinto's IT team needs to perform initial planning on the integration of the models into the system. After then, the ML models can be used to assist in taking executive decisions. In terms of deployment method, Boosting recommends the use of Edge deployment method. Edge deployment method allows the deployment of the ML models on the user's device directly such as on a smartphone. It allows onsite employees to access the model and make predictions on how much fuel will be used which can assist onsite decisions. In order to apply this deployment method, the size of models needs to be small in storage and requires minimal computing power. The overview can be seen on *Figure 15*.

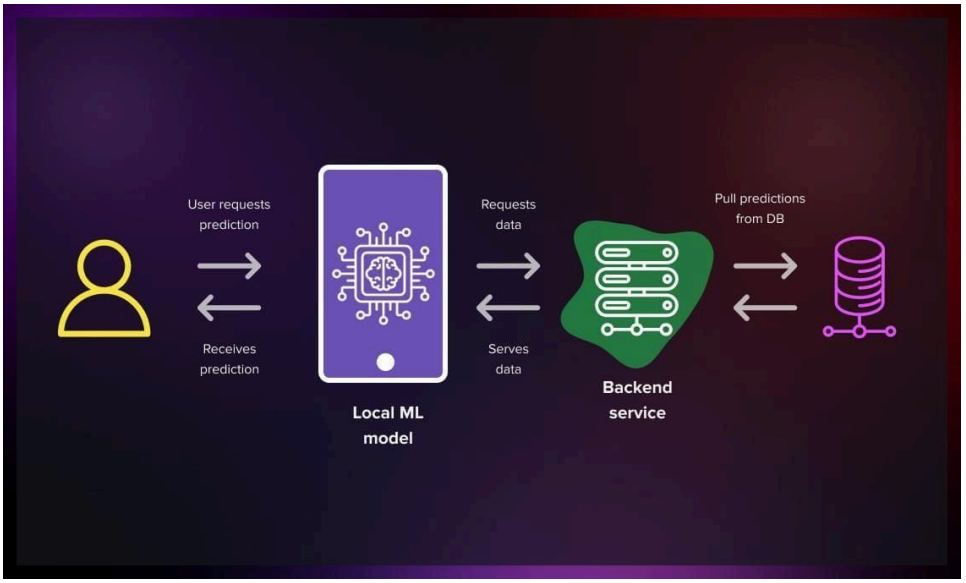


Figure 15. Edge Deployment Overview (Serokell Software Development Company, 2023).

A deployment environment needs to be first selected that will be suitable for the machine learning models. This could be cloud platforms such as AWS, Azure, and Google Cloud our it could be an on-premises infrastructure. Scalability, cost , security, and integration needs to be considered for the chosen infrastructure. Boosting recommends that Rio Tinto integrate the ML models into on-premises servers to ensure security since sensitive data might be handled. Database infrastructure also needs to be integrated such as SQL server. Next, the machine learning models should be contained inside a container such as a Docker container to encapsulate all elements together. Containers are easy to update and modify, making it an essential solution to use since it can reduce the risk downtime and easier model maintenance (Serokell Software Development Company, 2023).

After deployed, Rio Tinto's IT team needs to consistently monitor the performance of the machine learning models. Logging and alerts need to be integrated to assist troubleshooting. The deployment process should also be automated using, a CI/CD pipeline that guarantees seamless model updates (Serokell Software Development Company, 2023). Additionally, it also improves the deployment workflow, automate the testing, version control, and deployment phases as well (Serokell Software Development Company, 2023).

In terms of post-deployment stage, the models has to be improved overtime especially when new attributes come in. Additionally, a scaling strategy must be put in place in order to accommodate changes in demand and workload.

16. Conclusion

The project undertaken by the BOOSTING team demonstrates a successful application of XGBoost algorithm and other machine learning algorithms including random forest and neural network to predict fuel consumption of open-pit mine trucks with high degree of accuracy. However, the XGboost model produced the highest accuracy as evidenced by the performance metrics. This not only showcases the model's efficiency and effectiveness when handling such complex and large datasets consisting of different features, but also its application in the future of mining operation.

Furthermore, the project highlights the major aspect of data quality and preprocessing in building machine learning models. Due to the quality of the original data, in which a lot of inconsistencies in the naming and attributes description, irrelevant attributes, wrong data types and high volume of NULL value throughout the primary dataset, it was a long and time-consuming process of data cleaning, feature engineering, pattern recognition and exploratory data analysis. Through the process, the team was able to learn the dataset and detect findings that assisted in preprocessing which significantly improved the model's learning capability. The steps the team took in preprocessing included handling missing values, encoding categorical variables, and normalising data. These are crucial steps to ensure the model's reliability, validity and usability.

The results of this project also demonstrate the technical feasibility of using machine learning models in real-world applications, specifically in the decision making in the mining industry. By accurately predicting the fuel consumption, mine operators can better plan and allocate resources. As a result, minimise wasting resources such as crude oil, enhance operational efficiency and decrease harmful emission. While fuel is still the main source of energy generation for machinery, in this case, open-pit mining trucks, it is important to reduce their carbon footprint by using the model's prediction. This will contribute to environmental sustainability goals.

This project serves as a benchmark for the integration of machine learning into industrial applications, providing an optimistic future and benefits in terms of cost reduction, operational efficiency and protecting the environment. The project itself can be the initial research for further implementation across various industries.

17. Future Work

The success of using XGBoost algorithm in building fuel consumption prediction for open-pit mining trucks opens several opportunities for future research and development. There are vast of different ways of expanding and enhancing this project considering the rapid growth of new technologies and need for efficiency and sustainability.

One of the work Boosting is planning to do is enhancing the machine learning models by using a different optimisation algorithm and experience with an extensive range of machine learning such as deep learning. This will expand the project further which may offer maximum accuracy and efficiency in the model's performance.

In our project, most environmental related data such as locations, weather, timestamp and operational data such as hauling, loading were removed due to its high percentage of NULL value and incomplete data collection. This additional data would allow the model to take a wider range of variables and factors into account and evaluate the effect of them on fuel consumption. As a result, incorporating data on these factor can further complete and strengthen the model's prediction power, thus improving its accuracy and reliability.

After enhancing the model's capability using various methods, the model can be developed into a real-time prediction system, where it provides ongoing insights and predictions during the mining operation. This will allow operators to perform adjustments on the spot, optimising fuel usage and make plans for future operation.

One of other potential future works could be using this model in different industries such as logistic, agriculture or construction where heavy machinery/trucks are also used in the operation. Exploring the transferability of the model to different fields will extend its applications and maximising the impact of this project.

18. Updated Project Plan

The old Gantt chart the team used to guide our project has been reviewed. The main changes revolve around the Modelling and Evaluation Phase (Phase 4 and 5). The modelling phase was originally planned to start in weeks 3-4 and continue in week 7, but now, the team has decided to start the phase in week 6. This is due to the client's request to build a simple model in the mid-project update report instead of the first impression that a simple model must be included in the proposal report. The same goes for the evaluation phase, the client wants the simple model to be somewhat improved hence, the evaluation phase has to start earlier. Another change was made to the presentation date, which was postponed until after mid-semester Stuvac.

For full access to the updated Gantt chart, follow this GanttChart link.

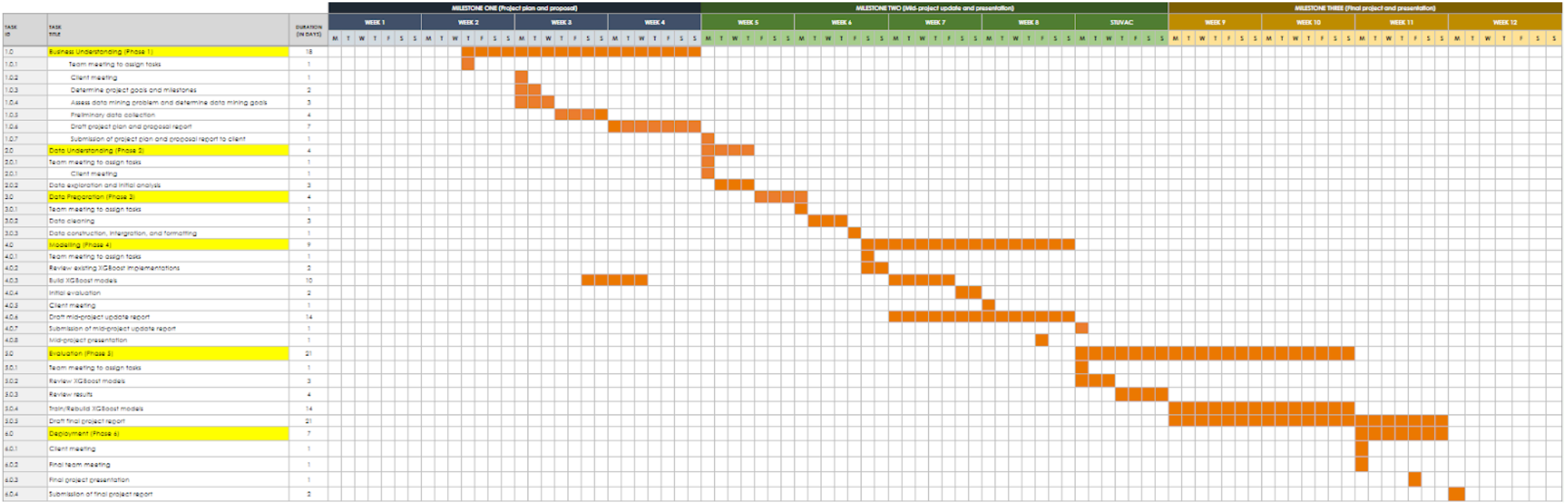


Figure 18.1 Old Gantt Chart.

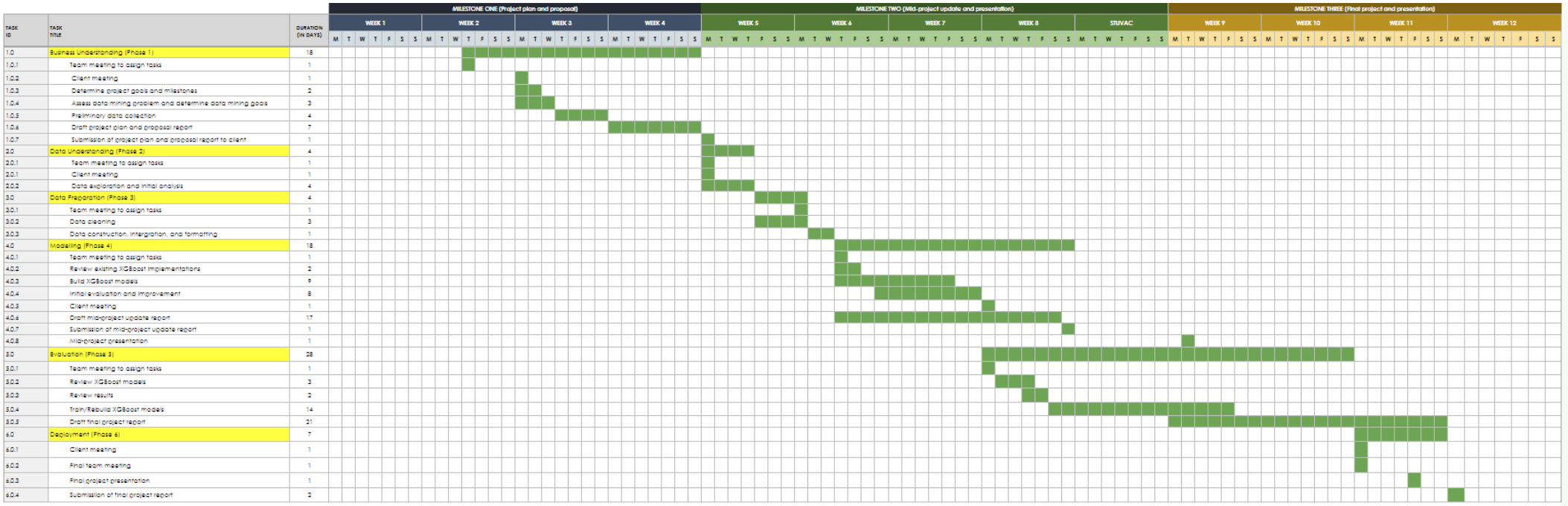


Figure 18.2 Updated Gantt Chart

Figure 18.1 shows the old Gantt Chart the team used previously, and *Figure 18.2* shows the updated Gantt Chart the team will use. The structure of the phases is maintained, only the dates change for several phases. The team still plan to maintain the CRISP-DM methodology for the rest of the project.

19. Appendix

Google Colab Link: AI Capstone.ipynb DOE.ipynb
Random Forest in AI Capstone.ipynb AI capstone - marv code.ipynb

GitHub Repo:

https://github.com/valerieeesss/AI-Capstone-Group-7B.git

PowerBI:

https://studentutsedu.sharepoint.com/:u:/r/sites/Group7B/Shared%20Documents/General/truck.pbix?csf=1&web=1&e=giTIba

Dataset Drive:
https://drive.google.com/drive/folders/1_Ict5VmMmOwlnt8I6WJD8x5xWh0k-ZuH?usp=sharing

DESIGN OF EXPERIMENT----------

```
import pandas as pd

# Load dataset

data = pd.read_csv('/content/drive/MyDrive/CycleData.csv')

# Calculate the percentage of NULL values in each column

null_percentage = data.isnull().mean()

# Filter out columns where the null percentage is greater than 60%

columns_to_keep = null_percentage[null_percentage < 0.6].index.tolist()

# Keep only the columns that have less than 60% NULL values

data = data[columns_to_keep]

# Filter to include only rows where the cycle type is 'TruckCycle'

# Ensure that the 'Cycle Type' column was not removed due to NULL values before filtering rows

if 'Cycle Type' in data.columns:

    data = data[data['Cycle Type'] == 'TruckCycle']

else:

    print("Filtered dataset does not include 'Cycle Type' due to high null values or it does not exist.")

# Save the filtered data to a new CSV file
```

```
data.to_csv('FilteredData_TruckCycle.csv', index=False)

# Print a message about the completion of the filtering

print("Data filtering complete. Here is the snippet of the filtered dataset:")

print(data.head())

# Load the dataset

file_path = '/content/FilteredData_TruckCycle.csv'

data = pd.read_csv(file_path)

# Initial filtered attributes focusing on key aspects for fuel consumption prediction

filtered_data = data[['Fuel Used', 'COMPLETEDCYCLEDURATION', 'Payload (kg)',
'Autonomous',

                      'Available SMU Time', 'iMine Operating Hours', 'AT Available Time (iMine)',

                      'Travelling Empty Duration', 'Travelling Full Duration', 'Idle Duration', 'Primary
Machine Class Name']]

# One-hot encode the truck models

filtered_data = pd.get_dummies(filtered_data, columns=['Primary Machine Class Name'])

# Display the first few rows of the filtered dataset to verify

print(filtered_data.head())

# Now the dataset includes additional columns for each truck model, which can be used in
regression analysis.

import pandas as pd

import statsmodels.api as sm

import numpy as np

# Load the dataset

file_path = '/content/FilteredData_TruckCycle.csv'

data = pd.read_csv(file_path)

# One-hot encode the truck models

data = pd.get_dummies(data, columns=['Primary Machine Class Name'])

# Convert all columns to float, coercing errors and filling NaNs

for column in data.columns:

    data[column] = pd.to_numeric(data[column], errors='coerce')
```

```
data.fillna(data.mean(), inplace=True)

# Convert boolean columns to integer to avoid any Statsmodels issues

data['Primary Machine Class Name_CAT 793F CMD'] = data['Primary Machine Class Name_CAT 793F CMD'].astype(int)

data['Primary Machine Class Name_Cat 789C'] = data['Primary Machine Class Name_Cat 789C'].astype(int)

# Ensure no infinite values exist

data.replace([np.inf, -np.inf], np.nan, inplace=True)

data.fillna(data.mean(), inplace=True)

# Define predictors and add a constant for intercept

predictors = ['COMPLETEDCYCLEDURATION', 'Payload (kg)', 'Autonomous', 'Available SMU Time',

        'iMine Operating Hours', 'AT Available Time (iMine)', 'Travelling Empty Duration',

        'Travelling Full Duration', 'Idle Duration',

        'Primary Machine Class Name_CAT 793F CMD', 'Primary Machine Class Name_Cat 789C']

X = sm.add_constant(data[predictors])  # Adding a constant for the intercept

# Dependent variable

y = data['Fuel Used']

# Fit the regression model

model = sm.OLS(y, X).fit()

# Print out the summary of the regression model

print(model.summary())

# Compute the correlation matrix

correlation_matrix = X.iloc[:, 1:].corr()  # Exclude the constant

print(correlation_matrix)

# You might also use variance inflation factor (VIF) to quantify multicollinearity

from statsmodels.stats.outliers_influence import variance_inflation_factor

vif_data = pd.DataFrame()

vif_data['feature'] = X.columns[1:]  # Exclude the intercept

vif_data['VIF'] = [variance_inflation_factor(X.values, i+1) for i in range(len(X.columns[1:]))]
```

```
print(vif_data)
import pandas as pd
import numpy as np
import statsmodels.api as sm
# Define predictors, excluding 'Autonomous', and add a constant for intercept
predictors = ['COMPLETEDCYCLEDURATION', 'Payload (kg)', 'Available SMU Time',
        'iMine Operating Hours', 'AT Available Time (iMine)', 'Travelling Empty Duration',
        'Travelling Full Duration', 'Idle Duration',
        'Primary Machine Class Name_CAT 793F CMD', 'Primary Machine Class
Name_Cat 789C']
X = sm.add_constant(data[predictors])
# Fit the regression model
model = sm.OLS(y, X).fit()
# Print out the summary of the regression model
print(model.summary())
# Recalculate VIFs to check improvements
vif_data = pd.DataFrame()
vif_data['feature'] = X.columns[1:]  # Exclude the intercept
vif_data['VIF'] = [variance_inflation_factor(X.values, i+1) for i in range(len(X.columns[1:]))]
print(vif_data)
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
# Splitting the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Fitting the model on the training data
model_train = sm.OLS(y_train, X_train).fit()
# Predicting on the test data
y_pred = model_train.predict(X_test)
# Calculating the RMSE
```

```
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error:", rmse)
import matplotlib.pyplot as plt
import seaborn as sns
# Predictions and residuals
y_train_pred = model_train.predict(X_train)
residuals = y_train - y_train_pred
# Plotting residuals
plt.figure(figsize=(10, 6))
plt.scatter(y_train_pred, residuals)
plt.axhline(y=0, color='red', linestyle='--')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Residuals vs Predicted Values')
plt.show()
# Histogram of the residuals
plt.figure(figsize=(10, 6))
sns.histplot(residuals, kde=True)
plt.title('Distribution of Residuals')
plt.xlabel('Residuals')
plt.show()
# Q-Q plot for normality check
import scipy.stats as stats
plt.figure(figsize=(10, 6))
stats.probplot(residuals, dist="norm", plot=plt)
plt.title('Q-Q Plot of Residuals')
plt.show()
from statsmodels.stats.outliers_influence import OLSInfluence
# Leverage and influence plot
```

```
influence = OLSInfluence(model_train)
fig, ax = plt.subplots(figsize=(10, 6))
influence.plot_influence(ax=ax)
plt.title('Influence Plot')
plt.show()

# Cook's distance to identify potential outliers
fig, ax = plt.subplots(figsize=(10, 6))
influence.plot_index(y_var="cooks", threshold=4 / len(X_train), ax=ax)
plt.title('Cook\'s Distance')
plt.show()
```

XGBOOST----------

```
"Original file is located at"
"https://colab.research.google.com/drive/1qO7BojMX_3Zlz1JnwXKZtD-zvsMzde_B"

from google.colab import drive
drive.mount('/content/drive')

# Commented out IPython magic to ensure Python compatibility.
import pandas as pd
import pylab as pl
import numpy as np
import scipy.optimize as opt
from sklearn import preprocessing
# %matplotlib inline
import matplotlib.pyplot as plt
```

```
from sklearn.model_selection import train_test_split,
RandomizedSearchCV

from sklearn.metrics import mean_absolute_error, mean_squared_error

from xgboost import XGBRegressor


df_cycle = pd.read_csv('/content/drive/MyDrive/Colab
Notebooks/CycleData.csv')


# Check for missing values

print("\n\nMissing Values of CycleData:\n")

print(df_cycle.isnull().sum())


# Remove missing values

df_cycle = df_cycle.dropna(subset=['Fuel Used'],axis=0)

df_cycle = df_cycle.dropna(axis=1)


from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder


X = df_cycle.drop('Fuel Used', axis=1)  # Features

y = df_cycle['Fuel Used']  # Target variable

# Take the object var only and change to int type

object_columns = X.select_dtypes(include=['object']).columns

label_encoders = {}

for col in object_columns:

    label_encoders[col] = LabelEncoder()

    X[col] = label_encoders[col].fit_transform(X[col])

#print(X.dtypes)

# Split the data into training and testing sets
```

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


print("X_train shape:", X_train.shape)

print("X_test shape:", X_test.shape)

print("y_train shape:", y_train.shape)

print("y_test shape:", y_test.shape)

print(y.head())

print(X_train.head())


xgb_regressor = XGBRegressor(objective='reg:squarederror',
eval_metric='rmse', random_state=42)

xgb_regressor.fit(X_train, y_train)

# Predict

y_pred = xgb_regressor.predict(X_test)

# Evaluate the model

mse = mean_squared_error(y_test, y_pred)

print("Mean Squared Error:", mse)


mae = mean_absolute_error(y_test, y_pred)

print("Mean Absolute Error:", mae)


rmse = mean_squared_error(y_test, y_pred, squared=False)

print("Root Mean Squared Error:", rmse)




import matplotlib.pyplot as plt

import seaborn as sns

from scipy.stats import probplot


```

```
# Calculate residuals
residuals = y_test - y_pred


# Residual Plot
plt.figure(figsize=(10, 6))
plt.scatter(y_pred, residuals, alpha=0.5)
plt.axhline(y=0, color='r', linestyle='--')
plt.title('Residual Plot')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.show()


# Distribution of Errors
plt.figure(figsize=(10, 6))
sns.histplot(residuals, kde=True)
plt.title('Distribution of Errors')
plt.xlabel('Errors')
plt.ylabel('Density')
plt.show()


#improvements
mse_percent= (mse - mse_best) / mse * 100
print("MSE improvement: {:.2f}%".format(mse_percent))


#parameter search spacew
param_grid = {
    'n_estimators': [100, 200, 300, 400, 500],
    'learning_rate': [0.01, 0.05, 0.1, 0.2, 0.3],
    'max_depth': [3, 4, 5, 6, 7],
```

```
    'min_child_weight': [1, 2, 3, 4, 5],
    'gamma': [0, 0.1, 0.2, 0.3, 0.4],
    'subsample': [0.6, 0.7, 0.8, 0.9, 1.0],
    'colsample_bytree': [0.6, 0.7, 0.8, 0.9, 1.0],
    'reg_alpha': [0, 0.1, 0.5, 1, 2],
    'reg_lambda': [0, 0.1, 0.5, 1, 2]
}
#Monte Carlo Simulation Parameters
num_simulations = 5
sample_size = 0.8


results = []
for i in range(num_simulations):
    X_sample, _, y_sample, _ = train_test_split(X, y,
train_size=sample_size, random_state=i)
    xgb_regressor = XGBRegressor(objective='reg:squarederror',
eval_metric='rmse', random_state=42)
    #Random Search
    random_search = RandomizedSearchCV(estimator=xgb_regressor,
param_distributions=param_grid, n_iter=50,

scoring='neg_mean_squared_error', cv=5, verbose=2, random_state=42,
n_jobs=-1)
    random_search.fit(X_sample, y_sample)


    #Append Results
    results.append({
        'best_params': random_search.best_params_,
        'best_score': random_search.best_score_
    })
#convert results to dataframe
```

```
results_df = pd.DataFrame(results)


print("Mean best score:", results_df['best_score'].mean())
print("Standard deviation of best scores:",
results_df['best_score'].std())
print("Best hyperparameters:")
print(results_df['best_params'].mode())


# Initialize and train the XGBoost with optimized hyper parameters
best_params = {
    'subsample': 0.7,
    'reg_lambda': 0.5,
    'reg_alpha': 0.1,
    'n_estimators': 500,
    'min_child_weight': 4,
    'max_depth': 4,
    'learning_rate': 0.3,
    'gamma': 0,
    'colsample_bytree': 0.9
}
# Initialize XGBRegressor with best hyperparameters
xgb_regressor_best = XGBRegressor(objective='reg:squarederror',
eval_metric='rmse', random_state=42, **best_params)
xgb_regressor_best.fit(X_train, y_train)
# Predict
y_pred = xgb_regressor_best.predict(X_test)
# Evaluate the model
mse_best = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse_best)
```

```
mae_best = mean_absolute_error(y_test, y_pred)

print("Mean Absolute Error:", mae_best)


rmse_best = mean_squared_error(y_test, y_pred, squared=False)

print("Root Mean Squared Error:", rmse_best)




import matplotlib.pyplot as plt

import seaborn as sns

from scipy.stats import probplot


# Calculate residuals

residuals = y_test - y_pred


# Residual Plot

plt.figure(figsize=(10, 6))

plt.scatter(y_pred, residuals, alpha=0.5)

plt.axhline(y=0, color='r', linestyle='--')

plt.title('Residual Plot')

plt.xlabel('Predicted Values')

plt.ylabel('Residuals')

plt.show()


# Distribution of Errors

plt.figure(figsize=(10, 6))

sns.histplot(residuals, kde=True)

plt.title('Distribution of Errors')

plt.xlabel('Errors')

plt.ylabel('Density')
```

```
plt.show()


model_best_params = results_df['best_params'].mode().iloc[0]
for key, value in model_best_params.items():
    print(f"{key}: {value}")


#k-cross validation


from sklearn.model_selection import KFold, cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score


# Define the number of folds for cross-validation
k = 5


# Initialize the cross-validation splitter
kf = KFold(n_splits=k, shuffle=True, random_state=42)


# Initialize your regression model (e.g., LinearRegression)
model = LinearRegression()


# Perform k-fold cross-validation and calculate regression metrics
mse_scores = -cross_val_score(xgb_regressor, X_train, y_train, cv=kf,
scoring='neg_mean_squared_error')
r2_scores = cross_val_score(xgb_regressor, X_train, y_train, cv=kf,
scoring='r2')


print("Mean Squared Error (MSE) Scores for Each Fold:", mse_scores)
print("R-squared Scores for Each Fold:", r2_scores)
average_mse = mse_scores.mean()
```

```
print("Average Mean Squared Error (MSE):", average_mse)

average_r2 = r2_scores.mean()

print("Average R-squared:", average_r2)

#compare

mse_percent2= (average_mse - mse_best) / average_mse * 100

print("MSE improvement: {:.2f}%".format(mse_percent2))
```

RANDOM FOREST----------

from google.colab import drive

drive.mount('/content/drive')

import pandas as pd

import pylab as pl

import numpy as np

import scipy.optimize as opt

from sklearn import preprocessing

%matplotlib inline

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_absolute_error, mean_squared_error

import seaborn as sns

df_delay= pd.read_csv('/content/drive/MyDrive/AI Capstone Dataset/DelayData.csv')

df_cycle= pd.read_csv('/content/drive/MyDrive/AI Capstone Dataset/CycleData.csv')

df_location= pd.read_csv('/content/drive/MyDrive/AI Capstone Dataset/LocationData.csv')

Check for missing values

print("\n\nMissing Values of CycleData:\n")

print(df_cycle.isnull().sum())

```
print("\n\nMissing Values of DelayData:\n")
print(df_delay.isnull().sum())
print("\n\nMissing Values of LocationData:\n")
print(df_location.isnull().sum())

# Remove missing values
df_cycle = df_cycle.dropna(subset=['Fuel Used'],axis=0)
df_cycle = df_cycle.dropna(axis=1)
df_delay = df_delay.dropna(axis=1)
df_location = df_location.dropna(axis=1)

import matplotlib.pyplot as plt
# Check for missing values
print("\n\nMissing Values of CycleData:\n")
print(df_cycle.isnull().sum())
print("\n\nMissing Values of DelayData:\n")
print(df_delay.isnull().sum())
print("\n\nMissing Values of LocationData:\n")
print(df_location.isnull().sum())

# Check first rows on dataframe
print(df_cycle.head())
print(df_delay.head())
print(df_location.head())

##OEE Calculation
oee = df_cycle[['AT Available Time (iMine)', 'Down Time', 'iMine Operating Hours',
'OPERATINGTIME (CAT)', 'Idle Duration']]
oee['Availability'] = ((oee['AT Available Time (iMine)'] - oee['Down Time']) / oee['AT
Available Time (iMine)']) * 100
```

```
oee['Performance'] = ((oee['OPERATINGTIME (CAT)'] - oee['Idle Duration']) / oee['OPERATINGTIME (CAT)']) * 100

oee['Quality'] = ((oee['iMine Operating Hours'] - oee['Down Time']) / (oee['Down Time'] + oee['Idle Duration'])) * 100

oee['oee'] = oee['Availability'] * oee['Performance'] * oee['Quality']


##OEE Calculations

def calculate_oee():

   return oee[['AT Available Time (iMine)', 'Down Time', 'iMine Operating Hours', 'OPERATINGTIME (CAT)',

               'Idle Duration', 'Availability', 'Performance', 'Quality', 'oee']]


##Print OEE Calculations

print(calculate_oee())


from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder


X = df_cycle.drop('Fuel Used', axis=1)  # Features

y = df_cycle['Fuel Used']  # Target variable

# Take the object var only and change to int type

object_columns = X.select_dtypes(include=['object']).columns

label_encoders = {}

for col in object_columns:

   label_encoders[col] = LabelEncoder()

   X[col] = label_encoders[col].fit_transform(X[col])

#print(X.dtypes)

# Split the data into training and testing sets

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

```
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
print(y.head())

from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

# Initialize the Random Forest regressor

rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)
rf_regressor.fit(X_train, y_train)

# Make predictions on the test data
predictions = rf_regressor.predict(X_test)

# Calculate accuracy
mse = mean_squared_error(y_test, predictions)
print("Mean Squared Error:", mse)

from sklearn.linear_model import LinearRegression
# X_train, X_test, y_train, y_test are already defined so we train our model
model= LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)
```

```
#Create random samples of n =1000
sample_n = np.random.choice(len(y_test), size=1000, replace=False)

# Visualize actual vs predicted values

plt.figure(figsize=(8, 6))
plt.scatter(y_test.iloc[sample_n], y_pred[sample_n], color='blue')  # Use sampled data points
plt.plot(y_test, y_test, color='red', linestyle='--')  # Plotting the diagonal line for reference
plt.title('Actual vs Predicted Fuel Used')
plt.xlabel('Actual Fuel Used')
plt.ylabel('Predicted Fuel Used')
plt.show()

from sklearn.metrics import mean_squared_error, r2_score

# Calculate MSE
mse_linear = mean_squared_error(y_test, y_pred)

# Calculate R^2 score
r2_linear = r2_score(y_test, y_pred)

print("Linear Regression Model:")
print("MSE:", mse_linear)
print("R^2 Score:", r2_linear)

import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import RandomizedSearchCV, train_test_split
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

# Define a smaller parameter grid
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [5, 10],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
}

# Reduce the number of iterations
n_iterations = 5

# Define the sample size
sample_size = 500

mse_results = []
feature_importances = []

for iteration in range(n_iterations):
    # Randomly sample the data
    random_indices = np.random.choice(len(X), size=sample_size, replace=False)
    X_sampled = X.iloc[random_indices]
    y_sampled = y.iloc[random_indices]

    # Split the sampled data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X_sampled, y_sampled, test_size=0.2,
random_state=iteration)
```

```
    # Initialize the Random Forest regressor
    rf_regressor = RandomForestRegressor(random_state=42)

    # Initialize RandomizedSearchCV with a smaller number of iterations
    random_search = RandomizedSearchCV(estimator=rf_regressor,
param_distributions=param_grid,
                                       n_iter=4, cv=5, verbose=2, random_state=42, n_jobs=-1)

    # Fit the RandomizedSearchCV
    random_search.fit(X_train, y_train)

    # Get the best model
    best_rf_regressor = random_search.best_estimator_

    # Make predictions on the test data
    predictions = best_rf_regressor.predict(X_test)

    # Calculate mean squared error
    mse = mean_squared_error(y_test, predictions)
    mse_results.append(mse)

    print(f"Iteration {iteration + 1}/{n_iterations} - MSE: {mse}")

      # Get feature importances
    feature_importances.append(best_rf_regressor.feature_importances_)

# Print average MSE over all iterations
print("Average MSE:", np.mean(mse_results))
```

```
# Plot distribution of MSE
plt.figure(figsize=(8, 6))
sns.histplot(mse_results, kde=True, bins=5)
plt.title('Distribution of MSE')
plt.xlabel('MSE')
plt.ylabel('Frequency')
plt.show()

# Plot feature importances

mean_feature_importances = np.mean(feature_importances, axis=0)
sorted_indices = np.argsort(mean_feature_importances)[::-1][:5]  # Selecting only the top 5 features
plt.figure(figsize=(10, 6))
sns.barplot(x=mean_feature_importances[sorted_indices], y=X.columns[sorted_indices], palette="viridis")
plt.title('Top 5 Feature Importances')
plt.xlabel('Importance')
plt.ylabel('Features')
plt.show()

from sklearn.metrics import r2_score

# Calculate R^2 for each iteration
r_squared_results = []

for iteration in range(n_iterations):
    # Make predictions on the test data
    predictions = best_rf_regressor.predict(X_test)
```

```
    # Calculate R^2
    r_squared = r2_score(y_test, predictions)
    r_squared_results.append(r_squared)

    print(f"Iteration {iteration + 1}/{n_iterations} - R^2: {r_squared}")

# Print average R^2 over all iterations
print("Average R^2:", np.mean(r_squared_results))

# Print average MSE over all iterations
print("Average MSE:", np.mean(mse_results))
```

NEURAL NETWORK ----------

```
from google.colab import driveimport pandas as pd
import pylab as pl
import numpy as np
import scipy.optimize as opt
from sklearn import preprocessing
%matplotlib inline
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.metrics import mean_absolute_error, mean_squared_error
from xgboost import XGBRegressor

drive.mount('/content/drive')
df_cycle = pd.read_csv('/content/drive/MyDrive/AI Capstone Dataset/CycleData.csv')
# Check for missing values
print("\n\nMissing Values of CycleData:\n")
```

```
print(df_cycle.isnull().sum())


# Remove missing values
df_cycle = df_cycle.dropna(subset=['Fuel Used'],axis=0)
df_cycle = df_cycle.dropna(axis=1)
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

X = df_cycle.drop('Fuel Used', axis=1)  # Features
y = df_cycle['Fuel Used']  # Target variable
# Take the object var only and change to int type
object_columns = X.select_dtypes(include=['object']).columns
label_encoders = {}
for col in object_columns:
  label_encoders[col] = LabelEncoder()
  X[col] = label_encoders[col].fit_transform(X[col])
#print(X.dtypes)
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
print(y.head())
print(X_train.head())
#Performing Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
#Neural Network (NN) Algorithm
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from sklearn.preprocessing import LabelEncoder
from keras.utils import to_categorical

from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.callbacks import EarlyStopping

#Initialising ANN
ann = tf.keras.models.Sequential()

#Adding first and second hidden layer
ann.add(tf.keras.layers.Dense(units=128,activation="relu"))
ann.add(tf.keras.layers.Dense(units=64,activation="relu"))

#Adding output layer
ann.add(tf.keras.layers.Dense(units=1,activation="linear"))

#Compile mae
ann.compile(optimizer="adam",loss="mean_squared_error",metrics=['mean_absolute_error'])

es = keras.callbacks.EarlyStopping(monitor='val_loss', mode='min', patience=10,
restore_best_weights=True)
```

```
history1 = ann.fit(X_test, y_test, callbacks=[es], epochs=10, batch_size=10, shuffle=True,
validation_split=0.2, verbose=1)


#plot

loss = history1.history['loss']

val_loss = history1.history['val_loss']

epochs = range(1,len(loss)+1)

plt.plot(epochs, loss, label='Training Loss')

plt.plot(epochs, val_loss, 'r', label="Validation loss")

plt.title('Training and validation loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()


acc = history1.history['mean_absolute_error']

val_acc = history1.history['val_mean_absolute_error']

plt.plot(epochs, acc, 'y', label='Training MAE')

plt.plot(epochs, val_acc, 'r', label='Validation MAE')

plt.title('Training and validation MAE')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()


#predict on test data

predictions = ann.predict(X_test[:5])

print("Predicted values are: ", predictions)
```

```
print("Real values are: ", y_test[:5])
mae = np.mean(np.abs(predictions - y_test.values))
print("mae: ", mae)
#Optimised code with random search
#Neural Network (NN) Algorithm
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from sklearn.preprocessing import LabelEncoder
from keras.utils import to_categorical
from sklearn.metrics import mean_squared_error
import random

from keras.models import Sequential
from keras import layers
from keras.layers import Dense, Dropout
from keras.callbacks import EarlyStopping

def create_model(hidden_layers, learning_rate):
  model = keras.Sequential()
  model.add(layers.Dense(units=hidden_layers[0], activation='relu',
input_dim=X_train.shape[1]))
  for units in hidden_layers[1:]:
    model.add(layers.Dense(units=units, activation='relu'))
  model.add(layers.Dense(1, activation='linear'))
  optimizer = keras.optimizers.Adam(learning_rate=learning_rate)
  model.compile(optimizer=optimizer, loss='mean_squared_error',
metrics=['mean_absolute_error'])
  return model
```

```
# Define hyperparameter search space
param_grid = {
  'hidden_layers': [(32,), (64,), (32,32), (64,64)],
  'learning_rate': [0.001, 0.01, 0.1]
}

# Perform random search
best_mae = float('inf')
best_params = None
##best_mse = float('inf')
for _ in range(10):  # Number of random searches
  params = {k: random.choice(v) for k, v in param_grid.items()}
  model = create_model(params['hidden_layers'], params['learning_rate'])
  es = keras.callbacks.EarlyStopping(monitor='val_loss', mode='min', patience=10,
restore_best_weights=True)
  history = model.fit(X_test, y_test, callbacks=[es], epochs=10, batch_size=10, shuffle=True,
validation_split=0.2, verbose=1)

  predictions = model.predict(X_test)
   mse = mean_squared_error(y_test, predictions)
  mae = np.mean(np.abs(predictions - y_test.values))
  if mae < best_mae and mse < best_mse:
      best_mae = mae
      best_params = params

  print()

print("Best MAE:", best_mae)
print("Best parameters:", best_params)
```

```
#plot
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()

plt.plot(history.history['mean_absolute_error'], label='Training MAE')
plt.plot(history.history['val_mean_absolute_error'], label='Validation MAE')
plt.title('Training and validation MAE')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
#predict on test data
predictions = model.predict(X_test[:5])
print("Predicted values are: ", predictions)
print("Real values are: ", y_test.values[:5])
print("MAE: ", best_mae)
```

---

20. References

3AG (n.d.). What is OEE?

https://www.3agsystems.com/post/what-is-oee#:~:text=OEE%2C%20or%20overall%20equipment%20effectiveness,employed%20across%20mining%20and%20manufacturing.

Amazon (2024). XGBoost Algorithm - Amazon SageMaker. https://docs.aws.amazon.com/sagemaker/latest/dg/xgboost.html

Analytics Vidhya. Understanding the Math behind the XGBoost Algorithm. (2018). https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/

Arizona Department of Environmental Quality. (2021). Ground-Level Ozone (O3) Pollution. https://www.azdeq.gov/ground-level-ozone-o3-pollution

ASQ. (2023).What Is Design of Experiments (DOE)?. Asq.org. https://asq.org/quality-resources/design-of-experiments

BHP (2024). Jimblebar. https://www.labonline.com.au/content/research-development/news/air-pollution-impacts-bees-ability-to-find-flowers-986942486

Brownlee, J. (2016, August 16). A Gentle Introduction to XGBoost for Applied Machine Learning - MachineLearningMastery.com. MachineLearningMastery.com. https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/

Choi, Y., Nguyen, H., Bui, X.-N., Nguyen-Thoi, T., & Park, S. (2020). Estimating Ore Production in Open-pit Mines Using Various Machine Learning Algorithms Based on a Truck-Haulage System and Support of Internet of Things. Natural Resources Research, 30(2), 1141–1173. https://doi.org/10.1007/s11053-020-09766-5

Data Science Process Alliance. (2023). CRISP-DM. https://www.datascience-pm.com/crisp-dm-2

DigitalSreeni. (2020, July 14). *141 - regression using neural networks and comparison to other models*. YouTube. https://www.youtube.com/watch?v=2yhLEx2FKoY

Donges, N. (2023, August 15). *4 disadvantages of Neural Networks*. Built In. https://builtin.com/data-science/disadvantages-neural-networks

Filipe Schmitz Beretta, Áttila Leães Rodrigues, Rodrigo Oliva Peroni, & Costa, J. (2019). Automated lithological classification using UAV and machine learning on an open cast mine. 128(3), 79–88. https://doi.org/10.1080/25726838.2019.1578031

GeeksForGeeks . (2024) What are the Advantages and Disadvantages of Random Forest? https://www.geeksforgeeks.org/what-are-the-advantages-and-disadvantages-of-random-forest/

Gupta, B. (2023, January 13). Random Search in Machine Learning. Scaler Topics. https://www.scaler.com/topics/machine-learning/random-search-in-machine-learning/

IBM (n.d.). What is random forest?. https://www.ibm.com/topics/random-forest

Kumral, M. (2013). Optimizing ore–waste discrimination and block sequencing through simulated annealing. Applied Soft Computing, 13(8), 3737-3744. https://doi.org/10.1016/j.asoc.2013.03.005

Labonline. (2023). Air pollution impacts bees' ability to find flowers. https://www.labonline.com.au/content/research-development/news/air-pollution-impacts-bees-ability-to-find-flowers-986942486

Lorenz, A. (2023). When will we run out of fossil fuels? FairPlanet. https://www.fairplanet.org/story/when-will-we-run-out-of-fossil-fuels/#:~:text=Over%20the%20past%20200%20years

MATLAB (n.d.). What Is the Genetic Algorithm? https://au.mathworks.com/help/gads/what-is-the-genetic-algorithm.html

Mine Payload Technologies (2023). Onboard truck scale alternative.

https://minepayloadtechnologies.com/onboard-truck-scale-alternative/

Neural networks - what are they and why do they matter?. SAS. (2024). https://www.sas.com/en_au/insights/analytics/neural-networks.html

Noriega, R., Pourrahimian, Y. (2022). A systematic review of artificial intelligence and data-driven approaches in strategic open-pit mine planning. Resources Policy, 77, 102727. https://doi.org/10.1016/j.resourpol.2022.102727

Pendharkar, P. C., & Rodger, J. A. (2000). Nonlinear programming and genetic search application for production scheduling in coal mines. Annals of Operations Research, 95, 251-267.https://doi.org/10.1023/a:1018958209290

Rahman, Md. A. (2023, November 29). *Neural network is nothing but a linear regression*. Medium. https://medium.com/@asifurrahmanaust/lesson-3-neural-network-is-nothing-but-a-linear-regression-e05a328a0f23

Serokell Software Development Company (2023). How to Deploy an ML Model in Production. https://serokell.io/blog/ml-model-deployment

Tabesh, M., & Askari-Nasab, H. (2013). Automatic creation of mining polygons using hierarchical clustering techniques. Journal of Mining Science, 49, 426-440.

https://doi.org/10.1134/s1062739149030106

Tavassoli, R. (2021). Fuel Management Opportunities for Open Pit Mining Operations. Info.coencorp.com.

https://info.coencorp.com/blog/fuel-management-opportunities-for-open-pit-mining-operations?trk=article-ssr-frontend-pulse_little-text-block

Wang, Q., Zhang, R., Lv, S., & Wang, Y. (2021). Open-pit mine truck fuel consumption pattern and application based on multi-dimensional features and XGBoost. Sustainable Energy Technologies and Assessments, 43, 100977. https://doi.org/10.1016/j.seta.2020.100977

IBM. (2021, October 6). What is a neural network?. https://www.ibm.com/topics/neural-networks

NVIDIA(2019). Data Science Glossary What is XGBoost?. https://www.nvidia.com/en-au/glossary/xgboost/

McKinsey & Company. (2020, October 5). The mine-to-market value chain: A hidden gem.https://www.mckinsey.com/industries/metals-and-mining/our-insights/the-mine-to-market-value-chain-a-hidden-gem

Creamer Media Reporter. (2022). Supply chain disruptions have hit hard – what can be done to futureproof mining projects and operations? https://www.miningweekly.com/article/supply-chain-disruptions-have-hit-hard-what-can-be-done-to-futureproof-mining-projects-and-operations-2022-04-28

Russell, R., Abel, M., Fernandez, D., Harwood, K., Leach, M., & McCabe, G. (2021). Net-Zero Trucks? Yes, It's Possible. BCG Global. https://www.bcg.com/publications/2021/alternative-fuels-for-trucks-factors-to-consider

Team, E. (2023). How AI Helps Prevent Human Error In Data Analytics. InsideBIGDATA. https://insidebigdata.com/2023/03/18/how-ai-helps-prevent-human-error-in-data-analytics/#:~:text=Humans%20are%20also%20prone%20to,by%20analyzing%20data%20without%20bias.