WARWICK BUSINESS SCHOOL
THE UNIVERSITY OF WARWICK

Masters Programmes: Dissertation Cover Sheet

Student Number:	**5558022**
Degree Course:	**MSBA**
Dissertation Title:	**Predicting Target Grades for Engineering Subjects in GCSE Exams using Predictive Analytics at WMG Academy**
Module Code:	**IB93Y0**
Submission Deadline:	**29 August 2024**
Date Submitted:	**29 August 2024**
Word Count:	**9581**
Number of Pages:	
Have you used Artificial Intelligence (AI) in any part of this assignment?	**No**

Title - Predicting Target Grades for Engineering Subjects in GCSE Exams using Predictive Analytics at WMG Academy

(Student ID:- 5558022)

Acknowledgement

I would like to extend my profound appreciation to my dissertation supervisor, Dr. Jurgen Branke, for his continued guidance and support, always providing valuable insights that have made this work possible. His expertise and words of encouragement have made the entire process quite workable and went a long way in the success of this dissertation.

I also would like to express my gratitude to the Warwick Business School for the great opportunity to learn and develop academically. Finally, I would like to express my deepest gratitude to my family and friends, who supported and encouraged me throughout this period and were my motivating force to overcome every challenge and successfully finalize this dissertation.

Abstract

This dissertation examines the use of machine learning models to predict student grades in GCSE-level engineering subjects at the WMG Academy. Engineering subjects, with its inherent complexity, presents unique challenges for accurate prediction. The study assesses various models, including Neural Networks, Random Forest, and several regression techniques, focusing on how well they can generalise from training data to test data as compared to the simple average benchmark. The results indicate that Neural Networks, after tuning, outperformed the simple average benchmark. The regression models suffered from overfitting due to the small dataset and perform worse as compared to the benchmark. The study also underscores the importance of model tuning and adequate dataset size in improving prediction accuracy, contributing valuable insights to the field of educational data mining.

Contents

Chapter 1: Introduction

Artificial Intelligence (AI) has increasingly been integrated into educational settings, particularly in predicting academic performance. The application of AI spans various domains, with a significant focus on core subjects such as Mathematics and English, where predictive models have traditionally been effective (Chen et al., 2020). Despite these successes, there is a continuous drive to improve the accuracy of these AI-based predictions. This study is motivated by the ongoing efforts to enhance the precision of AI predictions in the context of education.

1.1 Background

For students in Year 11 (ages 15–16) in England, the General Certificates of Secondary Education (GCSEs) are the main credentials. These qualifications are offered across various subjects, including English, mathematics, science, history, geography, art and design, design and technology, business studies, classical civilisation, drama, music, and foreign languages. Engineering is still a specialised field with few predictive models, despite several research demonstrating various techniques to forecast grades for core topics using their historical performances or other socioeconomic characteristics.

Predicting student performance in advance can help students and their teachers to keep track of the progress of a student (Zhang and Li, 2018). Many institutions, such as WMG Academy, have implemented continuous evaluation systems, which have proven beneficial in helping students improve their performance. However, predicting grades in complex subjects like Engineering sets unique challenges due to the interdisciplinary nature of the field, which integrates theoretical knowledge, practical skills, and creativity (Xu et al., 2022). There is a strong case to investigate advanced techniques for predicting students' performance in this subject because of this gap in the literature. With an emphasis on students at the WMG academy, the dissertation seeks to investigate and create a prediction model that can precisely forecast student grades in engineering courses.

1.2 Dissertation Objectives

This dissertation's main goal is to investigate how well different machine learning models predict WMG Academy students' grades in engineering courses. The development and assessment of machine learning models, including neural networks, Lasso, Ridge, Random Forest, and linear regression, are the goals of this project. Metrics including Mean Squared Error (MSE), Mean

Absolute Error (MAE), and R2 Score are emphasised in this research as a means of evaluating these models' predictive performance against a basic average baseline.

The dissertation examines the impact of data preprocessing techniques, including standardisation, normalisation, and principal component analysis (PCA), on the performance of these models. The dissertation also investigates the impact of feature engineering techniques, including polynomial expansion and log transformations, on model performance. The models were meticulously tuned to improve their predictive accuracy and ensure robust performance. However, as revealed through preliminary findings, these advanced models, except for neural networks, did not outperform simple benchmarks based on the average of predicted math and English grades. This observation underscores the complexity of predicting grades in subjects as interdisciplinary as engineering.

1.3 Dissertation Structure

This report is systematically structured across five chapters including Introduction, each addressing key components of the dissertation: -

Chapter 2: Literature Review - This chapter provides an in-depth analysis of pertinent literature, laying the groundwork for the analytical framework that will guide the interpretation of the study's data and findings.

Chapter 3: Methodology - This section describes the procedures for gathering data, preparing it, and using analytical techniques. It focuses especially on the choice and rationale of the machine learning models that were applied to the analysis.

Chapter 4: Results, Interpretation, and Discussion - In this chapter, the results of several predictive models are presented, and their performance is assessed at predetermined benchmarks. It also discusses the difficulties that were faced and offers suggestions derived from the analysis.

Chapter 5: Conclusion - The final chapter synthesises the key findings of the study, discussing their implications for educational practice and suggesting directions for future research.

In terms of predicting students' grades at the WMG Academy, the research findings may help determine if a machine learning model can perform better than the benchmark or whether a simple benchmark is the most effective.

Chapter 2: Literature Review

2.1 Introduction

This literature review provides the foundation for studying how student grades in engineering subjects at the GCSE level can be predicted using machine learning techniques. It begins by looking at how Educational Data Mining (EDM) has developed globally, highlighting how data mining techniques have been used to extract meaningful insights from educational data.

The review then delves into the specific challenges of applying predictive models like Bayesian Networks Random Forests, and neural networks to different subjects. These models have been effective in other areas, but engineering education at the GCSE level presents unique complexities that need careful consideration.

This chapter also touches on the effects of the COVID-19 pandemic on student performance, which is an important factor in this study. Reviewing existing research, not only provides a comprehensive overview of EDM and its applications but also identifies key gaps and opportunities for further investigation. This sets the direction of the dissertation for developing more accurate methods to predict grades in engineering subjects, ultimately aiming to help improve educational outcomes in this field. While this study focuses on the GCSE setting, it is important to note that insights from other studies, discussed below, at other educational levels, are highly relevant and may be transferable for predicting students in engineering subjects at the GCSE level.

2.2 Educational Data Mining and Predictive Modelling

Educational data mining (EDM) has emerged as a significant field over the past two decades, it has been driven by the increasing availability of student data and the need for more personalized educational experiences. EDM means applying data mining techniques to educational settings to improve learning outcomes and understand educational processes. Predictive modelling within EDM is a significant application that involves using algorithms to predict future student performance based on historical data (Alshareef et al., 2020), (Toradmal et al., 2023). These predictive models can be instrumental in identifying at-risk students, optimizing curriculum, and enhancing decision-making processes in educational institutions.

The evolution of EDM has been further enriched by the growing complexity of learning environments, as highlighted in the work of (Baker et al., 2016). The paper mentioned that EDM tools are becoming more suitable for such complex tasks. Their research emphasizes the need for advanced data mining methods that can address the intricate dynamics of modern educational settings, such as project-based learning and intelligent tutoring systems. These environments generate multifaceted data that require sophisticated analytical techniques to draw meaningful inferences about student learning processes and outcomes. The study also concluded the importance of tailoring data mining methods to specific educational contexts, ensuring that the insights generated are both accurate and actionable.

Alshareef et al. (2020) that while Bayesian Networks or Random Forest algorithms are highly effective for predicting student performance, challenges persist when dealing with small datasets, and there is a continuous effort to refine methodologies for better classification and recommendation systems. Toradmal et al. (2023) in his paper shows that Artificial Neural Networks (ANN) also have reliable dataset performance. Moreover, the integration of EDM with Learning Analytics (LA), has been suggested to develop a more student-focused strategy to predict students’ performance, which can lead to continuous improvement in higher education (Aldowah et al., 2019).

Machine learning (ML) and artificial neural networks (ANNs) have been applied to predict student grades with varying degrees of success. Baashar et al. (2022) highlight the use of ANNs in combination with data analysis and data mining methodologies to predict academic achievement in higher education. It notes that while ANN techniques are frequently used, the selection of input variables is context-dependent, and there is a gap in applying these techniques to real-life situations to improve educational outcomes. Similarly, Ojajuni et al. (2021) presents a study where several ML models, including ANNs, were compared to predict student academic success and found that the Extreme Gradient Boosting (XGBoost) model had a high accuracy rate of 97.12%. Rodríguez-Hernández et al. (2021) also discusses the use of ANNs to systematically classify students' academic performance, with the ANN model outperforming other ML algorithms in terms of recall and the F1 score.

While Baashar et al. (2022) suggests a lack of tangible findings in the actual application of ANN techniques for improving student outcomes, Rodríguez-Hernández et al. (2021) demonstrates a successful implementation of ANNs in a large-scale educational context. Moreover, Ojajuni et al.

(2021) results indicate that ML technology can be effectively applied in educational settings to identify at-risk students and inform educators' decision-making.

In brief, EDM and predictive modelling play a crucial role in the academic domain by providing insights that can lead to improved educational strategies and student support. The effectiveness of these models is contingent upon the continuous evaluation and enhancement of the techniques used. Future research should focus on addressing the challenges of large datasets and further collaboration between EDM and LA communities to advance the field (Siemens and Baker, 2012), (Toradmal et al., 2023). The promising results from the application of ML and ANNs Baashar et al. (2022), Ojajuni et al. (2021), Rodríguez-Hernández et al. (2021) further underscore the need for systematic implementation to fully realize their potential in improving student outcomes.

2.3 Impact of COVID-19 on Educational Outcomes

The COVID-19 pandemic has had a profound impact on education worldwide, disrupting traditional learning environments and altering student performance patterns. The key impact of COVID-19 on GCSE was that the GCSE marks were not determined by the Office of Qualifications and Examinations Regulation (Ofqual), the government department that regulates qualifications, exams and tests in England. They were forced to consider alternative approaches to award grades which included their algorithm and teachers awarding the marks.

ofqual (2019) use their algorithm to predict student’s grades. However, Denes (2023) observed that the grades were inflated in the COVID-19 years. Additionally, teachers also decide the marks and they tend to inflate the marks as well (Anders et al., 2020). This can be the possible reason for higher marks in COVID years as compared to non-COVID years in the dataset. Also, in their framework Ofqual used standardisation to scale the marks before prediction, this method is used in this project.

Several studies have explored how the pandemic affected educational outcomes, often showing declines in student performance due to the sudden shift to online learning. In their report, Di Pietro et al. (2020) indicated a weekly learning loss of between 0.82 and 2.3% of a standard deviation and mentioned that students from less advantaged backgrounds are likely to experience a larger decline in learning.

The shift to online learning during the COVID-19 pandemic has been widely studied, revealing a mixed impact on student performance. For instance, projections suggested that students who lacked remote instruction in spring 2020 would start the new academic year with only 37% to 50% of typical learning gains in mathematics and 63% to 68% in reading. Even those with partial remote learning retained only 60% to 87% of their usual gains (Kuhfeld et al., 2020).

The collective research from various studies indicates that the COVID-19 pandemic has had a predominantly negative impact on educational outcomes across different levels and regions. The studies consistently report a decline in student learning outcomes during the pandemic compared to pre-pandemic levels (Agarwal and Behera, 2023) (Lestari and Syaimi, 2021) (Rahman, 2021). This decline is attributed to the shift to online learning, which, despite the technological advancements, has presented challenges in maintaining the quality of education (Miranda et al., 2023) (Lestari and Wachidah, 2023).

The paper by Azevedo et al. (2021) adds significant insights into the global impact of COVID-19 on education, particularly emphasizing the long-term consequences of school closures. The authors simulate the potential effects of these closures on schooling and learning outcomes, predicting a substantial decline in the Learning Adjusted Years of Schooling (LAYS). Their analysis indicates that school closures could lead to a loss of between 0.3 and 1.1 years of schooling when adjusted for quality, reflecting severe disruptions in educational attainment. Additionally, the study forecasts that nearly 11 million students could drop out due to the economic shocks induced by the pandemic, further exacerbating inequalities, especially among marginalized groups.

While the general trend points towards a decrease in learning outcomes, there is evidence of variability. For example, a study found that parental involvement during online learning led to an increase in average scores for a Physics course from 80.04 to 82.11 (Suhariyono and Retnawati, 2022). In addition, the impact of the pandemic on educational outcomes is not uniform, with factors such as student anxiety Wu et al. (2023) and socioeconomic status Hammerstein et al. (2021) influencing the extent of the impact. Besides, the adoption of online learning has been recognized as a necessary tool to continue education during the pandemic, despite its associated challenges (Hidalgo-Camacho et al., 2021).

Overall, the COVID-19 pandemic has significantly disrupted educational outcomes, with most studies reporting a decline in student learning achievements. However, the effects are nuanced and influenced by various factors, including the level of parental involvement, student anxiety, and socioeconomic disparities. The findings of Azevedo et al. (2021) underscore the global scale of these disruptions, highlighting the potential for long-term setbacks in educational attainment and the widening of existing inequalities.

2.4 Research Gaps

The literature reviewed highlights several gaps that this study aims to address. First, there is a lack of predictive modelling research specifically focused on engineering education, a niche area with unique challenges and opportunities. Second, while machine learning models have been widely used in educational research, their application in predicting engineering grades remains underexplored specifically for the GCSE level.

Furthermore, the impact of COVID-19 on educational outcomes, especially through Ofqual's pandemic grading standardization, has influenced this study's methodology. Specifically, standardisation of the marks in the data preprocessing stage was inspired by the Ofqual framework to enhance prediction accuracy. By addressing these gaps, this research contributes to the broader field of educational data mining and predictive modelling. The integration of advanced machine learning models, the consideration of COVID-19's impact, and the rigorous evaluation of data preprocessing techniques offer new insights into student performance prediction in engineering education.

2.5 Summary

This literature review sets the stage for exploring how machine learning can be used to predict student grades in GCSE-level engineering subjects. It begins by examining the global development of Educational Data Mining (EDM), focusing on how data analysis techniques have been applied to draw meaningful insights from educational data. The review then explores the challenges of applying various predictive models across subjects, highlighting the complexities of engineering education at the GCSE level that require careful consideration.

The review also addresses the significant impact of the COVID-19 pandemic on student performance, which is a crucial factor in this study. The pandemic caused widespread disruption in education, leading to mixed outcomes for students. Research indicates that those without access to remote learning experienced significant setbacks. Furthermore, the review considers the long-term effects of the pandemic on educational practices, noting how the shift to online learning has led to ongoing changes, with a growing reliance on digital tools alongside traditional teaching methods.

In the end, the literature review identifies key gaps in existing research, particularly the lack of focus on predictive modelling in engineering education and the need for better data preparation to enhance model accuracy. By addressing these gaps, the dissertation aims to provide new insights into predicting grades in engineering subjects, contributing to improved educational strategies and outcomes.

Chapter 3: Methodology

The methodology outlines the systematic approach taken, linking the research objectives with the methods employed. The approach ensures that the research is conducted with rigor, transparency, and alignment with the Dissertation goal.

3.1 Research Design

The dissertation adopts a quantitative research design, focusing on the development and evaluation of predictive models using numerical data. The structured approach of this research ensures that the results can be generalized to a larger population, making the findings relevant beyond the specific context of WMG Academy. The study use of machine learning algorithms to predict student performance based on historical data, demographic information, and other relevant variables. The research design involves the selection and application of various machine learning models, including Linear Regression, Lasso, Ridge, and Neural Networks. These models were chosen because they can handle the complexity of the relationship between the variables efficiently.

The research applies these models to both original and pre-processed datasets, exploring the effects of data transformations like normalisation and feature engineering. The model performances are compared against a simple average benchmark for maths and English subjects, discussed in detail in Section 3.4.1. Several evaluation metrics, such as Mean Squared Error (MSE), Mean Absolute Error (MAE), and R^2 scores, are used to assess model accuracy and reliability, ensuring a comprehensive evaluation of the best-performing models for predicting student grades.

3.2 Data Collection

The data for this study was sourced from WMG Academy and includes anonymised student grades across various subjects. The dataset contains 314 rows, comprising 29 columns, includes the target variable (Grade), a COVID-19 indicator, and 27 other metrics (Metric1 to Metric27) (refer Table A1 for data dictionary). The data has student grades during both the COVID-19 and non-COVID-19 periods. The inclusion of the COVID-19 indicator is important to this study, as it enables an examination of potential changes in student performance attributable to the pandemic.

3.3 Data pre-processing

Data preprocessing is a crucial stage in developing predictive models. Common techniques such as normalisation, standardization, outlier treatment, and feature engineering play a significant role in enhancing model accuracy. These preprocessing steps are meticulously chosen and customised to fit the specific data characteristics and the machine learning tasks being addressed.

3.3.1 Exploratory data analysis (EDA)

Descriptive statistics and missing values

Exploratory Data Analysis (EDA) was conducted to gain an initial understanding of the dataset. Descriptive statistics revealed that the target variable, *Grade*, has a mean of 4.57 and a standard deviation of 2.13, as shown in Figure 3.1. The histogram peaks at grade 4, indicating that this is the most common grade. There is also a noticeable drop-off in frequency at both the lower and higher ends of the scale, suggesting fewer students either performed very poorly or exceptionally well. The complete set of descriptive statistics is provided in Table (refer Table A2). The distribution of all the variables is in Figure A1. The dataset was also examined for missing values, and none were identified.

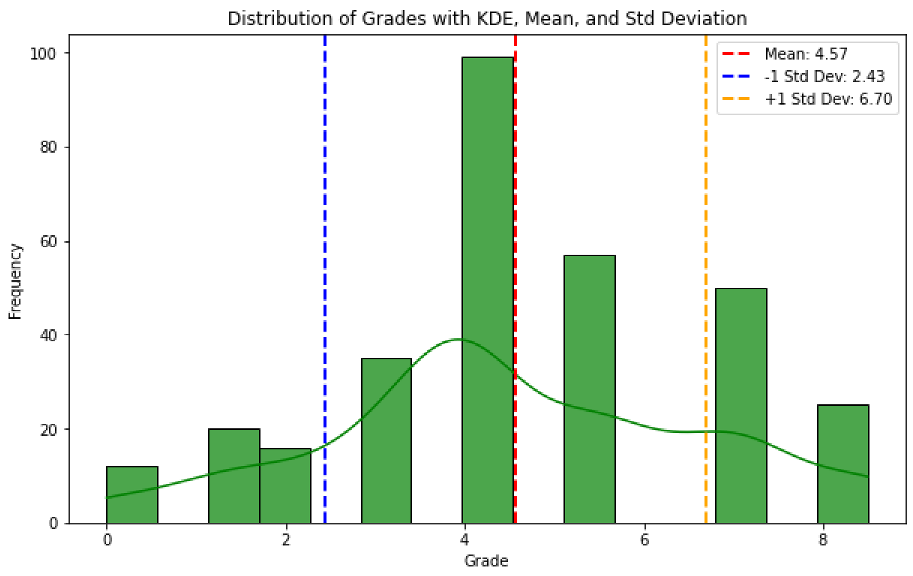


Figure 3.1. Distribution of target variable ‘*Grade*’

Correlation Analysis and Insights

A Pearson correlation matrix was computed to determine the correlation coefficients between each pair of features and between each feature and the target variable, Grade. The target variable, Grade, was shown to have rather high positive correlations with features like *Metric13, Metric15, Metric26, Metric27* (a proxy for "Maths" grades), and *Metric14* (refer Table 3.1). Also, it is notable that *Metric1* shows a negative correlation with Grade, indicating that it may behave differently from the other features (refer Figure A2) and (refer Table A3 for correlation coefficient of all variables).

Variable name	Correlation Coefficient
Metric13	0.528508
Metric15	0.527822
Metric26	0.525316
Metric27	0.523568
Metric14	0.522542
Metric1	-0.234406

Table 3.1. Correlation coefficient of features with the grade

3.3.2 Feature Selection

The process of feature selection involves identifying and removing irrelevant and redundant information to reduce the dimensionality of the data. This reduction is crucial, as it enables models to operate faster and more efficiently, improving predictive performance. After doing a multi-collinearity check on the normalised data, it was discovered that several variables showed strong correlations; these correlations were identified using a threshold of 0.8.

The Variance Inflation Factor (VIF) analysis also revealed that some variables, such as *COVID* (1.089), *Metric1* (1.127) had low VIF values, indicating minimal multicollinearity and suggesting that these variables are not highly collinear with others (refer Table A4). Other variables like *Metric2, Metric12, Metric13,* and *Metric27* had VIF values greater than 10, indicating severe multicollinearity. Alarmingly, several metrics, including *Metric6* through *Metric25*, exhibited a VIF of infinity, which indicates perfect multicollinearity. This suggests that these variables are highly correlated with others, potentially leading to instability in the regression models (refer Table A5). To manage multicollinearity and preserve the integrity of the dataset, Principal Component Analysis (PCA) was conducted later.

The pair plot (refer Figure 3.2), focusing on selected variables, provides a detailed overview of pairwise relationships between variables. The diagonal elements of the pair plot show histograms of the individual variables, revealing that most metrics exhibit a roughly normal distribution. However, *Metric1* and *COVID* appear to have discrete values. Notably, *Metric5, Metric6*, and *Metric7* show strong positive linear relationships, as shown by the tight clustering of points along a line, revealing potential multicollinearity. the other hand, certain scatter plots show a diffuse cloud of dots devoid of any clear pattern, which suggests that the correlations between those variable pairs are weak or non-linear.

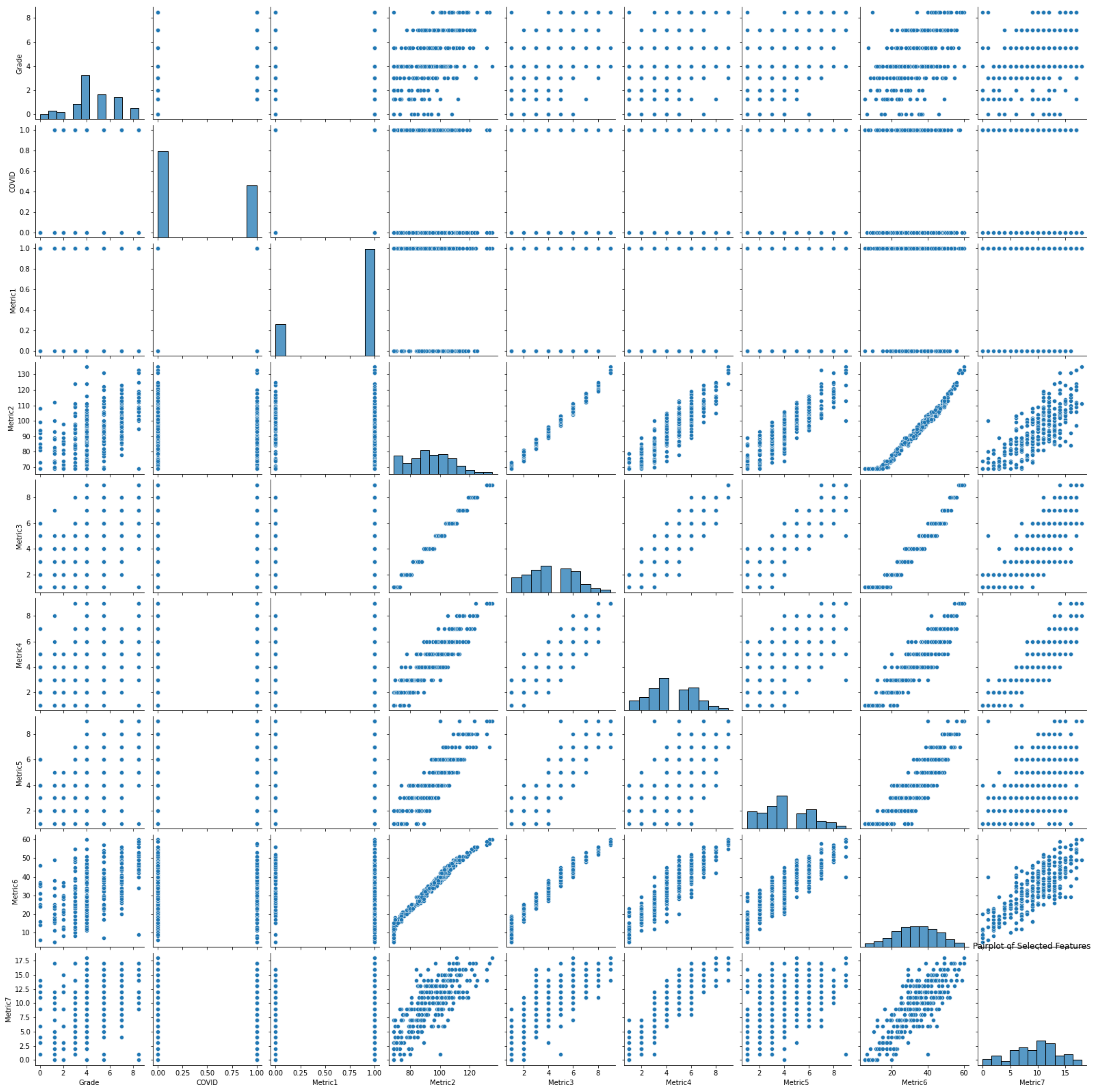


Figure 3.2 Pair-plot visualizing relationships among selected features

3.3.3 Normalisation

The impact of the COVID-19 pandemic on student performance introduced unique challenges in data analysis, particularly due to the discrepancies in academic outcomes between students affected by the pandemic and those who were not. To rectify this, student grades were adjusted by normalisation, guaranteeing equitable comparisons throughout the dataset.

Steps Taken in Normalisation

1. Segregation of COVID and Non-COVID Student Data

- The dataset was divided into two groups: students affected by COVID-19 (*covid_data*) and those who were not (*non_covid_data*). This segregation was necessary to apply targeted normalisation and accurately reflect the challenges faced by the COVID-affected students.

2. Calculation of Average Grades and COVID Bonus

- For the COVID-affected group, the average grade (*average_grade_covid*) was calculated to be 4.9470 compared to 4.3380 for the non-affected group (*average_grade_non_covid*). The difference between these two averages, termed as "*COVID Bonus*," was 0.6090. This "*bonus*" quantifies the additional challenges faced by students during the pandemic and serves as a corrective factor in the normalisation process.

3. Adjustment of Grades for COVID-Affected Students

- To normalize the grades, the "*COVID Bonus*" was subtracted from the grades of the COVID-affected students. This step was taken to level the playing field, ensuring that the academic performance of students impacted by COVID-19 was comparable to those who were not. The resulting normalized dataset (*data_normalized*) reflects this adjustment, offering a more accurate representation of student performance.

After normalisation, the average grades for both groups were recalculated to verify the process's effectiveness. Both the COVID-affected and non-affected groups now had an average grade of 4.3380 after the averages were adjusted, demonstrating a successful reduction of the pandemic's effect on student grades.

3.3.4 Data Standarisation

Data standardisation was done to scale the variables. Since the dataset includes a broad range of metrics, such as attendance, prior exam scores, demographic information, and other relevant factors which are anonymised in this case, models can become sensitive to the scale of input feature.

The dataset consisted of various independent variables (Metrics) and one dependent variable, *Grade*. For standardization:

- The Independent variables (Metrics) were grouped into a feature matrix denoted as *X*
- The dependent variable, *Grade*, was stored separately as *Y*.

Standardisation Process

The standardisation was performed using the *StandardScaler* from the *sklearn.preprocessing* module in Python. The process involved two steps: fitting and transforming the data.

1. **Fitting the Scaler**:
   - The *StandardScaler* was first fitted to the feature matrix *X*. During this step, the scaler computed the mean (μ) and standard deviation (σ) for each feature.
2. **Transforming the Data**:
   - The scaler transformed the feature set '*X*' using the formula:

$$X_{Scaled} = \frac{X - \mu}{\sigma}$$

The transformed features (*X_scaled*) represent the standardized features, ensuring that each feature has a mean of 0 and a standard deviation of 1. This ensures all features are placed on an equal footing, allowing the machine learning models including regression models and neural networks to detect and learn from the underlying patterns in the data more effectively.

3.3.5 Handling Outliers

Outlier handling is also an important component of the data preprocessing, due to its potential influence on model accuracy and reliability. To identify potential outliers, a box plot was generated (refer Figure 3.3).

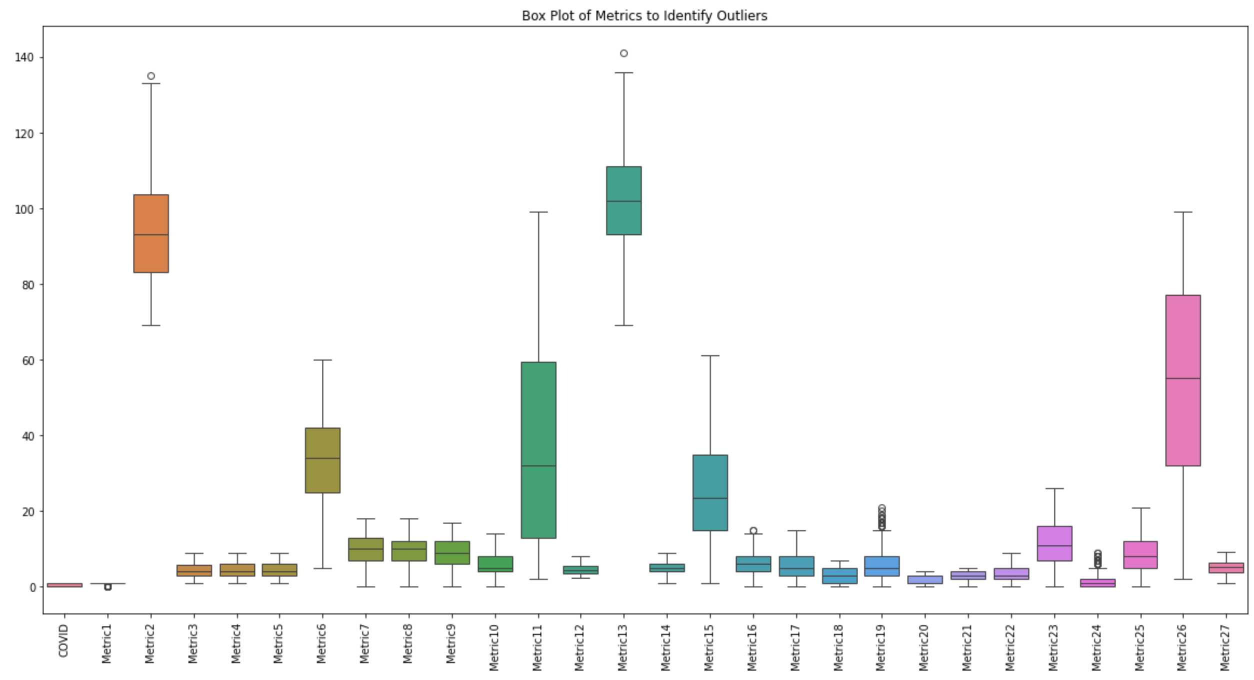


Figure 3.3 Box plot representing outliers

The box plot offers a detailed visualisation of potential outliers across various metrics. Metric19*, Metric24* show outliers, suggesting potential subgroups or unique conditions within the data. In contrast, *Metric3, Metric4,* and *Metric5* display minimal variability with no significant outliers.

To lessen the impact of high values on the prediction models, an initial method was used to cap outliers at the 95th percentile. This strategy was used as a precaution to avoid these uncommon extreme values from unduly impacting the results, even though there were not many outliers in the measurements. Subsequent investigation, however, showed that this strategy did not result in the anticipated increases in model performance.

3.4 Data Analysis

The data analysis was structured to systematically explore the relationships within the dataset, establish benchmarks, build predictive models, and refine them through advanced techniques such as feature engineering and hyperparameter tuning.

3.4.1 Benchmarking

Benchmarking was conducted to establish a baseline against which the performance of various predictive models is compared. This process helps determine whether the added complexity of machine learning models offers any significant improvement in prediction accuracy over a simple, straightforward approach. The benchmark was calculated on the COVID adjusted data.

In this project, the simple average benchmark was calculated based on the average of predicted grades in two key subjects: English and Maths using the below formula. The expectation is that any machine learning model should ideally outperform this simple average benchmark. These features were chosen based on guidance from WMG Academy, which identified Metric12 and Metric27 as proxies for English and Maths, respectively and they currently employ a similar approach for evaluating student performance.

$$Simple\ average_i = \frac{English\ Grade_i + Maths\ Grade_i}{2}$$

Where:

- $English\ Grade_i$ is value from *Metric12* for *i*-th student
- $Maths\ Grade_i$ is value from *Metric27* for *i*-th student

Performance metrics of the Benchmark Model

The performance metrics, Mean Squared Error (MSE), Mean Absolute Error (MAE), and R² Score, of the simple average benchmark was calculated as follows:

1. Mean Squared Error (MSE):

- MSE is calculated as the average of the squared differences between the actual grades and the predicted grades (Simple Average).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_{ai} - y_i)^2$$

- n : Number of observations
- y_{ai} : Actual grade for *i*-th student
- y_i : Simple average for *i*-th student

2. Mean Absolute Error (MAE):

- MAE measures the average of the absolute differences between the actual grades and the predicted grades.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_{ai} - y_i|$$

3. R-Squared (R²) Score:

- The R² score indicates how well the simple average predictions approximate the actual grades. It is a measure of the proportion of variance in the dependent variable that is predictable from the independent variables.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_{pi} - y_i)^2}{\sum_{i=1}^{n}(y_i - y_m)^2}$$

- y_m : Mean of the actual grades

Calculation of the Benchmark Metrics

The performance metrics of the benchmark are shown in Table 3.2, which provides a reasonable starting point but highlights significant room for improvement. These metrics are used with the performance metrics of the machine learning models further discussed in the results section. The purpose of this comparison was to determine if the complexity introduced by these models provided any significant predictive advantage over the simple average benchmark.

Benchmark MSE	Benchmark MAE	Benchmark R² score
3.19	1.41	0.28

Table 3.2. Simple average benchmark performance metrics

3.4.2 Principal Component Analysis (PCA)

A normalised dataset is subjected to Principal Component Analysis (PCA) in order to resolve multicollinearity. It has been done such that a threshold retains 95% of the variance; hence, the principal component captures most of the variability in the dataset while reducing dimensionality. Then, all machine learning models with PCA-transformed data were executed except the neural networks. However, this does not exhibit too much improvement in performance as compared to the models using original features.

The PCA analysis retained 10 principal components (PC), which carried 95% of the variance. These 10 PC represent new, uncorrelated features that are linear combinations of the original features, potentially making the model more robust by reducing the risk of overfitting. The scree plot shows that 2 PCs carry almost 76% of the variance and remaining is captured by rest of the PCs (Refer figure 3.4).

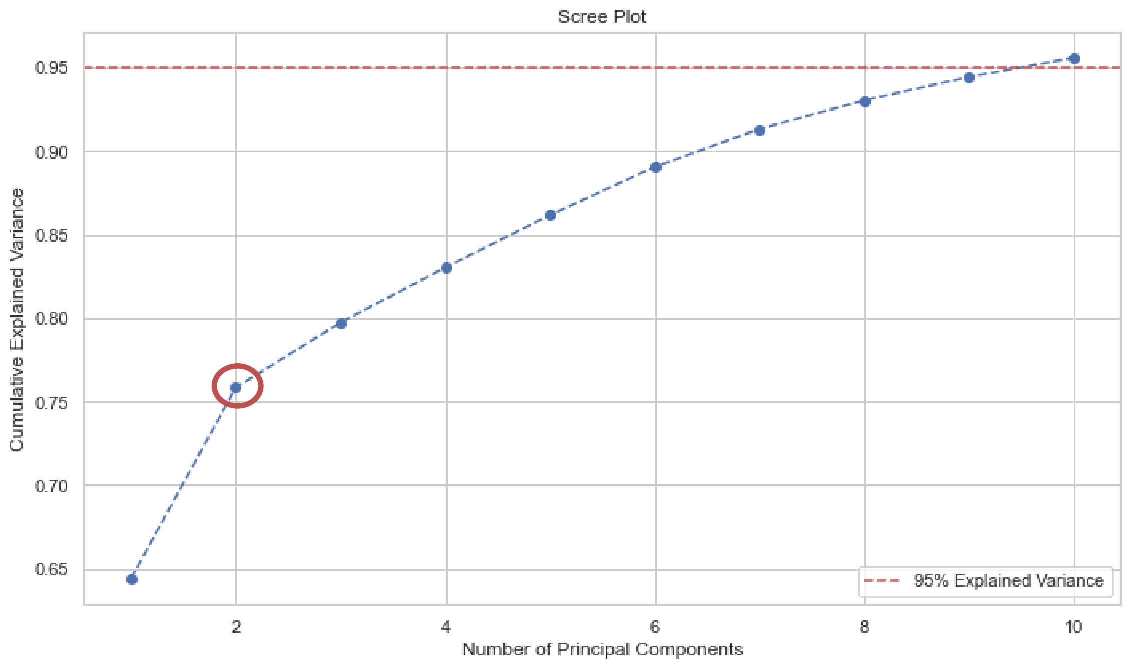


Figure 3.4 Scree plot for PC selection

The loading table show the correlation between variables and top 5 PC's (refer Table 3.3). The *COVID* variable has significant loadings on PC4, suggesting these components capture almost all COVID-related variance. *Metric2* and *Metric3* were strong drivers of PC1 and PC2 to indicate a general pattern in the data. Metric5 shows mixed influence, with positive contributions to PC1, PC2, and PC5, and a negative impact on PC3. As the results section explains, however, the PCA-transformed data did not result in appreciable increases in model performance despite these efforts. All the loading coefficients can be found in the Appendix (refer Table A6, Table A7).

Variable	**PC1**	**PC2**	**PC3**	**PC4**	**PC5**
COVID	-3.2E-17	-1.4E-16	-1.1E-15	1	3.43E-16
Metric1	-0.01774	-0.08919	0.837454	8.78E-16	0.334956
Metric2	0.210903	0.249794	0.019155	-2.2E-16	0.011703
Metric3	0.208341	0.250649	0.022917	1.28E-16	-0.0006
Metric4	0.182944	0.265707	0.191176	4.06E-16	-0.28581
Metric5	0.195167	0.192011	-0.17876	-3.8E-16	0.329933

Table 3.3. Top 5 PC's loading with the variable

3.4.3 Feature engineering

Feature engineering plays a crucial role in machine learning by transforming raw data into meaningful features that enhance model predictive power. In this project, it was done on the original and normalised datasets to see the effect of feature engineering on model performance. To create new features that more effectively capture underlying data patterns, methods like polynomial features and log transformations were employed.

The importance of feature engineering lies not only in its potential to boost the accuracy of predictive models but also in its varying impact depending on the machine learning algorithms used, as highlighted by (Heaton, 2016) in his empirical analysis.

Steps in Feature Engineering

1. Polynomial Features:

One of the primary techniques applied was the creation of polynomial features. Polynomial feature expansion involves generating new features by taking combinations of the original features raised to a power. For example, if x_1 and x_2 are original features, the polynomial expansion might create features like

$$Polynomial\ features = \{x_1^2,\ x_1 \times x_2,\ x_2^2\}$$

This allows the model to capture non-linear relationships between the features, which can be particularly useful in intricate datasets where the interaction of variables plays a very vital role in determining the target outcome. In this project, polynomial features were generated up to the second degree (interaction-only not the square terms).

Log Transformation:

Another transformation applied was the log transformation of skewed features. The skewed features are those with asymmetrical data distributions, either skewed to the left (negative skew) or right (positive skew). In the project, *Metric24* and *Metric19* have significant positive skewness and *Metric1* has significant negative skewness (refer Table A8). A threshold skewness > 0.5 or < -0.5 was decided to select which features to transform. This technique is used to reduce skewness and stabilize variance, making the data more normally distributed.

(Heaton, 2016) discusses the use of logarithmic transformations, particularly in models like Deep Neural Networks (DANN), Random Forests, where these transformations help in handling multiplicative relationships more effectively. The formula applied in this study was:

$$y = \log(x)$$

where x is the original feature, and y is the transformed feature. This transformation was particularly beneficial in stabilizing variance across different features, contributing to more reliable model performance. The transformation used log (1 + x) rather than log(x) since there are 0's in COVID features. This was done to avoid mathematical error. Although log transformation generally helps in stabilizing variance and reducing the influence of outliers, it did not significantly alter the model's predictive power in this study on either of the datasets (refer Figure 3.5).

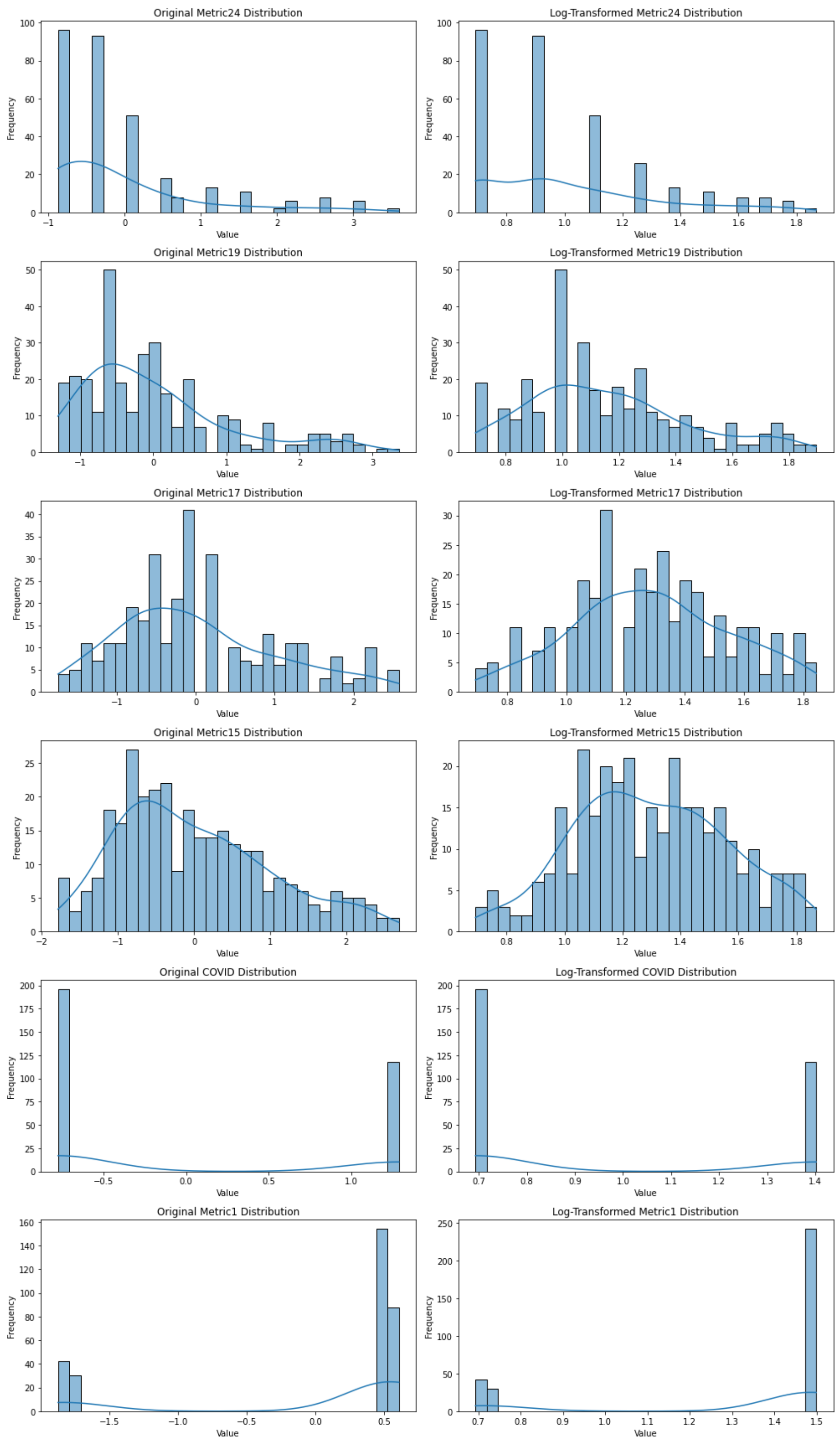


Figure 3.5 Effect of Log transformation on skewed variables on the normalised data

Overall Impact on Model Performance

Although these feature engineering techniques have advantages in theory, their deployment in this project did not appear to appreciably improve the benchmark's performance. This suggests that additional modification may not provide much benefit because the initial characteristics were already adequate in capturing the relationships required for prediction. Considering the short dataset, the overfitting issue that was discovered following the result analysis may worsen if more features are added.

3.4.4 Modelling

This phase involved the development and evaluation of various machine learning models to predict student grades. After ensuring uniform data processing, the dataset was split into training and testing sets using an 80-20 split—80% for training and 20% for testing. This split was chosen to maximize the data available for model training while reserving a sufficient portion for evaluating performance on test data. The test set provided as unseen data, while the training set was utilised to create the models.

Implementation of Machine Learning Models

A variety of machine learning models were selected for this study, including Linear Regression, Lasso, Ridge Regression, Random Forest and neural networks. These models have different build-in parameters which can be adjusted, tuning is used in this project, which is discussed in a later section, enabling them to handle different types of data and to capture both linear and non-linear relationships.

For the project, a feedforward neural network was constructed. The architecture included multiple hidden layers equipped with ‘*ReLU’* activation functions to effectively model these relationships. Dropout layers were strategically incorporated to prevent overfitting, a common issue in neural network training where the model becomes too specialized to the training data and performs poorly on new data. It was trained with the Adam optimiser, which is renowned for having an adjustable learning rate and facilitating more rapid and effective convergence during training. MSE was chosen as the loss function and served as the main performance indicator for the model.

Model training and evaluation of Machine Learning Models

Each model was applied to different versions of the dataset, including the original data, PCA-transformed data, and feature-engineered data, to evaluate the impact of these transformations on model performance. The models were trained on the training data.

The neural network was trained over 50 epochs and 100 epochs with a batch size of 32, a configuration chosen based on a balance between computational efficiency and model performance. However, the error improved a little with the 100 epochs further discussed in the results. In the neural network training, 20% of the training data was set aside as a validation set, allowing for real-time monitoring of the model's performance on data not seen during each epoch. This validation process helped in identifying any potential overfitting early, ensuring that the model could generalize well to the test data. The training errors were calculated to compare the model for potential overfitting issues.

The test set was subsequently used to evaluate the performance of the machine learning models and to assess how accurately they make predictions. All the models were evaluated using MSE, MAE and R^2 score, like benchmark (refer Section 3.4.1) but instead of 'Simple average for *i*-th student', the Predicted value of the target variable by the model was considered.

These error metrics were also calculated on the training data to check for signs of overfitting. It was found that neural networks performed best based on these metrics as compared to the benchmark, which is discussed in detail in the results.

3.4.5 Hyperparameter tuning

The hyperparameter tuning delves into the process of optimizing machine learning models. The objective here is to identify the most effective model configuration for accurate prediction of student grades.

Tuning Machine learning models

Multiple datasets like original data, normalised, PCA-transformed and featured engineered, were employed to ensure a comprehensive evaluation of the models. The hyperparameter tuning was performed using grid search with cross-validation. The *GridSearchCV* function was employed to perform an exhaustive search over the specified hyperparameter grid. In this study, 5-fold cross-validation was employed as part of the hyperparameter tuning process. The choice of 5 folds balances computational efficiency with the accuracy of the performance estimate and is widely used in hyperparameter tuning.

In cross-validation, the dataset is randomly divided into five equal-sized folds. For each fold, the model is trained on four of the folds (80% of the data) and validated on the remaining fold (20% of the data). This process is repeated 5 times, with each fold serving as the validation set once.

Table 3.4 shows the different hyperparameters for each model and their meaning:

Model Name	Parameter value	Meaning
Linear Regression	fit_intercept = True	The model calculates the intercept, allowing the regression line to not necessarily pass through the origin
	copy_X = True	The model makes a copy of the input data before fitting the model
	Positive = False	Coefficients of the linear model are not constrained to be non-negative
Lasso Regression	Alpha = 0.01, 0.1, 1.0, 10.0	Controls the strength of the L1 regularisation
	max_iter = 5000	Maximum number of iterations for the optimization algorithm
Ridge Regression	Alpha = 0.01, 0.1, 1.0, 10.0	Controls the strength of the L2 regularisation
	max_iter = 5000	Maximum number of iterations for the optimization algorithm
Random Forest Regressor	n_estimators = 100, 300	Number of trees in the forest
	max_depth = 10, 20	Maximum depth of the trees in the forest
	min_samples_split = 5, 10	Minimum number of samples required to split an internal node
	min_samples_leaf = 2, 4	Minimum number of samples that a leaf node must have
	Bootstrap = True	Bootstrap samples (random sampling with replacement) are used when building trees
Neural Network	optimizer = *adam*	Effective in dealing with sparse data and situations where the learning rate needs to be adjusted dynamically during training
	optimizer = *rmsprop*	Deals with the vanishing and exploding gradient problems by dividing the learning rate by an exponentially decaying average of squared gradients
	batch Size = 32, 64	Number of training examples utilized in one forward/backward pass
	epochs = 50, 100	Number of times learning algorithm will work through the entire training dataset
	dropout_rate = 0.3, 0.5, 0.7	Randomly set a fraction of input units to 0 at each update during training, forcing network to learn redundant representations and prevents co-adaptation of neurons

Table 3.4 Hyper parameters of machine learning models

The models were run using these hyperparameters with the best hyperparameters, determined by cross-validation, were selected and refitted on the entire training dataset to maximize performance. The main idea behind this was to carefully adjust the parameters so that the models were not only able to fit the training data effectively but also demonstrated strong generalization to the testing data, reducing the likelihood of overfitting. However, the model performance was not impacted by a good amount except for neural networks which showed an improvement over the non-tuned model. The results from the tuned models were evaluated using the same evaluation metrics MSE, MAE and R^2 score, discussed in the results.

Chapter 4: Results & Interpretations

This chapter presents the findings from the analysis conducted using various machine learning models such as linear regression, ridge regression, lasso regression, random forest and neural networks. The study explores the performance of these models and evaluates the model performances with the benchmark. These models were tested on both normalized and original datasets. However, the normalized dataset was selected for further analysis because it includes both COVID and non-COVID year grades. This approach was chosen to minimize the impact of mark adjustments on the models and to avoid any potential model bias or learning issues, ensuring a more accurate and reliable study outcome. The results for the original dataset can be found in the Table A9.

The performance of each model was evaluated using MSE, MAE and R^2 score values, as discussed in the methodology section. This chapter is organized into four main sections: the first discusses the results of the benchmarking, followed by the performance of machine learning models, the impact of hyperparameters tuning on the models and finally the summary.

4.1 Benchmark model

A benchmark metric was established using a simple average of key predictors (as discussed in the methodology section 3.4.1) to provide a baseline for comparing more complex models. The benchmark results are as follows (refer Figure 4.1):

- **MSE:** The benchmark MSE of 3.19 represents the average squared difference between actual and predicted values. A model with a higher MSE than this may not perform better than a simple average-based prediction.
- **R²:** The benchmark R² of 0.28 indicates that 28% of the variance in the target variable is explained by this simple average. A model with an R² significantly lower than 0.28 is less effective in capturing data variance.
- **MAE:** The benchmark MAE of 1.41 measures the average absolute difference between predicted and actual values. A higher MAE than this suggests less accuracy on average.

4.2 Performance of Machine learning models

4.2.1 Neural Network

In this section, the performance of the neural network is discussed on the training and test datasets before hyperparameter tuning. The model was trained over 50 and 100 epochs, and its performance was evaluated using MSE, R², and MAE.

4.2.1.1 Evaluation of the training data

The performance of the neural network on the training dataset before hyperparameter tuning (refer Figure 4.1) shows promising results after 50 epochs. The model achieves an MSE of 2.00, which is lower than the benchmark MSE of 3.19, indicating higher accuracy in predictions. The R² value of 0.70 surpasses the benchmark R² of 0.28, demonstrating the model's capability to capture complex relationships in the data. Similarly, an MAE of 0.98, slightly better than the benchmark MAE of 1.41, further confirms the model's superior performance at this stage.

With 100 epochs, the model shows continued improvement, with an MSE decreasing to 1.84 and R² increasing to 0.72, both outperforming the benchmark. The MAE also decreases to 0.93, demonstrating that as training progresses, the model refines its predictions effectively. This consistent improvement across all metrics demonstrates that as the training progresses, the neural network continues to refine its predictions, making it more effective at capturing the complex patterns within the data that the benchmark method cannot.

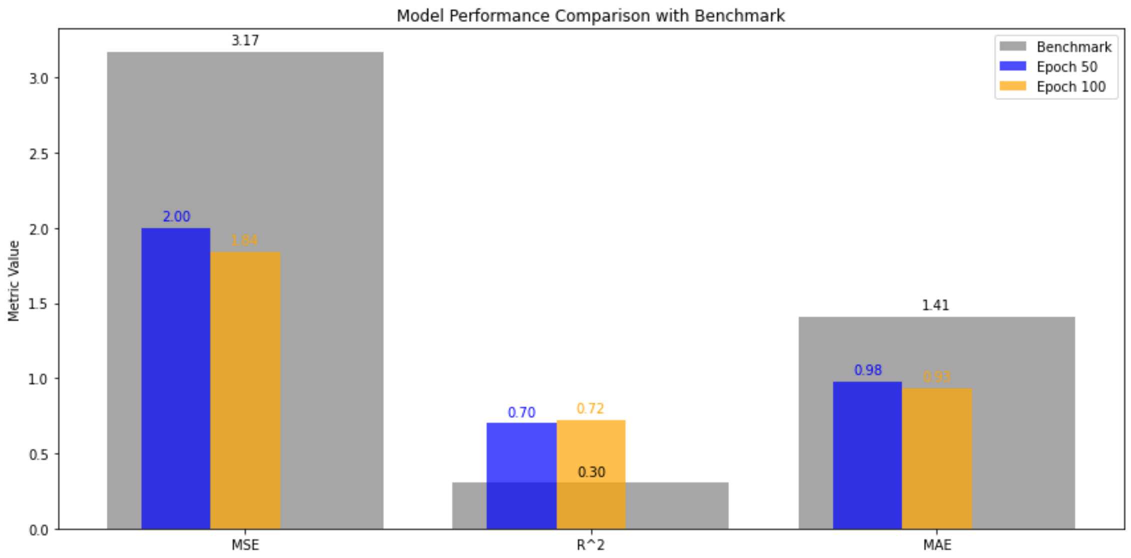


Figure 4.1 Performance metrics comparison of the neural network on training data

4.2.1.2 Evaluation of the test data

This section evaluates the neural network's performance on the test dataset compared to the benchmark and train metrics.

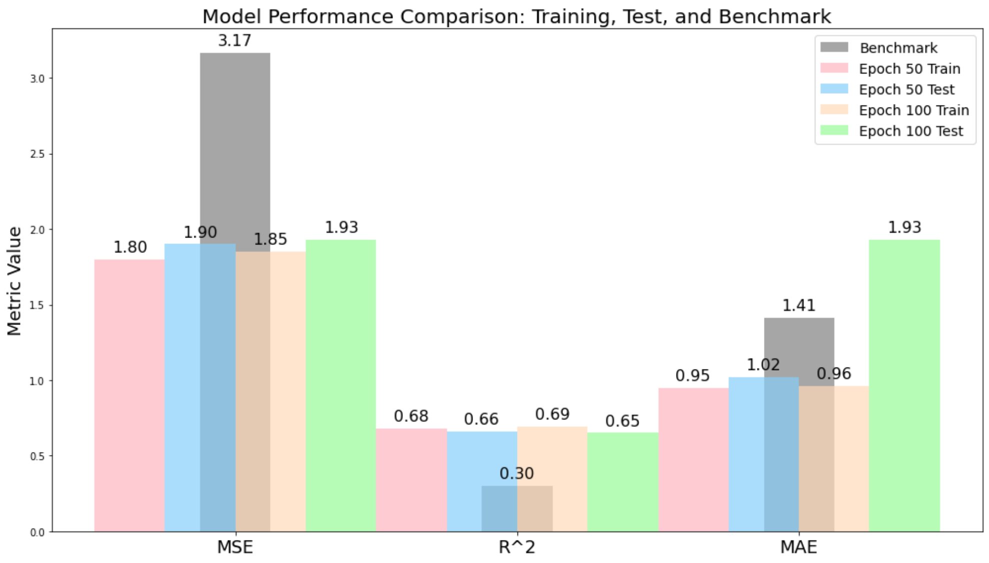
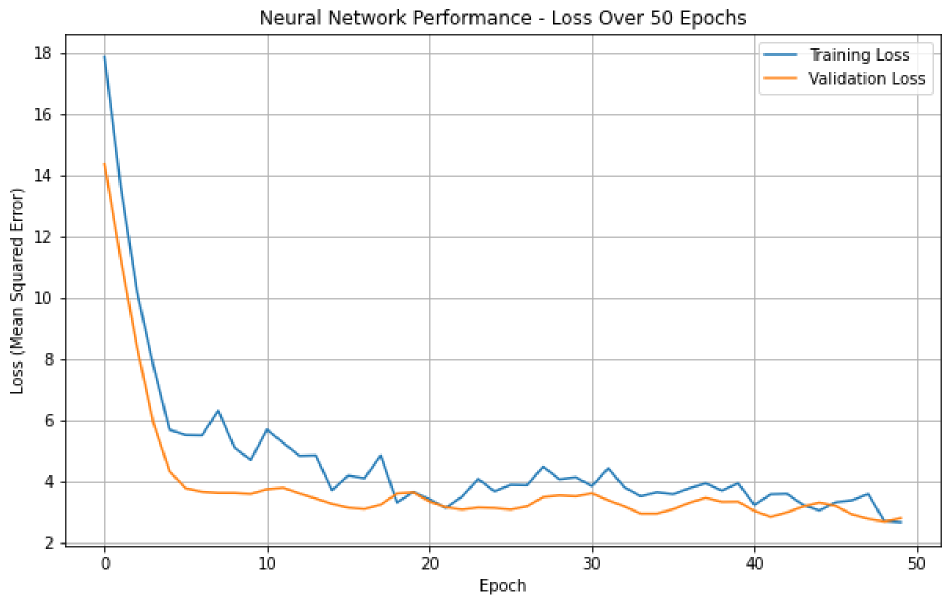


Figure 4.2 Performance metrics comparison of a neural network on test data

The neural network's performance on the test dataset compared to the benchmark and training metrics (refer Figure 4.2) shows that at 50 epochs, the test MSE of 1.90 is slightly higher than the training MSE of 1.80, as expected when encountering unseen data. MSE values are lower than the benchmark, confirming effective learning. The R² values also show strong performance, with the test R^2 of 0.66 close to the training R^2 of 0.68, both surpassing the benchmark but minimally.

However, by 100 epochs, although the training MSE improves to 1.85, the test MSE increases to 1.93, and the R^2 for the test data aligns with the benchmark at 0.28, indicating a plateau in generalization and the onset of overfitting. The MAE at 50 epochs (1.02) is slightly worse than the training MAE (0.95) but remains competitive. This substantial increase in test MAE indicates a decline in predictive accuracy on unseen data, further suggesting that the model is beginning to overfit as training progresses.

The neural network exhibits strong initial performance at 50 epochs as it outperforms the benchmark across key metrics while avoiding the overfitting observed at 100 epochs. Also, the loss curves (refer Figure 4.3) further support the conclusion that the model trained for 50 epochs is optimal. The training and validation losses both decrease sharply within the first few epochs and begin to stabilize around 50 epochs. At this point, the model shows minimal fluctuation in loss, indicating that it has learned the most relevant patterns without overfitting. Conversely, extending training to 100 epochs results in continued minor fluctuations without significant improvement, suggesting that the model is refining its fit to the training data at the expense of generalization.



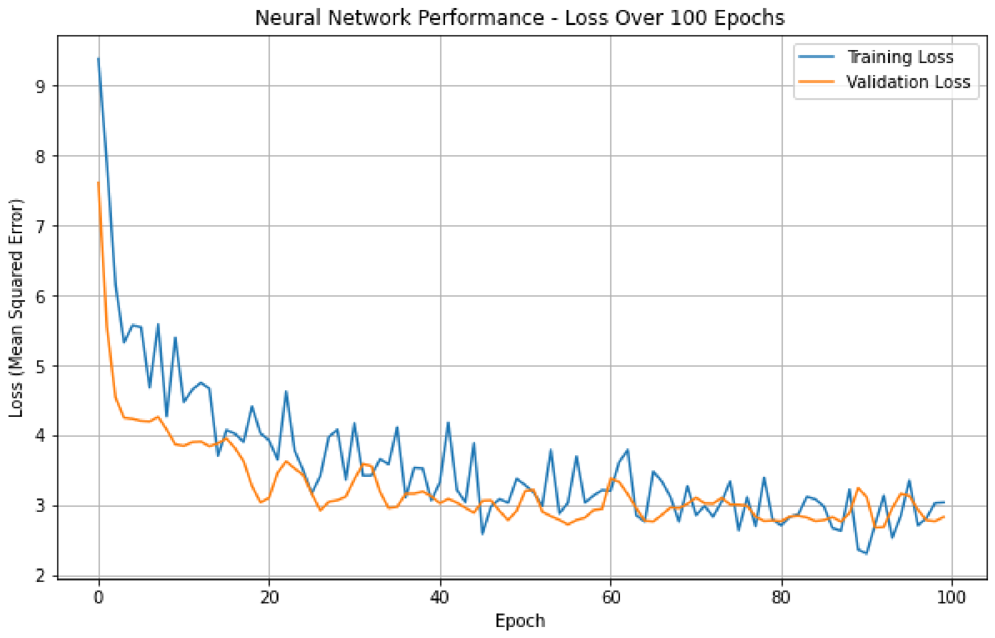


Figure 4.3 Comparison of loss curve for different epochs

4.2.2 Regression Models

This section contains the interpretation of all the regression models' performance on the train and test datasets.

4.2.2.1 Evaluation of the Training Data

Train MSE varies significantly across models and data types (refer Table 4.1). Linear Regression has the lowest Train MSE, particularly on the Combined Features dataset, indicating a tendency to overfit by capturing noise that doesn't generalize well. Lasso and Ridge Regression exhibit moderate Train MSE values (around 2.4 to 3.8), benefiting from regularisation that balances model complexity and data fit.

Train R² values reveal how well models explain variance in the training data. Linear Regression, despite high MSE, shows high R² values (around 0.9), especially on the Combined Features dataset. Lasso and Ridge Regression offer more balanced R² values (around 0.7 to 0.8. Random Forest outperforms with high R² values (close to 0.9) while maintaining low MSE and MAE.

Train MAE Analysis

Train MAE reflects the average prediction error magnitude. Linear Regression has a higher MAE, particularly on the Combined Features dataset, underscoring overfitting issues. Lasso and Ridge Regression show lower MAE (around 0.5 to 1.6), indicating fewer errors. Random Forest consistently delivers the lowest MAE (under 0.6), reinforcing its status as the most reliable model in terms of accuracy and overall performance.

Model	Data Type	Train MSE	Train R²	Train MAE
Linear Regression	Normalized	2.36	0.47	1.23
Lasso Regression	Normalized	3.81	0.15	1.60
Ridge Regression	Normalized	2.37	0.47	1.22
Random Forest	Normalized	0.43	0.90	0.52
Linear Regression	PCA_Normalised	2.42	0.46	1.23
Lasso Regression	PCA_Normalised	2.69	0.40	1.33
Ridge Regression	PCA_Normalised	2.42	0.46	1.23
Random Forest	PCA_Normalised	0.44	0.90	0.52
Linear Regression	Combined Features	0.00	1.00	0.00
Lasso Regression	Combined Features	3.81	0.15	1.60
Ridge Regression	Combined Features	0.59	0.87	0.58
Random Forest	Combined Features	0.45	0.90	0.53

Table 4.1 Performance metrics of regression model on training data

4.2.2.2 Evaluation on the test data

Test MSE Analysis:

The Test MSE values highlight significant variability in model performance. Linear Regression, particularly on the Combined Features dataset, exhibits a dramatic spike in Test MSE, exceeding 40, indicating severe overfitting where the model fails to generalize and captures noise instead (refer Table 4.2). In contrast, Lasso and Ridge Regression show more stable Test MSE values across all datasets, generally staying below 10, reflecting better generalization due to their regularisation techniques. Random Forest, while having higher Test MSE compared to its Train MSE on some datasets, generally maintains lower Test MSE than Linear Regression, indicating better generalization.

Test R² Analysis:

The Test R² values reveal how well models explain variance in the test data. Linear Regression, despite high training R² values, performs poorly on the test set, with R² values dropping significantly, even turning negative on the Combined Features dataset. This sharp drop to -10 is a strong indicator of overfitting. Lasso and Ridge Regression maintain more consistent Test R² values, generally hovering around zero, indicating less severe overfitting. Random Forest, although showing some decline in Test R² from training to testing, maintains positive R² values across datasets, indicating better generalization.

Test MAE Analysis:

Test MAE further illustrates prediction accuracy. Ridge Regression shows the highest Test MAE, especially on the Combined Features dataset. Lasso and Ridge Regression present more moderate Test MAE values, typically between 1.5 and 2.0, suggesting more accurate and consistent predictions. Random Forest maintains the lowest Test MAE values, generally under 2.0, indicating it produces the most accurate predictions on test data, aligning with its robustness observed in other metrics.

Model	Data Type	Test MSE	Test R²	Test MAE
Linear Regression	Normalized	3.74	0.11	1.52
Lasso Regression	Normalized	4.13	0.01	1.59
Ridge Regression	Normalized	3.84	0.08	1.56
Random Forest	Normalized	4.09	0.02	1.62
Linear Regression	PCA_Normalised	3.85	0.08	1.57
Lasso Regression	PCA_Normalised	3.89	0.07	1.59
Ridge Regression	PCA_Normalised	3.85	0.08	1.57
Random Forest	PCA_Normalised	3.90	0.07	1.60
Linear Regression	Combined Features	46.20	-10.03	5.36
Lasso Regression	Combined Features	4.13	0.01	1.59
Ridge Regression	Combined Features	8.28	-0.98	2.19
Random Forest	Combined Features	4.02	0.04	1.61

Table 4.2 Performance metrics of regression model on training data

The chart highlights how different models generalize to test data. Linear Regression is particularly prone to overfitting, especially on the Combined Features dataset, where it shows high Train R² but poor Test R², along with the highest Test MSE and MAE, indicating poor generalization. Lasso and Ridge Regression offer more balanced performance, with moderate discrepancies between training and test metrics, reflecting better generalization due to regularisation.

Random Forest emerges as the most robust model, achieving lower Test MSE and MAE values, and maintaining positive Test R² across datasets, although it also shows some signs of overfitting. Despite this, feature engineering applied in the *Combined Features* dataset did not significantly enhance generalization, as test metrics remained poor compared to the benchmark. This suggests that feature engineering, while potentially beneficial, does not always lead to better generalization and may introduce complexity that hinders performance.

4.3 Impact of hyperparameter tuning

The model tuning aimed to enhance performance by adjusting specific hyperparameters (as detailed in Methodology section 3.4.4). The performance evaluation was conducted using MSE, R², and MAE as the key metrics.

4.3.1 Neural Networks

This section presents a detailed analysis of the neural network's performance after hyperparameter tuning, comparing its effectiveness at 50 and 100 epochs against the pre-tuning results and benchmark metrics. The primary focus is on assessing the model's ability to generalize to unseen data and its overall improvement in accuracy.

1. Analysis on 50 Epochs

Performance on Training Data

After hyperparameter tuning, the model exhibits significant improvements on the training data at 50 epochs, with the MSE decreasing from 1.80 (pre-tuning) to 1.78, R² increasing from 0.68 to 0.74, and MAE decreasing from 0.95 to 0.92. These results indicate that the model has become more accurate and better at capturing patterns within the training data. The tuned model not only surpasses its pre-tuning performance but also outperforms the benchmark (MSE 3.19, R² 0.28, MAE 1.41), showcasing its superior fit to the training data (refer Figure 4.4).

Performance on Test Data

On the test data, the model also shows clear improvements after hyperparameter tuning at 50 epochs. The MSE decreases from 1.90 (pre-tuning) to 1.56, R² increases from 0.66 to 0.72, and MAE decreases from 1.02 to 0.89. These enhancements indicate that the model has become more effective at generalizing to unseen data, surpassing the benchmark (MSE 3.19, R² 0.28, MAE 1.41) across all metrics.

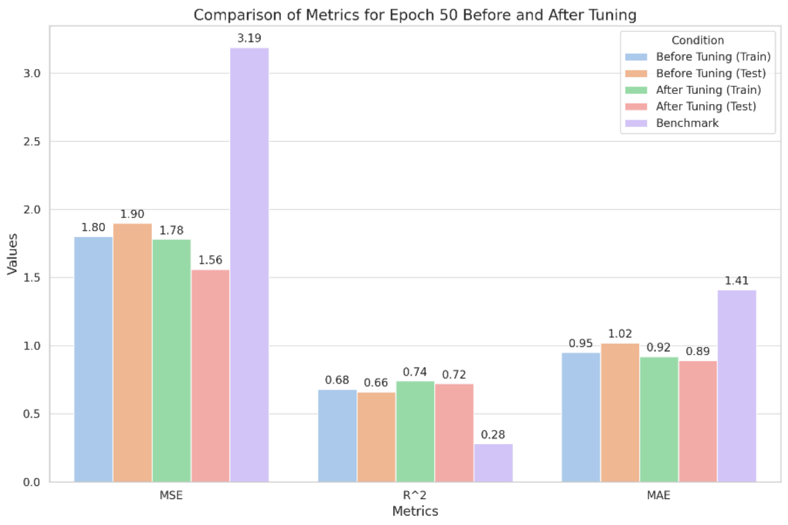


Figure 4.4 Comparison of metrics before and after tuning, Epochs = 50

2. Analysis on 100 Epochs

Performance on Train Data

At 100 epochs, the model's performance on the training data continues to improve slightly from pre-tuning, with the MSE decreasing from 1.85 to 1.82, R² improving from 0.69 to 0.73, and a minor decrease in MAE from 0.96 to 0.94. These results suggest that while the model's fit remains strong, the adjustments introduced during tuning, such as regularisation, are helping to prevent overfitting, even if they slightly impact the tightness of the fit to the training data.

Performance on test data

At 100 epochs, the model's test performance remains strong but shows signs of diminishing returns. The MSE slightly increases to 1.66, R² decreases to 0.70, and MAE rises marginally to 0.90. While these metrics are still better than the pre-tuning results and the benchmark, the slight increase in errors suggests that extending training further may lead to overfitting, making the improvements less impactful than at 50 epochs (refer Figure 4.5).

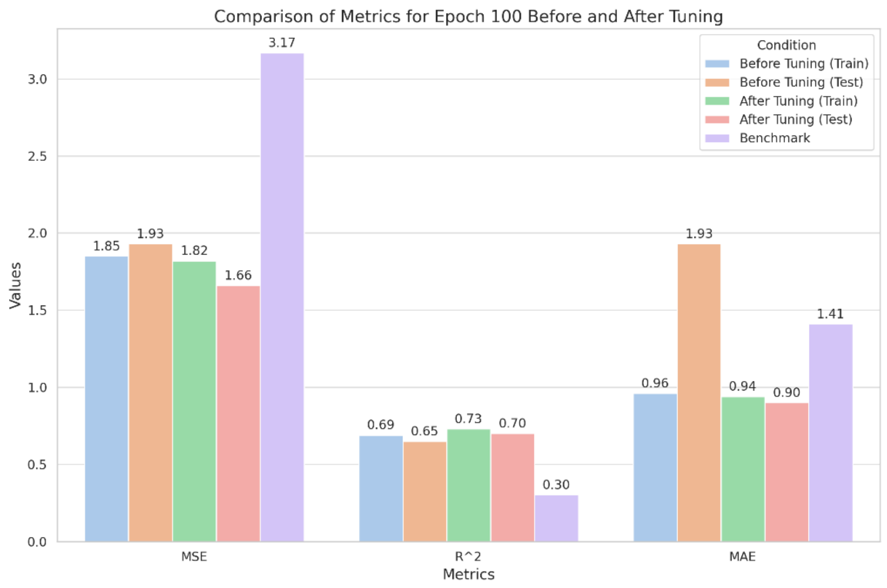


Figure 4.5 Comparison of metrics before and after tuning, Epochs = 100

4.3.2 Regression Models

Performance on Train Data

Table 4.3 shows that Random Forest consistently outperforms the linear models (Linear Regression, Lasso, and Ridge Regression) across all preprocessing techniques. With Normalized and PCA Transformed data, Random Forest achieves the lowest MSE and highest R^2, indicating superior predictive power. Linear models exhibit moderate performance, with Linear Regression explaining about 47% of the variance, while Lasso and Ridge have similar but slightly varied results. PCA does not notably improve performance over normalized data for linear models.

Feature Engineered data enhances model performance, especially for Random Forest, which achieves the best results overall. Linear Regression reports perfect metrics with feature-engineered data, suggesting possible issues like overfitting or data leakage. Ridge Regression generally performs better than Lasso and Linear Regression. Random Forest with feature-engineered data is the most effective combination, but the perfect Linear Regression results need further investigation.

Data Type	Model	Train MSE	Train R^2	Train MAE
Normalized	Linear Regression	2.36	0.47	1.23

Normalized	Lasso	2.43	0.46	1.25
Normalized	Ridge	2.42	0.46	1.24
Normalized	Random Forest	1.08	0.76	0.83
PCA Transformed	Linear Regression	2.42	0.46	1.23
PCA Transformed	Lasso	2.46	0.45	1.25
PCA Transformed	Ridge	2.44	0.46	1.25
PCA Transformed	Random Forest	0.96	0.79	0.77
Feature Engineered	Linear Regression	0.00	1.00	0.00
Feature Engineered	Lasso	2.35	0.48	1.23
Feature Engineered	Ridge	1.77	0.60	1.07
Feature Engineered	Random Forest	0.80	0.82	0.69

Table 4.3 Performance of models after hyperparameter tuning on train data

Performance on Test Data

When comparing model performance across data types, clear patterns emerge. Linear Regression excels with normalized data, achieving the lowest Test MSE of 3.74, MAE of 1.52, and a decent R^2 of 0.11. In contrast, Random Forest performs poorly with normalized data, showing the highest MSE 3.93, lowest R^2 0.06, and highest MAE 1.59, suggesting it struggles with normalized features (refer Table 4.4).

With PCA-transformed data, Linear Regression's performance slightly declines, with an MSE of 3.85, MAE of 1.57, and R^2 of 0.08. Lasso and Ridge show similar results but lag behind Linear Regression in R^2. Random Forest continues to underperform, with the highest MSE 3.97, lowest R^2 0.05, and highest MAE 1.60. For feature-engineered data, Linear Regression's performance drops dramatically, with an MSE of 46.20 and R^2 of -10.03, indicating poor generalization. Lasso fares better but still underperforms compared to other data types, while Ridge has the worst performance with a high MSE of 4.40 and negative R^2. Random Forest performs better than Linear Regression but remains less effective overall. This highlights the importance of choosing the right model and preprocessing technique based on the data characteristics.

Data Type	Model	Test MSE	Test R^2	Test MAE
Normalized	Linear Regression	3.74	0.11	1.52
Normalized	Lasso	3.86	0.08	1.58
Normalized	Ridge	3.89	0.07	1.58
Normalized	Random Forest	3.93	0.06	1.59
PCA Transformed	Linear Regression	3.85	0.08	1.57
PCA Transformed	Lasso	3.92	0.06	1.58
PCA Transformed	Ridge	3.90	0.07	1.58

PCA Transformed	Random Forest	3.97	0.05	1.60
Feature Engineered	Linear Regression	46.20	-10.03	5.36
Feature Engineered	Lasso	3.87	0.07	1.60
Feature Engineered	Ridge	4.40	-0.05	1.70
Feature Engineered	Random Forest	3.97	0.05	1.58

Table 4.4 Performance of models after hyperparameter tuning on train data

4.4 Summary of Findings

The analysis revealed that the neural network, particularly after 50 epochs of training, outperformed all other models including the benchmark, achieving an MSE of 1.56, R^2 of 0.72, and MAE of 0.89 after tuning on the normalized dataset.

Among the regression models, Random Forest demonstrated the best overall performance, maintaining lower error rates and stronger generalization compared to Linear Regression, which was prone to severe overfitting. Lasso and Ridge Regression showed more balanced results due to regularisation but did not match the robustness of Random Forest. Ultimately, the neural network was selected as the final model due to its superior accuracy and ability to generalize effectively, especially at 50 epochs.

Chapter 5: Conclusion

This dissertation investigated the effectiveness of various machine learning models in predicting student grades within the context of GCSE-level engineering subjects at the WMG Academy. The research was motivated by the need to improve grade prediction accuracy in niche subjects like engineering, which presents unique challenges due to its interdisciplinary nature.

5.1 Key Findings

1. Neural Networks Performance:

Neural networks were found to be the most effective model for predicting engineering grades. After 50 epochs of training, the neural network achieved the best performance metrics as compared to the simple benchmark, including a MSE of 1.56, an R^2 score of 0.72, and a MAE of 0.89. These results were significantly better than those of other models and the benchmark, highlighting the neural network's strong generalisation capabilities and suitability for complex, interdisciplinary data. Further training to 100 epochs showed diminishing returns, with slight increases in test errors, indicating that overfitting might begin if training is extended beyond optimal points.

2. Regression Models Performance:

Among regression models, Random Forest demonstrated the best overall performance with lower error rates and stronger generalisation compared to Linear Regression, Lasso, and Ridge Regression. However, a notable issue observed across all regression models was the significant difference between training and test errors as compared to benchmark. While the models performed well on the training data, their performance declined considerably on the test data, indicating overfitting is a common issue when working with small datasets.

Specifically, Linear Regression exhibited severe overfitting, as it performed exceptionally well on the training data but showed increase in error rates on the test data. This decline in performance suggests that the model was too closely fitted to the training data, capturing noise that did not generalise to unseen data. Lasso and Ridge Regression, though more stable due to regularisation, also exhibited signs of overfitting, with test errors significantly higher than training errors.

The Random Forest model, despite being more robust, still faced challenges in generalising from the training data to the test data, further highlighting the limitations imposed by the small dataset.

3. Impact of Data Preprocessing and Feature Engineering

The study highlighted the critical role of data preprocessing and feature engineering in enhancing model performance. Techniques such as standardisation, normalisation, and the creation of polynomial features and log transformations were applied. Although these methods improved the model's performance to some extent, they did not lead to substantial gains across all models. This suggests that while preprocessing is essential, it must be carefully tailored to the specific characteristics of the data and the machine learning task.

PCA was employed to address multicollinearity, but it did not significantly enhance model performance, indicating that dimensionality reduction techniques might have limited effectiveness in this context.

4. Hyperparameter Tuning

Hyperparameter tuning was crucial in optimising the models, particularly the neural network. Grid search with cross-validation was used to fine-tune the parameters, which led to improved performance metrics. For the neural network, the tuning process helped reduce overfitting and enhanced the model's ability to generalise to unseen data. This underscores the importance of careful parameter adjustment in predictive modelling.

5. Benchmarking

The models were benchmarked against a simple average prediction based on key predictors. The neural network, especially after tuning, outperformed this benchmark, demonstrating that advanced machine learning models can provide more accurate predictions than basic statistical methods, even in complex and interdisciplinary fields like engineering.

5.2 Implications and Future Research

The findings of this dissertation have significant implications for educational practice, particularly in subjects where traditional predictive models struggle to capture the full complexity of student performance. The research suggests that neural networks, with appropriate tuning and preprocessing, are well-suited to predicting outcomes in interdisciplinary subjects like engineering as compared to the benchmark which was simple average benchmark in this project.

However, the study also indicates that there is room for improvement, especially in the areas of data preprocessing and feature engineering for engineering subjects in GCSE level. Future research could explore the use of more advanced neural network architectures and investigate the potential of incorporating additional data features that capture the unique aspects of engineering education.

The overfitting observed in regression models due to the small dataset highlights the importance of obtaining larger datasets to train more generalised models. Additionally, future studies could explore techniques to mitigate overfitting, such as advanced regularisation methods and more robust feature selection techniques.

Moreover, the impact of the COVID-19 pandemic on student performance was considered in this study, highlighting the importance of accounting for external factors in predictive modelling.

Future studies could further examine the long-term effects of such disruptions on educational outcomes and refine predictive models to account for these variables more effectively. It is recommended that WMG academy could implement a neural network to predict the student grades for engineering subjects for GCSE rather than using a simple average benchmark.

In conclusion, this dissertation contributes to the growing body of research on educational data mining and predictive modelling by demonstrating the potential of advanced machine learning models in predicting student performance in niche subjects. The findings underscore the need for continued innovation and refinement in this field to better support educational outcomes across diverse disciplines.

References

AGARWAL, S. & BEHERA, S. R. 2023. Geo-visualizing the impact of the COVID-19 pandemic on students' learning outcomes: Evidence from grade 5 students. *Environment and Planning B: Urban Analytics and City Science,* 50**,** 2322-2325.

ALDOWAH, H., AL-SAMARRAIE, H. & FAUZY, W. M. 2019. Educational data mining and learning analytics for 21st century higher education: A review and synthesis. *Telematics and Informatics,* 37**,** 13-49.

ALSHAREEF, F., ALHAKAMI, H., ALSUBAIT, T. & BAZ, A. 2020. Educational data mining applications and techniques. *International Journal of Advanced Computer Science and Applications,* 11.

ANDERS, J., DILNOT, C., MACMILLAN, L. & WYNESS, G. 2020. Grade Expectations: How well can we predict future grades based on past performance?

AZEVEDO, J. P., HASAN, A., GOLDEMBERG, D., GEVEN, K. & IQBAL, S. A. 2021. Simulating the potential impacts of COVID-19 school closures on schooling and learning outcomes: A set of global estimates. *The World Bank Research Observer,* 36**,** 1-40.

BAASHAR, Y., ALKAWSI, G., MUSTAFA, A., ALKAHTANI, A. A., ALSARIERA, Y. A., ALI, A. Q., HASHIM, W. & TIONG, S. K. 2022. Toward predicting student's academic performance using artificial neural networks (ANNs). *Applied Sciences,* 12**,** 1289.

BAKER, R. S., MARTIN, T. & ROSSI, L. M. 2016. Educational data mining and learning analytics. *The Wiley handbook of cognition and assessment: Frameworks, methodologies, and applications***,** 379-396.

CHEN, L., CHEN, P. & LIN, Z. 2020. Artificial intelligence in education: A review. *Ieee Access,* 8**,** 75264-75278.

DENES, G. 2023. A case study of using AI for General Certificate of Secondary Education (GCSE) grade prediction in a selective independent school in England. *Computers and Education: Artificial Intelligence,* 4**,** 100129.

DI PIETRO, G., BIAGI, F., COSTA, P., KARPIŃSKI, Z. & MAZZA, J. 2020. *The likely impact of COVID-19 on education: Reflections based on the existing literature and recent international datasets*, Publications Office of the European Union Luxembourg.

HAMMERSTEIN, S., KÖNIG, C., DREISÖRNER, T. & FREY, A. 2021. Effects of COVID-19-related school closures on student achievement-a systematic review. *Frontiers in psychology,* 12**,** 746289.

HEATON, J. An empirical analysis of feature engineering for predictive modeling. SoutheastCon 2016, 2016. IEEE, 1-6.

HIDALGO-CAMACHO, C., ESCUDERO, G. I., VILLACÍS, W. & VARELA, K. 2021. The Effects of Online Learning on EFL Students' Academic Achievement during Coronavirus Disease Pandemic. *European Journal of Educational Research,* 10**,** 1867-1879.

KUHFELD, M., SOLAND, J., TARASAWA, B., JOHNSON, A., RUZEK, E. & LIU, J. 2020. Projecting the potential impact of COVID-19 school closures on academic achievement. *Educational Researcher,* 49**,** 549-565.

LESTARI, L. D. & WACHIDAH, K. 2023. Student Anxiety and Math Learning Outcomes in Grade 4 During Covid-19. *Academia Open,* 8**,** 10.21070/acopen. 8.2023. 4761-10.21070/acopen. 8.2023. 4761.

LESTARI, N. & SYAIMI, K. U. 2021. Impact of Covid-19 on Student Learning Outcomes at SDIT Alfarabi Tanjung Selamat. *Sensei International Journal of Education and Linguistic,* 1**,** 321-327.

MIRANDA, E., ARYUNI, M., RAHMAWATI, M. I., HIERERRA, S. E. & SANO, D. 2023. Machine learning's model-agnostic interpretability on The Prediction of Students' Academic Performance in Video-Conference-Assisted Online Learning During the Covid-19 Pandemic.

OFQUAL 2019. Referencing the Qualifications

Frameworks of England and Northern

Ireland to the European Qualifications

Framework. 192.

OJAJUNI, O., AYENI, F., AKODU, O., EKANOYE, F., ADEWOLE, S., AYO, T., MISRA, S. & MBARIKA, V. Predicting student academic performance using machine learning. Computational Science and Its Applications–ICCSA 2021: 21st International Conference, Cagliari, Italy, September 13–16, 2021, Proceedings, Part IX 21, 2021. Springer, 481-491.

RAHMAN, A. 2021. The impact of Covid-19 pandemic on students' learning outcome in higher education. *Al-Ishlah: Jurnal Pendidikan,* 13**,** 1425-1431.

RODRÍGUEZ-HERNÁNDEZ, C. F., MUSSO, M., KYNDT, E. & CASCALLAR, E. 2021. Artificial neural networks in academic performance prediction: Systematic implementation and predictor evaluation. *Computers and Education: Artificial Intelligence,* 2**,** 100018.

SIEMENS, G. & BAKER, R. S. D. Learning analytics and educational data mining: towards communication and collaboration. Proceedings of the 2nd international conference on learning analytics and knowledge, 2012. 252-254.

SUHARIYONO, S. & RETNAWATI, H. 2022. The Influence of Online Learning on Physics Learning Outcomes during the Covid-19 Pandemic. *Radiasi: Jurnal Berkala Pendidikan Fisika,* 15**,** 35-42.

TORADMAL, M. B., MEHTA, M. & MEHENDALE, S. 2023. Machine Learning Approaches for Educational Data Mining. *Inventive Systems and Control: Proceedings of ICISC 2023.* Springer.

WU, J., KUAN, G., LOU, H., HU, X., MASRI, M. N., SABO, A. & KUEH, Y. C. 2023. The impact of COVID-19 on students' anxiety and its clarification: a systematic review. *Frontiers in Psychology,* 14**,** 1134703.

XU, C., WU, C.-F., XU, D.-D., LU, W.-Q. & WANG, K.-Y. 2022. Challenges to Student Interdisciplinary Learning Effectiveness: An Empirical Case Study. *Journal of Intelligence,* 10**,** 88.

ZHANG, L. & LI, K. F. Education analytics: Challenges and approaches. 2018 32nd international conference on advanced information networking and applications workshops (WAINA), 2018. IEEE, 193-198.

Appendix

Appendix 1 - Tables

Variable	Data Type	Value range
Grade	Decimal	0-8.5
COVID	Binary	0-1
Metric1	Binary	0-1
Metric2	Integer	69-135
Metric3	Integer	1-9
Metric4	Integer	1-9
Metric5	Integer	1-9
Metric6	Integer	5-60
Metric7	Integer	0-18
Metric8	Integer	0-18
Metric9	Integer	0-17
Metric10	Integer	0-14
Metric11	Integer	2-99
Metric12	Decimal	2.3-8.0
Metric13	Integer	69-141
Metric14	Integer	1-9
Metric15	Integer	1-61
Metric16	Integer	0-15
Metric17	Integer	0-15
Metric18	Integer	0-7
Metric19	Integer	0-21
Metric20	Integer	0-4
Metric21	Integer	0-5
Metric22	Integer	0-9
Metric23	Integer	0-26
Metric24	Integer	0-9
Metric25	Integer	0-21
Metric26	Integer	2-99
Metric27	Decimal	1-9.3

Table A1. Data dictionary of the variables

	count	mean	std	min	25%	50%	75%	max
Grade	314	4.566879	2.134342	0	3	4	5.5	8.5
COVID	314	0.375796	0.485101	0	0	0	1	1
Metric1	314	0.770701	0.421053	0	1	1	1	1
Metric2	314	93.27389	14.49253	69	83	93	103.75	135
Metric3	314	4.105096	1.937961	1	3	4	5.75	9

Metric4	314	4.343949	1.877097	1	3	4	6	9
Metric5	314	4.05414	2.020723	1	3	4	6	9
Metric6	314	33.37898	11.79997	5	25	34	42	60
Metric7	314	9.458599	4.279366	0	7	10	13	18
Metric8	314	9.496815	3.427243	0	7	10	12	18
Metric9	314	8.799363	3.819556	0	6	9	12	17
Metric10	314	5.624204	2.944457	0	4	5	8	14
Metric11	314	37.18471	27.56354	2	13	32	59.5	99
Metric12	314	4.444904	1.305159	2.3	3.5	4.4	5.475	8
Metric13	314	101.6274	13.71706	69	93	102	111	141
Metric14	314	5.213376	1.804063	1	4	5	6	9
Metric15	314	25.6051	13.75749	1	15	23.5	34.75	61
Metric16	314	6.156051	3.505976	0	4	6	8	15
Metric17	314	5.875796	3.487154	0	3	5	8	15
Metric18	314	3.044586	2.144445	0	1	3	5	7
Metric19	314	5.840764	4.585725	0	3	5	8	21
Metric20	314	1.687898	1.165826	0	1	1	3	4
Metric21	314	3	1.445495	0	2	3	4	5
Metric22	314	3.417197	2.323891	0	2	3	5	9
Metric23	314	11.65605	6.103174	0	7	11	16	26
Metric24	314	1.751592	2.019572	0	0	1	2	9
Metric25	314	8.780255	4.803432	0	5	8	12	21
Metric26	314	53.21338	27.12029	2	32	55	77	99
Metric27	314	4.982484	1.745129	1	3.9	5.1	6.2	9.3

Table A2. Descriptive statistics for the dataset

Variables	Correlation Coefficient
Metric2 and Metric3	0.99
Metric2 and Metric4	0.9
Metric2 and Metric5	0.89
Metric2 and Metric6	0.99
Metric2 and Metric8	0.82
Metric2 and Metric9	0.83
Metric2 and Metric11	0.98
Metric2 and Metric12	1
Metric3 and Metric4	0.9
Metric3 and Metric5	0.87
Metric3 and Metric6	0.97
Metric3 and Metric8	0.82
Metric3 and Metric9	0.82
Metric3 and Metric11	0.97
Metric3 and Metric12	0.99
Metric4 and Metric6	0.9

Metric4 and Metric7	0.88
Metric4 and Metric8	0.84
Metric4 and Metric11	0.88
Metric4 and Metric12	0.9
Metric5 and Metric6	0.88
Metric5 and Metric9	0.9
Metric5 and Metric10	0.87
Metric5 and Metric11	0.89
Metric5 and Metric12	0.89
Metric6 and Metric7	0.8
Metric6 and Metric8	0.83
Metric6 and Metric9	0.84
Metric6 and Metric11	0.97
Metric6 and Metric12	0.99
Metric8 and Metric11	0.81
Metric8 and Metric12	0.82
Metric9 and Metric11	0.81
Metric9 and Metric12	0.83
Metric11 and Metric12	0.98
Metric13 and Metric14	0.99
Metric13 and Metric15	0.97
Metric13 and Metric16	0.87
Metric13 and Metric17	0.86
Metric13 and Metric18	0.83
Metric13 and Metric19	0.86
Metric13 and Metric22	0.83
Metric13 and Metric23	0.93
Metric13 and Metric25	0.9
Metric13 and Metric26	0.98
Metric13 and Metric27	1
Metric14 and Metric15	0.96
Metric14 and Metric16	0.85
Metric14 and Metric17	0.86
Metric14 and Metric18	0.83
Metric14 and Metric19	0.84
Metric14 and Metric22	0.82
Metric14 and Metric23	0.92
Metric14 and Metric25	0.89
Metric14 and Metric26	0.97
Metric14 and Metric27	0.99
Metric15 and Metric16	0.89
Metric15 and Metric17	0.9
Metric15 and Metric18	0.85

Metric15 and Metric19	0.9
Metric15 and Metric22	0.86
Metric15 and Metric23	0.95
Metric15 and Metric25	0.92
Metric15 and Metric26	0.96
Metric15 and Metric27	0.97
Metric16 and Metric22	0.92
Metric16 and Metric23	0.85
Metric16 and Metric26	0.86
Metric16 and Metric27	0.87
Metric17 and Metric23	0.83
Metric17 and Metric25	0.84
Metric17 and Metric26	0.85
Metric17 and Metric27	0.86
Metric18 and Metric23	0.83
Metric18 and Metric26	0.85
Metric18 and Metric27	0.83
Metric19 and Metric23	0.87
Metric19 and Metric26	0.82
Metric19 and Metric27	0.85
Metric22 and Metric23	0.8
Metric22 and Metric26	0.83
Metric22 and Metric27	0.83
Metric23 and Metric26	0.93
Metric23 and Metric27	0.93
Metric25 and Metric26	0.89
Metric25 and Metric27	0.9
Metric26 and Metric27	0.99

Table A3. Multicollinearity coefficients of variables above threshold (>0.8)

Feature	VIF
COVID	1.089429
Metric1	1.126557
Metric2	561.7465
Metric3	46.85734
Metric4	33.70436
Metric5	30.55599
Metric6	inf
Metric7	inf
Metric8	inf
Metric9	inf
Metric10	inf
Metric11	39.38334

Metric12	492.5499
Metric13	703.5161
Metric14	39.47522
Metric15	inf
Metric16	inf
Metric17	inf
Metric18	inf
Metric19	inf
Metric20	inf
Metric21	inf
Metric22	inf
Metric23	inf
Metric24	inf
Metric25	inf
Metric26	67.46592
Metric27	868.4445
Grade	1.089766

Table A4. VIF score of the variables

VIF Range	Interpretation
VIF < 5	Low multicollinearity; generally acceptable.
5 ≤ VIF < 10	Moderate multicollinearity; may need attention.
VIF ≥ 10	High multicollinearity; problematic, should be addressed.
VIF = Infinity	Perfect multicollinearity: severe issue, requires action.

Table A5. Interpretation of VIF Scores

Variable	PC1	PC2	PC3	PC4	PC5
COVID	-3.2E-17	-1.4E-16	-1.1E-15	1	3.43E-16
Metric1	-0.01774	-0.08919	0.837454	8.78E-16	0.334956
Metric2	0.210903	0.249794	0.019155	-2.2E-16	0.011703
Metric3	0.208341	0.250649	0.022917	1.28E-16	-0.0006
Metric4	0.182944	0.265707	0.191176	4.06E-16	-0.28581
Metric5	0.195167	0.192011	-0.17876	-3.8E-16	0.329933
Metric6	0.210123	0.250447	0.03018	1.38E-16	0.00975
Metric7	0.153309	0.247145	0.313046	3.86E-16	-0.32276
Metric8	0.177716	0.20632	0.005826	9.92E-17	-0.15499
Metric9	0.184047	0.168513	-0.14257	-1.7E-16	0.307856
Metric10	0.171075	0.183285	-0.15305	-2.3E-17	0.286132
Metric11	0.208301	0.245495	-0.01149	8.17E-17	0.007944
Metric12	0.211031	0.248722	0.017648	1.38E-16	0.015832
Metric13	0.222118	-0.1778	0.011999	-4.1E-17	-0.01179
Metric14	0.218624	-0.18238	0.014702	1.13E-16	-0.01002

Metric15	0.222871	-0.18597	-0.01234	-1.4E-16	-0.0498
Metric16	0.202352	-0.14043	0.107857	-7.2E-17	-0.07646
Metric17	0.196301	-0.18488	-0.04606	-1E-16	-0.08789
Metric18	0.192473	-0.15875	0.06678	1.57E-16	0.006926
Metric19	0.193608	-0.17453	-0.11346	2.95E-17	-0.2199
Metric20	0.117577	-0.14903	-0.04356	-4.2E-16	0.439978
Metric21	0.161225	-0.0725	0.029752	-1.1E-16	0.257855
Metric22	0.189993	-0.17108	0.099612	1.1E-16	-0.12204
Metric23	0.211678	-0.1704	0.034291	6.45E-17	-0.10031
Metric24	0.164974	-0.16122	-0.19331	-1.8E-16	-0.16925
Metric25	0.208105	-0.16555	-0.04576	-3E-17	0.115774
Metric26	0.220784	-0.17049	0.030145	-5.6E-18	0.008739
Metric27	0.221658	-0.17775	0.017027	3.18E-16	-0.00562

Table A6. Loading coefficients of variables with PCs (PC1 – PC5)

Variable	PC6	PC7	PC8	PC9	PC10
COVID	2.32E-16	-5.8E-17	9.08E-17	-5.4E-17	7.39E-17
Metric1	-0.34097	0.022382	0.165326	0.121602	-0.05636
Metric2	-0.01632	-0.01961	0.029848	-0.00174	0.000459
Metric3	-0.01946	-0.00739	0.037321	0.014825	0.010923
Metric4	0.158624	0.142622	0.075709	0.01856	-0.08142
Metric5	-0.18457	-0.16256	-0.03898	-0.02183	0.091586
Metric6	0.019621	-0.00913	0.01988	0.013698	0.035479
Metric7	0.221954	0.001237	0.172649	-0.3936	0.334401
Metric8	0.111684	0.264135	-0.09486	0.533088	-0.48009
Metric9	-0.06219	-0.16829	-0.03369	0.351057	0.553574
Metric10	-0.29102	-0.12826	-0.01539	-0.45302	-0.49602
Metric11	-0.05297	-0.02047	0.032678	0.013049	-0.04232
Metric12	-0.01241	-0.01414	0.025378	0.014925	0.012291
Metric13	0.046011	0.02039	-0.0192	-0.01375	0.042563
Metric14	0.037149	0.022456	-0.0352	0.003917	0.062357
Metric15	-0.03305	-0.01205	0.005873	-0.01092	-0.00518
Metric16	0.076418	-0.35289	-0.20811	-0.09175	-0.03852
Metric17	-0.191	0.091139	0.192757	0.01085	0.005486
Metric18	-0.01404	-0.13217	-0.21664	0.308883	-0.05322
Metric19	-0.18805	0.022602	0.135522	-0.03161	0.037538
Metric20	0.641771	-0.03267	0.517839	0.004615	-0.16131
Metric21	0.062099	0.67316	-0.43553	-0.27147	0.121291
Metric22	0.086509	-0.36854	-0.17824	-0.16051	-0.16489
Metric23	-0.00771	-0.11203	-0.11453	0.076587	0.017131
Metric24	-0.39058	0.203662	0.509927	0.025941	0.029097
Metric25	0.03815	0.201021	0.033785	-0.06216	0.030985
Metric26	0.081556	0.01448	-0.09139	0.027655	0.014532

Metric27	0.061889	0.022108	-0.03453	-0.00972	0.04447

Table A7. Loading coefficients of variables with PCs (PC6 – PC10)

Variable	Skewness
Metric24	1.570335
Metric19	1.202789
Metric17	0.59748
Metric15	0.595118
COVID	0.515357
Metric22	0.49174
Metric11	0.47259
Metric25	0.471279
Metric16	0.392378
Metric5	0.360183
Metric23	0.327999
Metric20	0.294552
Metric18	0.28808
Metric2	0.268155
Metric10	0.245072
Metric3	0.217967
Metric4	0.18765
Metric12	0.163522
Metric13	0.06789
Metric8	-0.04886
Metric14	-0.05039
Metric27	-0.07175
Metric6	-0.09745
Metric26	-0.13231
Metric9	-0.1912
Metric21	-0.2092
Metric7	-0.32139
Metric1	-1.29004

Table A8. Skew coefficient of variables in normalized data

Model	Data Type	Test MSE	Test R^2	Test MAE
Linear Regression	original Data	3.738037252	0.126777425	1.523204982
Lasso Regression	original Data	4.24745196	0.007775823	1.638927879
Ridge Regression	original Data	3.841242512	0.102668205	1.560430349
Random Forest	original Data	4.312771726	-0.007483173	1.654007937
Linear Regression	original Data PCA	3.844039607	0.102014791	1.566475314
Lasso Regression	original Data PCA	3.983242954	0.069496253	1.577754657
Ridge Regression	original Data PCA	3.844278748	0.101958926	1.56648871
Random Forest	original Data PCA	4.17886756	0.023797452	1.629087302

Linear Regression	Combined Features	54.37594414	-11.7024689	5.412802342
Lasso Regression	Combined Features	4.104760225	0.041109264	1.595843509
Ridge Regression	Combined Features	5.15590187	-0.204442225	1.79099908
Random Forest	Combined Features	4.054748313	0.052792275	1.575198413

Table A9. Performance metrics of the original dataset

Appendix 2 – Graphs

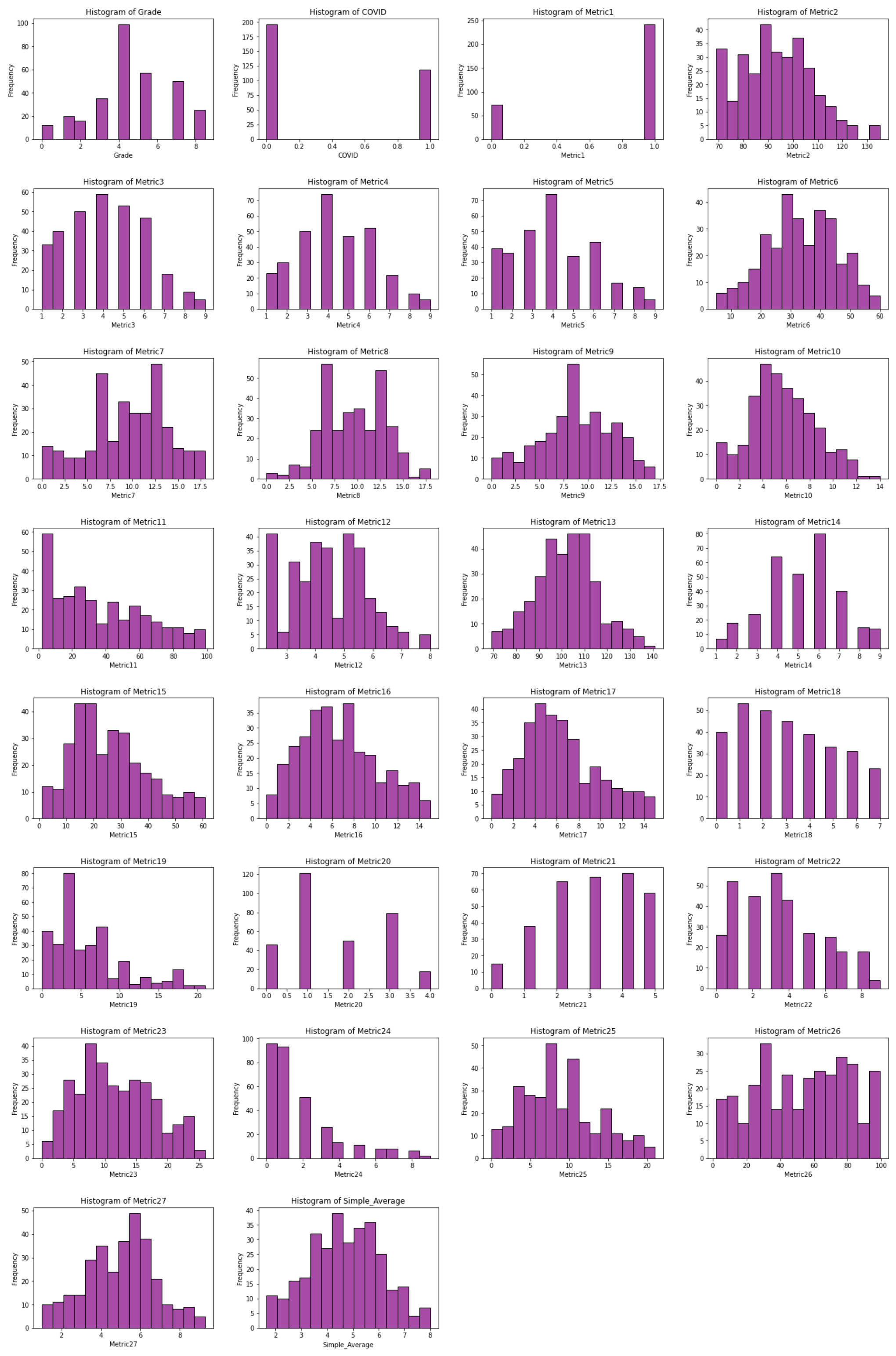


Figure A1. Distribution of all the metrics

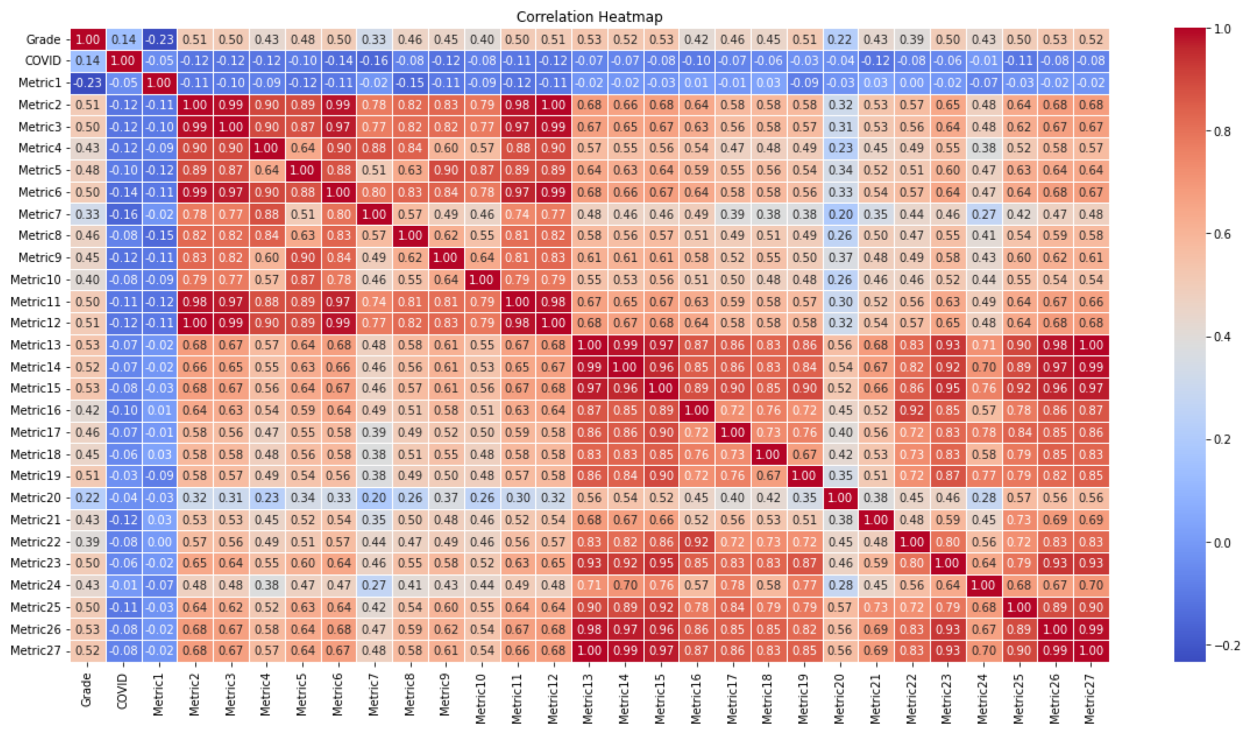


Figure A2. Correlation heatmap of entire dataset

Appendix 3 – Python script machine learning models

```
import pandas as pd

import numpy as np

from sklearn.preprocessing import StandardScaler, PolynomialFeatures

from sklearn.model_selection import train_test_split

from sklearn.decomposition import PCA

from sklearn.linear_model import LinearRegression, Lasso, Ridge

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

import matplotlib.pyplot as plt

import seaborn as sns

# Load the dataset

data = pd.read_excel('PredictionsData.xlsx')
```

```
# Display the first few rows of the data

data.head(), data.info()


# Performing Exploratory data analysis (EDA)


# Inspecting the data and generating descriptive statistics


# Generate descriptive statistics for the dataset

descriptive_stats = data.describe()

print(descriptive_stats)


# Output the descriptive statistics to an Excel file

output_file = 'descriptive_statistics.xlsx'

descriptive_stats.to_excel(output_file)


# Check for any missing values in the dataset

missing_values = data.isnull().sum()


# Display the descriptive statistics and missing values

descriptive_stats, missing_values


# Display the full correlation matrix

full_corr_matrix = data.corr()
```

```
# Display the correlation matrix

print(full_corr_matrix)


# Initial Visualisations


# Set up the matplotlib figure

plt.figure(figsize=(20, 30))


# Loop through each column in the dataset and create a histogram

for i, column in enumerate(data.columns):

    plt.subplot((len(data.columns) + 3) // 4, 4, i + 1)  # Adjust the grid size for 4 columns per row

    sns.histplot(data[column], kde=False, bins=15, color='purple', edgecolor='black', alpha=0.7)

    plt.title(f'Histogram of {column}')

    plt.xlabel(column)

    plt.ylabel('Frequency')


plt.tight_layout(pad=3.0)

plt.show()


# Calculate mean and standard deviation

mean = data['Grade'].mean()

std_dev = data['Grade'].std()


# Plotting the histogram with a distribution line (KDE)

plt.figure(figsize=(10, 6))  # Set the figure size
```

```
sns.histplot(data['Grade'], bins=15, kde=True, color='green', edgecolor='black', alpha=0.7)

# Plot mean line
plt.axvline(mean, color='red', linestyle='--', linewidth=2, label=f'Mean: {mean:.2f}')

# Plot mean - 1 std deviation line
plt.axvline(mean - std_dev, color='blue', linestyle='--', linewidth=2, label=f'-1 Std Dev: {mean - 
std_dev:.2f}')

# Plot mean + 1 std deviation line
plt.axvline(mean + std_dev, color='orange', linestyle='--', linewidth=2, label=f'+1 Std Dev: {mean + 
std_dev:.2f}')

# Add labels and title to the plot
plt.title('Distribution of Grades with KDE, Mean, and Std Deviation')
plt.xlabel('Grade')
plt.ylabel('Frequency')
plt.legend()

# Display the plot
plt.show()

# Plotting the heatmap for correlation analysis
plt.figure(figsize=(20, 10))
sns.heatmap(full_corr_matrix, annot=True,fmt='.2f', cmap='coolwarm', linewidths=0.5)
```

```
# Add title to the heatmap

plt.title('Correlation Heatmap')


# Show the plot

plt.show()


# Plotting pairplot for selected features

selected_features = ['Grade', 'COVID', 'Metric1', 'Metric2', 'Metric3', 'Metric4' , 'Metric5', 'Metric6',
'Metric7']

plt.subplot(4, 1, 3)

sns.pairplot(data[selected_features])

plt.title('Pairplot of Selected Features')


plt.tight_layout()

plt.show()


# Visualize outliers using box plots

plt.figure(figsize=(20, 10))

sns.boxplot(data=data.drop(columns=['Grade']))

plt.xticks(rotation=90)

plt.title('Box Plot of Metrics to Identify Outliers')

plt.show()


# Identify Key Correlations with the target variable

target_variable = 'Grade'

key_correlations = full_corr_matrix[target_variable].sort_values(ascending=False)
```

```
print(key_correlations)


# Multicollinearity check for the original dataset


# Define a threshold to identify highly correlated features
threshold = 0.8


# Find pairs of features with a correlation coefficient above the threshold
features_to_drop = set()
for i in range(len(full_corr_matrix.columns)):
    feature1 = full_corr_matrix.columns[i]
    for j in range(i + 1, len(full_corr_matrix.columns)):
        feature2 = full_corr_matrix.columns[j]
        if abs(full_corr_matrix.loc[feature1, feature2]) > threshold:
            # Drop the feature with the lower correlation to the target variable 'Grade'
            if abs(key_correlations[feature1]) > abs(key_correlations[feature2]):
                features_to_drop.add(feature2)
            else:
                features_to_drop.add(feature1)


# Keep features that are not in the drop list
features_to_keep = [feature for feature in full_corr_matrix.columns if feature not in features_to_drop]


# Display features to drop and those to keep
print("Features to drop due to multicollinearity:")
print(features_to_drop)


# Filter the dataset to only keep the selected features

filtered_data1 = data[features_to_keep]


# Display the features kept

print("Features to keep after removing multicollinearity:")

print(features_to_keep)


# Recalculate the correlation matrix for the filtered dataset

filtered_corr_matrix = filtered_data1.corr()


# Display the correlation matrix

print("Correlation matrix of the filtered dataset:")

print(filtered_corr_matrix)


# Check if any correlations are still above the threshold

high_corr_pairs = []

for i in range(len(filtered_corr_matrix.columns)):

    feature1 = filtered_corr_matrix.columns[i]

    for j in range(i + 1, len(filtered_corr_matrix.columns)):

        feature2 = filtered_corr_matrix.columns[j]

        if abs(filtered_corr_matrix.loc[feature1, feature2]) > threshold:

            high_corr_pairs.append((feature1, feature2, filtered_corr_matrix.loc[feature1, feature2]))
```

```
print("High correlation pairs remaining after filtering:")

print(high_corr_pairs)


# Output the new set of features

print("Final features after removing multicollinearity:")

print(filtered_data1.columns)


# The removal of the metrics is loss of information hence the PCA approach is used later so that
information is not lost and removal of the multi-collinearity


# Standardize the features filtered_data1

scaler = StandardScaler()

X = filtered_data1.drop('Grade', axis=1)

y = filtered_data1['Grade']

X_scaled = scaler.fit_transform(X)


# Split the data into training and testing sets

X_train1, X_test1, y_train1, y_test1 = train_test_split(X_scaled, y, test_size=0.2, random_state=42)


# Train a Linear Regression model on the reduced feature set

lr_model_reduced = LinearRegression()

lr_model_reduced.fit(X_train1, y_train1)


# Make predictions

y_pred_reduced = lr_model_reduced.predict(X_test1)
```

```
# Evaluate the model

mse_reduced = mean_squared_error(y_test1, y_pred_reduced)

r2_reduced = r2_score(y_test1, y_pred_reduced)

mae_reduced = mean_absolute_error(y_test1, y_pred_reduced)


print(f'Mean Squared Error (Reduced Features): {mse_reduced}')

print(f'R^2 Score (Reduced Features): {r2_reduced}')

print (f'Mean Absolute Error (Reduced Features): {mae_reduced}')



# Approach 1 - Original dataset


# Define the columns for predicted Maths, English grades, and the target Engineering grade

predicted_math_col = 'Metric27'

predicted_english_col = 'Metric12'

engineering_grade_col = 'Grade'


# Standardize the features full dataset

scaler = StandardScaler()

X = data.drop(columns=[engineering_grade_col])

y = data[engineering_grade_col]

X_scaled = scaler.fit_transform(X)


# Split data into training and testing sets

def split_data(X, y):
    return train_test_split(X, y, test_size=0.2, random_state=42)


X_train, X_test, y_train, y_test = split_data(X_scaled, y)


# Train and evaluate model function

def train_evaluate_model_with_mae(model, X_train, y_train, X_test, y_test):

    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    mse = mean_squared_error(y_test, y_pred)

    r2 = r2_score(y_test, y_pred)

    mae = mean_absolute_error(y_test, y_pred)

    return mse, r2, mae


# Initialize a list to store results

results_original = []



# Linear Regression on original data

mse_lr, r2_lr, mae_lr = train_evaluate_model_with_mae(LinearRegression(), X_train, y_train, X_test,
y_test)

results_original.append(('Linear Regression', 'original Data', mse_lr, r2_lr, mae_lr))


# Lasso Regression on original data

mse_lasso, r2_lasso, mae_lasso = train_evaluate_model_with_mae(Lasso(), X_train, y_train, X_test,
y_test)

results_original.append(('Lasso Regression', 'original Data', mse_lasso, r2_lasso, mae_lasso))
```

```
# Ridge Regression on original data

mse_ridge, r2_ridge, mae_ridge = train_evaluate_model_with_mae(Ridge(), X_train, y_train, X_test,
y_test)

results_original.append(('Ridge Regression', 'original Data', mse_ridge, r2_ridge, mae_ridge))


# Random Forest on original data

mse_rf, r2_rf, mae_rf = train_evaluate_model_with_mae(RandomForestRegressor(), X_train, y_train,
X_test, y_test)

results_original.append(('Random Forest', 'original Data', mse_rf, r2_rf, mae_rf))


# Implementing PCA fo original data

pca = PCA(n_components=0.95)

X_pca = pca.fit_transform(X_scaled)

X_train_pca, X_test_pca, y_train_pca, y_test_pca = split_data(X_pca, y)


# Linear Regression on PCA-transformed original data

mse_lr_pca, r2_lr_pca, mae_lr_pca = train_evaluate_model_with_mae(LinearRegression(), X_train_pca,
y_train_pca, X_test_pca, y_test_pca)

results_original.append(('Linear Regression', 'original Data PCA', mse_lr_pca, r2_lr_pca, mae_lr_pca))


# Lasso Regression on PCA-transformed original data

mse_lasso_pca, r2_lasso_pca, mae_lasso_pca = train_evaluate_model_with_mae(Lasso(), X_train_pca,
y_train_pca, X_test_pca, y_test_pca)

results_original.append(('Lasso Regression', 'original Data PCA', mse_lasso_pca, r2_lasso_pca,
mae_lasso_pca))
```

```
# Ridge Regression on PCA-transformed original data

mse_ridge_pca, r2_ridge_pca, mae_ridge_pca = train_evaluate_model_with_mae(Ridge(), X_train_pca,
y_train_pca, X_test_pca, y_test_pca)

results_original.append(('Ridge Regression', 'original Data PCA', mse_ridge_pca, r2_ridge_pca,
mae_ridge_pca))


# Random Forest on PCA-transformed original data

mse_rf_pca, r2_rf_pca, mae_rf_pca = train_evaluate_model_with_mae(RandomForestRegressor(),
X_train_pca, y_train_pca, X_test_pca, y_test_pca)

results_original.append(('Random Forest', 'original Data PCA', mse_rf_pca, r2_rf_pca, mae_rf_pca))


# Effect of Feature Engineering on original data


# Define the standardize_features function

def standardize_features(data, target_column):

    scaler = StandardScaler()

    X = data.drop(columns=[target_column])

    y = data[target_column]

    X_scaled = scaler.fit_transform(X)

    return X_scaled, y



# Split data into training and testing sets

def split_data(X, y):

    return train_test_split(X, y, test_size=0.2, random_state=42)


# Function to train, evaluate model, and calculate MSE, R^2, and MAE
```

```
def train_evaluate_model(model, X_train, y_train, X_test, y_test):

    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    mse = mean_squared_error(y_test, y_pred)

    r2 = r2_score(y_test, y_pred)

    mae = mean_absolute_error(y_test, y_pred)

    return mse, r2, mae


# Feature engineering

print("Evaluating models on feature-engineered raw data")


# Polynomial Feature engineering

numeric_features = data.select_dtypes(include=['int64', 'float64']).drop(columns=['Grade',
'COVID']).columns

poly = PolynomialFeatures(degree=2, interaction_only=True, include_bias=False)

X_poly = poly.fit_transform(data[numeric_features])

X_poly_df = pd.DataFrame(X_poly, columns=poly.get_feature_names_out(numeric_features))


# Calculate the skewness for all numeric features in the dataset

skewness = data.skew().sort_values(ascending=False)


# Display the skewness values

print(skewness)


# Set a threshold for skewness to determine which features to log transform

threshold = 0.5
```

```
# Identify the features that have skewness greater than the threshold

skewed_features = skewness[(skewness > threshold) | (skewness < -threshold)].index


# Display the identified skewed features

print(f"Features with skewness greater than {threshold} or less than {-threshold}:")

print(skewed_features)


# Visualize the distribution of the skewed features

plt.figure(figsize=(14, len(skewed_features) * 4))


for i, feature in enumerate(skewed_features):

    plt.subplot(len(skewed_features), 1, i + 1)

    sns.histplot(data[feature], bins=30, kde=True)

    plt.title(f'Distribution of {feature} (Skewness: {skewness[feature]:.2f})')

    plt.xlabel('Value')

    plt.ylabel('Frequency')


plt.tight_layout()

plt.show()


# Apply log transformation to skewed features i have not used log(x) since there are 0s in the COVID
column and to avoid error log(1+x) is used.

data_log_transformed = data.copy()

data_log_transformed[skewed_features] = data_log_transformed[skewed_features].apply(lambda x:
np.log1p(x))
```

```
# Set up the figure and axes for subplots the distribution before and after the log transformation
num_features = len(skewed_features)
fig, axes = plt.subplots(num_features, 2, figsize=(14, 4 * num_features))

# Loop through each skewed feature and create the plots
for i, feature in enumerate(skewed_features):
    # Original feature distribution
    sns.histplot(data[feature], bins=30, kde=True, ax=axes[i, 0])
    axes[i, 0].set_title(f'Original {feature} Distribution')
    axes[i, 0].set_xlabel('Value')
    axes[i, 0].set_ylabel('Frequency')

    # Log-transformed feature distribution
    sns.histplot(data_log_transformed[feature], bins=30, kde=True, ax=axes[i, 1])
    axes[i, 1].set_title(f'Log-Transformed {feature} Distribution')
    axes[i, 1].set_xlabel('Value')
    axes[i, 1].set_ylabel('Frequency')

# Adjust the layout for better readability
plt.tight_layout()
plt.show()

# Combine all engineered features with the original dataset
X_combined = pd.concat([X_poly_df, data_log_transformed[skewed_features]], axis=1)
```

```
X_combined['Grade'] = data['Grade']

X_combined['COVID'] = data['COVID']


X_scaled_combined, y_combined = standardize_features(X_combined, 'Grade')

X_train_comb, X_test_comb, y_train_comb, y_test_comb = split_data(X_scaled_combined, y_combined)


# Linear Regression on combined features

mse_comb, r2_comb, mae_comb = train_evaluate_model(LinearRegression(), X_train_comb,
y_train_comb, X_test_comb, y_test_comb) # Added mae_comb to store the Mean Absolute Error
returned by the function

print(f'Linear Regression (Combined Features) - MSE: {mse_comb}, R^2: {r2_comb}, MAE: {mae_comb}')
# Print the MAE value

results_original.append(('Linear Regression', 'Combined Features', mse_comb, r2_comb, mae_comb)) #
Append MAE to the results


# Lasso Regression on combined features

lasso_regressor = Lasso()

mse_lasso_comb, r2_lasso_comb, mae_lasso_comb = train_evaluate_model(lasso_regressor,
X_train_comb, y_train_comb, X_test_comb, y_test_comb) # Added mae_lasso_comb

print(f'Lasso Regression (Combined Features) - MSE: {mse_lasso_comb}, R^2: {r2_lasso_comb}, MAE:
{mae_lasso_comb}') # Print the MAE value

results_original.append(('Lasso Regression', 'Combined Features', mse_lasso_comb, r2_lasso_comb,
mae_lasso_comb)) # Append MAE to the results


# Ridge Regression on combined features

ridge_regressor = Ridge()

mse_ridge_comb, r2_ridge_comb, mae_ridge_comb = train_evaluate_model(ridge_regressor,
X_train_comb, y_train_comb, X_test_comb, y_test_comb) # Added mae_ridge_comb
```

```
print(f'Ridge Regression (Combined Features) - MSE: {mse_ridge_comb}, R^2: {r2_ridge_comb}, MAE: {mae_ridge_comb}') # Print the MAE value

results_original.append(('Ridge Regression', 'Combined Features', mse_ridge_comb, r2_ridge_comb, mae_ridge_comb)) # Append MAE to the results


# Random Forest on combined features

rf_model_comb = RandomForestRegressor()

mse_rf_comb, r2_rf_comb, mae_rf_comb = train_evaluate_model(rf_model_comb, X_train_comb, y_train_comb, X_test_comb, y_test_comb) # Added mae_rf_comb

print(f'Random Forest (Combined Features) - MSE: {mse_rf_comb}, R^2: {r2_rf_comb}, MAE: {mae_rf_comb}') # Print the MAE value

results_original.append(('Random Forest', 'Combined Features', mse_rf_comb, r2_rf_comb, mae_rf_comb)) # Append MAE to the results


# Convert the results into a DataFrame

results_original_df = pd.DataFrame(results_original, columns=[

    'Model', 'Data Type',

    'Test MSE', 'Test R^2', 'Test MAE'

])


print(results_original_df)


# Output the results to an Excel file

output_file = 'Model_results_Original_data.xlsx'

results_original_df.to_excel(output_file, index=False)
```

```
# Approach 2 - Normalisation of the original dataset for data analysis


# Define the standardize_features function
def standardize_features(data, target_column):
    scaler = StandardScaler()
    X = data.drop(columns=[target_column])
    y = data[target_column]
    X_scaled = scaler.fit_transform(X)
    return X_scaled, y


# Split data into training and testing sets
def split_data(X, y):
    return train_test_split(X, y, test_size=0.2, random_state=42)


# Function to train, evaluate model, and calculate MSE, R^2, and MAE for both training and test data
def train_evaluate_model(model, X_train, y_train, X_test, y_test):
    model.fit(X_train, y_train)


    # Predictions on the training set
    y_train_pred = model.predict(X_train)
    mse_train = mean_squared_error(y_train, y_train_pred)
    r2_train = r2_score(y_train, y_train_pred)
    mae_train = mean_absolute_error(y_train, y_train_pred)


    # Predictions on the testing set
    y_test_pred = model.predict(X_test)
    mse_test = mean_squared_error(y_test, y_test_pred)
    r2_test = r2_score(y_test, y_test_pred)
    mae_test = mean_absolute_error(y_test, y_test_pred)

    return mse_train, r2_train, mae_train, mse_test, r2_test, mae_test

# Copy the original dataset to avoid modifying it directly
data_normalized = data.copy()

# Identify numeric columns (excluding the 'COVID' column)
numeric_columns = data_normalized.select_dtypes(include=[np.number]).columns.tolist()
numeric_columns.remove('COVID')  # Exclude the 'COVID' column itself from normalization

# Iterate over each numeric column and normalize it
for col in numeric_columns:
    # Calculate average for COVID and non-COVID groups
    average_covid = data_normalized.loc[data_normalized['COVID'] == 1, col].mean()
    average_non_covid = data_normalized.loc[data_normalized['COVID'] == 0, col].mean()

    # Calculate the COVID bonus for this column
    covid_bonus = average_covid - average_non_covid

    print(f'Average {col} (COVID): {average_covid}')
    print(f'Average {col} (Non-COVID): {average_non_covid}')
    print(f'COVID Bonus for {col}: {covid_bonus}')


    # Normalize the COVID-affected students' data in this column
    data_normalized.loc[data_normalized['COVID'] == 1, col] -= covid_bonus


    # Recalculate and print new averages after normalization
    new_average_covid = data_normalized[data_normalized['COVID'] == 1][col].mean()
    new_average_non_covid = data_normalized[data_normalized['COVID'] == 0][col].mean()


    print(f'New Average {col} (COVID): {new_average_covid}')
    print(f'New Average {col} (Non-COVID): {new_average_non_covid}')


# Print the first few rows of the normalized dataset to verify
print(data_normalized.head())


# Save the normalized data to an Excel file
output_file = 'normalized_data.xlsx'
data_normalized.to_excel(output_file, index=False)


# Define the columns for predicted Maths, English grades, and the target Engineering grade
predicted_math_col = 'Metric27'
predicted_english_col = 'Metric12'
engineering_grade_col = 'Grade'


# Benchmark Calculation - Simple Average of Maths and English
```

```
data_normalized['Simple_Average'] = data_normalized[[predicted_math_col,
predicted_english_col]].mean(axis=1)


# Calculate MSE, R^2, and MAE for the benchmark

simple_avg_mse = mean_squared_error(data_normalized[engineering_grade_col],
data_normalized['Simple_Average'])

simple_avg_r2 = r2_score(data_normalized[engineering_grade_col],
data_normalized['Simple_Average'])

simple_avg_mae = mean_absolute_error(data_normalized[engineering_grade_col],
data_normalized['Simple_Average'])


# Print the results

print(f'Simple Average Benchmark - MSE: {simple_avg_mse}')

print(f'Simple Average Benchmark - R^2: {simple_avg_r2}')

print(f'Simple Average Benchmark - MAE: {simple_avg_mae}')


# Drop 'Simple_Average' from the data before further processing

data_normalized = data_normalized.drop(columns=['Simple_Average'])


# Standardizing and splitting the normalized data

X_scaled_normalized, y_normalized = standardize_features(data_normalized, 'Grade')

X_train_norm, X_test_norm, y_train_norm, y_test_norm = split_data(X_scaled_normalized,
y_normalized)


# Combine the standardized features with the target variable

X_scaled_normalized_df = pd.DataFrame(X_scaled_normalized,
columns=data_normalized.drop(columns=['Grade']).columns)

combined_df = pd.concat([X_scaled_normalized_df, y_normalized.reset_index(drop=True)], axis=1)
```

```
# Calculate the correlation matrix, including the target variable 'Grade'
corr_matrix_combined = combined_df.corr()

# Display the correlation matrix
print("Correlation Matrix including the Target Variable 'Grade':")
print(corr_matrix_combined)

# Define a threshold for high correlation
threshold = 0.8

# Find pairs of features with a correlation coefficient above the threshold
high_corr_pairs = []
for i in range(len(corr_matrix_combined.columns)):
    for j in range(i + 1, len(corr_matrix_combined.columns)):
        if abs(corr_matrix_combined.iloc[i, j]) > threshold:
            feature1 = corr_matrix_combined.columns[i]
            feature2 = corr_matrix_combined.columns[j]
            high_corr_pairs.append((feature1, feature2, corr_matrix_combined.iloc[i, j]))

# Display the pairs of highly correlated features, including correlations with 'Grade'
print("Highly Correlated Feature Pairs (Correlation > 0.8) Including the Target Variable:")
for pair in high_corr_pairs:
    print(f"{pair[0]} and {pair[1]}: Correlation = {pair[2]:.2f}")
```

```
from statsmodels.stats.outliers_influence import variance_inflation_factor


# Calculate VIF for each feature including the target variable

vif_data_combined = pd.DataFrame()

vif_data_combined["Feature"] = combined_df.columns

vif_data_combined["VIF"] = [variance_inflation_factor(combined_df.values, i) for i in
range(combined_df.shape[1])]


# Display the VIF results

print("Variance Inflation Factor (VIF) including the Target Variable 'Grade':")

print(vif_data_combined)


# Initialised the results

results_2 = []


# Implementing PCA for the normalised dataset


# Perform PCA with the goal of retaining 95% of the variance

pca = PCA(n_components=0.95)

X_pca = pca.fit_transform(X_scaled_normalized)


# Number of components retained

n_components_retained = pca.n_components_

print(f'Number of principal components retained: {n_components_retained}')


# Calculate explained variance
```

```
explained_variance = np.cumsum(pca.explained_variance_ratio_)


# Plotting Scree plot to visualise the component selection
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(explained_variance) + 1), explained_variance, marker='o', linestyle='--')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Scree Plot')
plt.grid(True)
plt.axhline(y=0.95, color='r', linestyle='--', label='95% Explained Variance')
plt.legend()
plt.tight_layout()
plt.show()


# X_scaled_normalized was created from a DataFrame called 'data_normalized'
original_feature_names = data_normalized.drop(columns=['Grade']).columns


# Create a DataFrame to display the loadings of the features in the principal components
loadings_df = pd.DataFrame(pca.components_, columns=original_feature_names)
loadings_df.index = [f'PC{i+1}' for i in range(pca.n_components_)]


# Display the loadings
print("Loadings of the different features for each principal component:")
print(loadings_df)
```

```
# Output the loadings_df to an Excel file

output_path = 'pca_loadings_actual.xlsx'

loadings_df.to_excel(output_path)


# Running models on the normalised data


# Linear Regression on normalized data

mse_lr_train_norm, r2_lr_train_norm, mae_lr_train_norm, mse_lr_test_norm, r2_lr_test_norm,
mae_lr_test_norm = train_evaluate_model(LinearRegression(), X_train_norm, y_train_norm,
X_test_norm, y_test_norm)

results_2.append(('Linear Regression', 'Normalized', mse_lr_train_norm, r2_lr_train_norm,
mae_lr_train_norm, mse_lr_test_norm, r2_lr_test_norm, mae_lr_test_norm))

print(f'Linear Regression (Normalized) - Train MSE: {mse_lr_train_norm}, Test MSE: {mse_lr_test_norm},
Train R^2: {r2_lr_train_norm}, Test R^2: {r2_lr_test_norm}, Train MAE: {mae_lr_train_norm}, Test MAE:
{mae_lr_test_norm}')


# Lasso Regression on normalized data

mse_lasso_train_norm, r2_lasso_train_norm, mae_lasso_train_norm, mse_lasso_test_norm,
r2_lasso_test_norm, mae_lasso_test_norm = train_evaluate_model(Lasso(), X_train_norm,
y_train_norm, X_test_norm, y_test_norm)

results_2.append(('Lasso Regression', 'Normalized', mse_lasso_train_norm, r2_lasso_train_norm,
mae_lasso_train_norm, mse_lasso_test_norm, r2_lasso_test_norm, mae_lasso_test_norm))

print(f'Lasso Regression (Normalized) - Train MSE: {mse_lasso_train_norm}, Test MSE:
{mse_lasso_test_norm}, Train R^2: {r2_lasso_train_norm}, Test R^2: {r2_lasso_test_norm}, Train MAE:
{mae_lasso_train_norm}, Test MAE: {mae_lasso_test_norm}')


# Ridge Regression on normalized data

mse_ridge_train_norm, r2_ridge_train_norm, mae_ridge_train_norm, mse_ridge_test_norm,
r2_ridge_test_norm, mae_ridge_test_norm = train_evaluate_model(Ridge(), X_train_norm,
y_train_norm, X_test_norm, y_test_norm)
```

```
results_2.append(('Ridge Regression', 'Normalized', mse_ridge_train_norm, r2_ridge_train_norm,
mae_ridge_train_norm, mse_ridge_test_norm, r2_ridge_test_norm, mae_ridge_test_norm))

print(f'Ridge Regression (Normalized) - Train MSE: {mse_ridge_train_norm}, Test MSE:
{mse_ridge_test_norm}, Train R^2: {r2_ridge_train_norm}, Test R^2: {r2_ridge_test_norm}, Train MAE:
{mae_ridge_train_norm}, Test MAE: {mae_ridge_test_norm}')


# Random Forest on normalized data

mse_rf_train_norm, r2_rf_train_norm, mae_rf_train_norm, mse_rf_test_norm, r2_rf_test_norm,
mae_rf_test_norm = train_evaluate_model(RandomForestRegressor(), X_train_norm, y_train_norm,
X_test_norm, y_test_norm)

results_2.append(('Random Forest', 'Normalized', mse_rf_train_norm, r2_rf_train_norm,
mae_rf_train_norm, mse_rf_test_norm, r2_rf_test_norm, mae_rf_test_norm))

print(f'Random Forest (Normalized) - Train MSE: {mse_rf_train_norm}, Test MSE: {mse_rf_test_norm},
Train R^2: {r2_rf_train_norm}, Test R^2: {r2_rf_test_norm}, Train MAE: {mae_rf_train_norm}, Test MAE:
{mae_rf_test_norm}')


# Split the PCA-transformed data into training and testing sets

X_train_pca, X_test_pca, y_train_pca, y_test_pca = train_test_split(X_pca, y_normalized, test_size=0.2,
random_state=42)


# Linear Regression on PCA-transformed normalised data

mse_lr_train_pca, r2_lr_train_pca, mae_lr_train_pca, mse_lr_test_pca, r2_lr_test_pca, mae_lr_test_pca
= train_evaluate_model(LinearRegression(), X_train_pca, y_train_pca, X_test_pca, y_test_pca)

results_2.append(('Linear Regression', 'PCA_Normalised', mse_lr_train_pca, r2_lr_train_pca,
mae_lr_train_pca, mse_lr_test_pca, r2_lr_test_pca, mae_lr_test_pca))

print(f'Linear Regression (PCA_Normalised) - Train MSE: {mse_lr_train_pca}, Test MSE:
{mse_lr_test_pca}, Train R^2: {r2_lr_train_pca}, Test R^2: {r2_lr_test_pca}, Train MAE:
{mae_lr_train_pca}, Test MAE: {mae_lr_test_pca}')


# Lasso Regression on PCA-transformed normalised data
```

```
mse_lasso_train_pca, r2_lasso_train_pca, mae_lasso_train_pca, mse_lasso_test_pca, r2_lasso_test_pca,
mae_lasso_test_pca = train_evaluate_model(Lasso(), X_train_pca, y_train_pca, X_test_pca, y_test_pca)

results_2.append(('Lasso Regression', 'PCA_Normalised', mse_lasso_train_pca, r2_lasso_train_pca,
mae_lasso_train_pca, mse_lasso_test_pca, r2_lasso_test_pca, mae_lasso_test_pca))

print(f'Lasso Regression (PCA_Normalised) - Train MSE: {mse_lasso_train_pca}, Test MSE:
{mse_lasso_test_pca}, Train R^2: {r2_lasso_train_pca}, Test R^2: {r2_lasso_test_pca}, Train MAE:
{mae_lasso_train_pca}, Test MAE: {mae_lasso_test_pca}')


# Ridge Regression on PCA-transformed normalised data

mse_ridge_train_pca, r2_ridge_train_pca, mae_ridge_train_pca, mse_ridge_test_pca,
r2_ridge_test_pca, mae_ridge_test_pca = train_evaluate_model(Ridge(), X_train_pca, y_train_pca,
X_test_pca, y_test_pca)

results_2.append(('Ridge Regression', 'PCA_Normalised', mse_ridge_train_pca, r2_ridge_train_pca,
mae_ridge_train_pca, mse_ridge_test_pca, r2_ridge_test_pca, mae_ridge_test_pca))

print(f'Ridge Regression (PCA_Normalised) - Train MSE: {mse_ridge_train_pca}, Test MSE:
{mse_ridge_test_pca}, Train R^2: {r2_ridge_train_pca}, Test R^2: {r2_ridge_test_pca}, Train MAE:
{mae_ridge_train_pca}, Test MAE: {mae_ridge_test_pca}')


# Random Forest on PCA-transformed data

mse_rf_train_pca, r2_rf_train_pca, mae_rf_train_pca, mse_rf_test_pca, r2_rf_test_pca,
mae_rf_test_pca = train_evaluate_model(RandomForestRegressor(), X_train_pca, y_train_pca,
X_test_pca, y_test_pca)

results_2.append(('Random Forest', 'PCA_Normalised', mse_rf_train_pca, r2_rf_train_pca,
mae_rf_train_pca, mse_rf_test_pca, r2_rf_test_pca, mae_rf_test_pca))

print(f'Random Forest (PCA_Normalised) - Train MSE: {mse_rf_train_pca}, Test MSE: {mse_rf_test_pca},
Train R^2: {r2_rf_train_pca}, Test R^2: {r2_rf_test_pca}, Train MAE: {mae_rf_train_pca}, Test MAE:
{mae_rf_test_pca}')


# Feature engineering on the normalised data


# Converting the normalized dataset back to a DataFrame for easier handling
```

```
X_scaled_normalized_df = pd.DataFrame(X_scaled_normalized,
columns=data_normalized.drop(columns=['Grade']).columns)


# Polynomial Feature Engineering


# Identify numeric features from the normalized data (excluding the target 'Grade')

numeric_features = X_scaled_normalized_df.columns


# Generate polynomial features (degree=2)

poly = PolynomialFeatures(degree=2, interaction_only=True, include_bias=False)

X_poly = poly.fit_transform(X_scaled_normalized_df[numeric_features])

X_poly_df = pd.DataFrame(X_poly, columns=poly.get_feature_names_out(numeric_features))


# Skewness Calculation


# Calculate the skewness for all numeric features in the normalized dataset

skewness = X_scaled_normalized_df.skew().sort_values(ascending=False)


# Display the skewness values

print("Skewness of Features in Normalized Data:")

print(skewness)


# Set a threshold for skewness to determine which features to log transform

threshold = 0.5


# Identify the features that have skewness greater than the threshold
```

```
skewed_features = skewness[(skewness > threshold) | (skewness < -threshold)].index


# Display the identified skewed features

print(f"Features with skewness greater than {threshold} or less than {-threshold}:")

print(skewed_features)


# Visualization of Skewed Features


# Visualize the distribution of the skewed features before log transformation

plt.figure(figsize=(14, len(skewed_features) * 4))


for i, feature in enumerate(skewed_features):

    plt.subplot(len(skewed_features), 1, i + 1)

    sns.histplot(X_scaled_normalized_df[feature], bins=30, kde=True)

    plt.title(f'Distribution of {feature} (Skewness: {skewness[feature]:.2f})')

    plt.xlabel('Value')

    plt.ylabel('Frequency')


plt.tight_layout()

plt.show()


# Log Transformation of Skewed Features


# Apply log transformation to skewed features, adding a small constant to avoid log(0) or log(negative)

X_log_transformed_df = X_scaled_normalized_df.copy()
```

```
X_log_transformed_df[skewed_features] = X_log_transformed_df[skewed_features].apply(lambda x: np.log1p(x - x.min() + 1))



# Set up the figure and axes for subplots: distribution before and after the log transformation
num_features = len(skewed_features)
fig, axes = plt.subplots(num_features, 2, figsize=(14, 4 * num_features))

# Loop through each skewed feature and create the plots
for i, feature in enumerate(skewed_features):
    # Original feature distribution
    sns.histplot(X_scaled_normalized_df[feature], bins=30, kde=True, ax=axes[i, 0])
    axes[i, 0].set_title(f'Original {feature} Distribution')
    axes[i, 0].set_xlabel('Value')
    axes[i, 0].set_ylabel('Frequency')

    # Log-transformed feature distribution
    sns.histplot(X_log_transformed_df[feature], bins=30, kde=True, ax=axes[i, 1])
    axes[i, 1].set_title(f'Log-Transformed {feature} Distribution')
    axes[i, 1].set_xlabel('Value')
    axes[i, 1].set_ylabel('Frequency')

# Adjust the layout for better readability
plt.tight_layout()
plt.show()
```

```
# Combine all engineered features with the original dataframe

X_combined_df = pd.concat([X_scaled_normalized_df, X_poly_df,
X_log_transformed_df[skewed_features]], axis=1)


# Check for NaN values in the combined dataframe

nan_columns = X_combined_df.columns[X_combined_df.isna().any()].tolist()


print(f"Columns with NaN values: {nan_columns}")

print(X_combined_df[nan_columns].isna().sum())


# Split data into training and testing sets

X_train_comb, X_test_comb, y_train_comb, y_test_comb = train_test_split(X_combined_df,
y_normalized, test_size=0.2, random_state=42)


# Linear Regression on combined features

mse_lr_train_comb, r2_lr_train_comb, mae_lr_train_comb, mse_lr_test_comb, r2_lr_test_comb,
mae_lr_test_comb = train_evaluate_model(LinearRegression(), X_train_comb, y_train_comb,
X_test_comb, y_test_comb)

results_2.append(('Linear Regression', 'Combined Features', mse_lr_train_comb, r2_lr_train_comb,
mae_lr_train_comb, mse_lr_test_comb, r2_lr_test_comb, mae_lr_test_comb))

print(f'Linear Regression (Combined Features) - Train MSE: {mse_lr_train_comb}, Test MSE:
{mse_lr_test_comb}, Train R^2: {r2_lr_train_comb}, Test R^2: {r2_lr_test_comb}, Train MAE:
{mae_lr_train_comb}, Test MAE: {mae_lr_test_comb}')


# Lasso Regression on combined features

mse_lasso_train_comb, r2_lasso_train_comb, mae_lasso_train_comb, mse_lasso_test_comb,
r2_lasso_test_comb, mae_lasso_test_comb = train_evaluate_model(Lasso(), X_train_comb,
y_train_comb, X_test_comb, y_test_comb)
```

```
results_2.append(('Lasso Regression', 'Combined Features', mse_lasso_train_comb,
r2_lasso_train_comb, mae_lasso_train_comb, mse_lasso_test_comb, r2_lasso_test_comb,
mae_lasso_test_comb))

print(f'Lasso Regression (Combined Features) - Train MSE: {mse_lasso_train_comb}, Test MSE:
{mse_lasso_test_comb}, Train R^2: {r2_lasso_train_comb}, Test R^2: {r2_lasso_test_comb}, Train MAE:
{mae_lasso_train_comb}, Test MAE: {mae_lasso_test_comb}')


# Ridge Regression on combined features

mse_ridge_train_comb, r2_ridge_train_comb, mae_ridge_train_comb, mse_ridge_test_comb,
r2_ridge_test_comb, mae_ridge_test_comb = train_evaluate_model(Ridge(), X_train_comb,
y_train_comb, X_test_comb, y_test_comb)

results_2.append(('Ridge Regression', 'Combined Features', mse_ridge_train_comb,
r2_ridge_train_comb, mae_ridge_train_comb, mse_ridge_test_comb, r2_ridge_test_comb,
mae_ridge_test_comb))

print(f'Ridge Regression (Combined Features) - Train MSE: {mse_ridge_train_comb}, Test MSE:
{mse_ridge_test_comb}, Train R^2: {r2_ridge_train_comb}, Test R^2: {r2_ridge_test_comb}, Train MAE:
{mae_ridge_train_comb}, Test MAE: {mae_ridge_test_comb}')


# Random Forest on combined features

rf_model_comb = RandomForestRegressor()

mse_rf_train_comb, r2_rf_train_comb, mae_rf_train_comb, mse_rf_test_comb, r2_rf_test_comb,
mae_rf_test_comb = train_evaluate_model(rf_model_comb, X_train_comb, y_train_comb,
X_test_comb, y_test_comb)

results_2.append(('Random Forest', 'Combined Features', mse_rf_train_comb, r2_rf_train_comb,
mae_rf_train_comb, mse_rf_test_comb, r2_rf_test_comb, mae_rf_test_comb))

print(f'Random Forest (Combined Features) - Train MSE: {mse_rf_train_comb}, Test MSE:
{mse_rf_test_comb}, Train R^2: {r2_rf_train_comb}, Test R^2: {r2_rf_test_comb}, Train MAE:
{mae_rf_train_comb}, Test MAE: {mae_rf_test_comb}')


# Create a DataFrame from the results

results_df_1 = pd.DataFrame(results_2, columns=['Model', 'Data Type', 'Train MSE', 'Train R^2', 'Train
MAE', 'Test MSE', 'Test R^2', 'Test MAE'])
```

```
# Specify the file name
file_name = 'Machine_learning_model_results_normalised_final.xlsx'

# Write the DataFrame to an Excel file
results_df_1.to_excel(file_name, index=False)

print(f'Results have been saved to {file_name}')

# Visualisation of the performance of the models on the normalised dataset

# Set the style for the plots
sns.set(style="whitegrid")

# Function to create line plots for comparison
def plot_line_comparison(df, metric, metric_name):
    plt.figure(figsize=(14, 8))

    # Melt the DataFrame to make it easier to plot
    df_melted = df.melt(id_vars=['Model', 'Data Type'],
                        value_vars=[f'Train {metric}', f'Test {metric}'],
                        var_name='Set', value_name=metric_name)

    # Create a line plot for each model with different data types
    sns.lineplot(x='Data Type', y=metric_name, hue='Model', style='Set',
                markers=True, dashes=False, data=df_melted, ci=None)


    plt.title(f'{metric_name} Comparison Across Models and Data Types')

    plt.ylabel(metric_name)

    plt.xlabel('Data Type')

    plt.xticks(rotation=45)

    plt.legend(title='Model and Set', loc='best')

    plt.tight_layout()

    plt.show()


# Plot MSE for all Data Types

plot_line_comparison(results_df_1, 'MSE', 'MSE')


# Plot R^2 for all Data Types

plot_line_comparison(results_df_1, 'R^2', 'R^2')


# Plot MAE for all Data Types

plot_line_comparison(results_df_1, 'MAE', 'MAE')


# Hyperparameter tuning


# Import library for tuning the models

from sklearn.model_selection import GridSearchCV


# Define function for tuning models
```

```
def tune_model(model, param_grid, X_train, y_train):
    grid_search = GridSearchCV(estimator=model, param_grid=param_grid,
scoring='neg_mean_squared_error', cv=5, n_jobs=-1)
    grid_search.fit(X_train, y_train)
    return grid_search.best_params_, grid_search.best_score_


# Define function for evaluating models (including both training and testing errors)
def evaluate_best_model(model, X_train, y_train, X_test, y_test):
    model.fit(X_train, y_train)


    # Predictions on training set
    y_train_pred = model.predict(X_train)
    train_mse = mean_squared_error(y_train, y_train_pred)
    train_r2 = r2_score(y_train, y_train_pred)
    train_mae = mean_absolute_error(y_train, y_train_pred)


    # Predictions on testing set
    y_test_pred = model.predict(X_test)
    test_mse = mean_squared_error(y_test, y_test_pred)
    test_r2 = r2_score(y_test, y_test_pred)
    test_mae = mean_absolute_error(y_test, y_test_pred)


    return train_mse, train_r2, train_mae, test_mse, test_r2, test_mae


# Initialize a list to store all results
all_results = []
```

```
# Define models and their parameter grids
model_param_grids = {
  'Linear Regression': (LinearRegression(), {
    'fit_intercept': [True],
    'copy_X': [True],
    'positive': [False]
  }),
  'Lasso': (Lasso(), {
    'alpha': [0.01, 0.1, 1.0, 10.0],
    'max_iter': [5000]
  }),
  'Ridge': (Ridge(), {
    'alpha': [0.1, 1.0, 10.0, 100.0],
    'max_iter': [5000]
  }),
  'Random Forest': (RandomForestRegressor(), {
    'n_estimators': [100, 300],
    'max_depth': [10, 20],
    'min_samples_split': [5,10],
    'min_samples_leaf': [2,4],
    'bootstrap': [True]
  })
}
```

```
# List of datasets to include in the tuning and evaluation

datasets = {

    'Normalized': (X_train_norm, X_test_norm, y_train_norm, y_test_norm),

    'PCA Transformed': (X_train_pca, X_test_pca, y_train_pca, y_test_pca),

    'Feature Engineered': (X_train_comb, X_test_comb, y_train_comb, y_test_comb)

}


# Function to handle hyperparameter tuning and evaluation for each dataset

def tune_and_evaluate_for_dataset(data_name, X_train_set, X_test_set, y_train_set, y_test_set):

    print(f"\nProcessing {data_name}...")


    # Process each model one by one

    for model_name, (model, param_grid) in model_param_grids.items():

        print(f"  Tuning {model_name} on {data_name}...")


        # Hyperparameter tuning

        best_params, _ = tune_model(model, param_grid, X_train_set, y_train_set)


        # Re-initialize the model with the best parameters

        model.set_params(**best_params)


        # Evaluate the tuned model and calculate both training and testing errors

        train_mse, train_r2, train_mae, test_mse, test_r2, test_mae = evaluate_best_model(

            model, X_train_set, y_train_set, X_test_set, y_test_set

        )
        # Store the results for this model
        all_results.append((
            data_name, model_name, best_params,
            train_mse, train_r2, train_mae,
            test_mse, test_r2, test_mae
        ))

# Loop over each dataset and process them
for data_name, (X_train_set, X_test_set, y_train_set, y_test_set) in datasets.items():
    tune_and_evaluate_for_dataset(data_name, X_train_set, X_test_set, y_train_set, y_test_set)

# Convert the results into a DataFrame
results_df_2 = pd.DataFrame(all_results, columns=[
    'Data Type', 'Model', 'Best Parameters',
    'Train MSE', 'Train R^2', 'Train MAE',
    'Test MSE', 'Test R^2', 'Test MAE'
])

# Save the results to an Excel file
output_file = 'hyperparameter_tuning_results_final.xlsx'
results_df_2.to_excel(output_file, index=False)

# Print confirmation
print(f'\nHyperparameter tuning and evaluation results have been saved to {output_file}')
```

```
# Visualisation of the performance of machine learning model

# Calculate benchmark values for all metrics (assuming these are pre-calculated)
simple_avg_mse = 2.2896530602940173
simple_avg_r2 = 0.6486128487763867
simple_avg_mae = 1.0107717033098549

# Construct the benchmark data to have the correct lengths
benchmark_data = {
    'Model': ['Benchmark'] * 3,
    'Data Type': ['Normalized', 'PCA Transformed', 'Feature Engineered'],
    'Train MSE': [simple_avg_mse] * 3,
    'Train R^2': [simple_avg_r2] * 3,
    'Train MAE': [simple_avg_mae] * 3,
    'Test MSE': [simple_avg_mse] * 3,
    'Test R^2': [simple_avg_r2] * 3,
    'Test MAE': [simple_avg_mae] * 3
}

# Convert the dictionary to a DataFrame
benchmark_df = pd.DataFrame(benchmark_data)

# Append the benchmark DataFrame to the results DataFrame
results_df_2 = pd.concat([results_df_2, benchmark_df], ignore_index=True)
```

```
# Function to create point plots for train and test metrics for all models with benchmark
def plot_train_test_performance_pointplot(df, benchmark_values):
    plt.figure(figsize=(18, 8))

    # Melt the DataFrame to make it easier to plot for Train metrics
    df_melted_train = df.melt(id_vars=['Model', 'Data Type'],
                              value_vars=['Train MSE', 'Train R^2', 'Train MAE'],
                              var_name='Metric', value_name='Value')

    # Create the point plot for Train metrics
    plt.subplot(1, 2, 1)
    sns.pointplot(x='Model', y='Value', hue='Metric', data=df_melted_train, markers=["o", "s", "D"],
linestyles=["-", "--", "-."])

    # Plot benchmark points for Train
    plt.axhline(y=benchmark_values['MSE'], color='red', linestyle='--', label='Benchmark MSE')
    plt.axhline(y=benchmark_values['R^2'], color='green', linestyle='--', label='Benchmark R^2')
    plt.axhline(y=benchmark_values['MAE'], color='blue', linestyle='--', label='Benchmark MAE')

    plt.title('Train Performance Across Models (MSE, R^2, MAE)')
    plt.ylabel('Metric Value')
    plt.xlabel('Model')
    plt.xticks(rotation=45)
    plt.legend(title='Metric', loc='best')
    # Melt the DataFrame to make it easier to plot for Test metrics
    df_melted_test = df.melt(id_vars=['Model', 'Data Type'],
                             value_vars=['Test MSE', 'Test R^2', 'Test MAE'],
                             var_name='Metric', value_name='Value')

    # Create the point plot for Test metrics
    plt.subplot(1, 2, 2)
    sns.pointplot(x='Model', y='Value', hue='Metric', data=df_melted_test, markers=["o", "s", "D"],
linestyles=["-", "--", "-."])

    # Plot benchmark points for Test
    plt.axhline(y=benchmark_values['MSE'], color='red', linestyle='--', label='Benchmark MSE')
    plt.axhline(y=benchmark_values['R^2'], color='green', linestyle='--', label='Benchmark R^2')
    plt.axhline(y=benchmark_values['MAE'], color='blue', linestyle='--', label='Benchmark MAE')

    plt.title('Test Performance Across Models (MSE, R^2, MAE)')
    plt.ylabel('Metric Value')
    plt.xlabel('Model')
    plt.xticks(rotation=45)
    plt.legend(title='Metric', loc='best')

    plt.tight_layout()
    plt.show()

# Generate the point plots for Train and Test metrics for all models with benchmark
plot_train_test_performance_pointplot(results_df_2, {
    'MSE': simple_avg_mse,
    'R^2': simple_avg_r2,
    'MAE': simple_avg_mae
})

# Data for the bar chart
metrics = ['MSE', 'R^2', 'MAE']
values = [simple_avg_mse, simple_avg_r2, simple_avg_mae]

# Create the bar chart
plt.figure(figsize=(10, 6))
plt.bar(metrics, values, color=['red', 'green', 'blue'])

# Add titles and labels
plt.title('Benchmark Model Performance')
plt.ylabel('Metric Value')

# Display the value on top of each bar
for i, value in enumerate(values):
    plt.text(i, value + 0.05, f'{value:.3f}', ha='center', fontsize=12)

plt.tight_layout()
plt.show()
```

```
# # Extra graphs


# # Data for the benchmark and epochs

# benchmark_data = {

#     'Metric': ['MSE', 'R^2', 'MAE'],

#     'Benchmark': [3.1677707006369427, 0.3023928811935258, 1.4105095541401274]

# }


# epochs_data = {

#     'Epocs': [50, 100],

#     'MSE': [2.0, 1.84],

#     'R^2': [0.7, 0.72],

#     'MAE': [0.98, 0.93]

# }


# # Convert to DataFrame

# benchmark_df = pd.DataFrame(benchmark_data)

# epochs_df = pd.DataFrame(epochs_data)


# # Plot the bar chart

# plt.figure(figsize=(12, 6))


# # Plot benchmark values

# plt.bar(benchmark_df['Metric'], benchmark_df['Benchmark'], color='grey', alpha=0.7,
label='Benchmark')
```

```
# # Plot values for each epoch

# width = 0.2  # width of the bars

# plt.bar([x - width for x in range(len(benchmark_df['Metric']))], epochs_df.loc[0, ['MSE', 'R^2', 'MAE']],

#        width=width, color='blue', alpha=0.7, label='Epoch 50')

# plt.bar([x for x in range(len(benchmark_df['Metric']))], epochs_df.loc[1, ['MSE', 'R^2', 'MAE']],

#        width=width, color='orange', alpha=0.7, label='Epoch 100')


# # Adding labels and titles

# plt.title('Model Performance Comparison with Benchmark')

# plt.ylabel('Metric Value')

# plt.xticks(range(len(benchmark_df['Metric'])), benchmark_df['Metric'])

# plt.legend()


# # Show values on top of bars

# for i, value in enumerate(benchmark_df['Benchmark']):

#     plt.text(i, value + 0.05, f'{value:.2f}', ha='center', fontsize=10)


# for i, value in enumerate(epochs_df.loc[0, ['MSE', 'R^2', 'MAE']]):

#     plt.text(i - width, value + 0.05, f'{value:.2f}', ha='center', fontsize=10, color='blue')


# for i, value in enumerate(epochs_df.loc[1, ['MSE', 'R^2', 'MAE']]):

#     plt.text(i, value + 0.05, f'{value:.2f}', ha='center', fontsize=10, color='orange')


# plt.tight_layout()

# plt.show()
# # New data including both training and test metrics for each epoch
# epochs_data_extended = {
#     'Epocs': ['Epoch 50 Train', 'Epoch 50 Test', 'Epoch 100 Train', 'Epoch 100 Test'],
#     'MSE': [1.8, 1.9, 1.85, 1.93],
#     'R^2': [0.68, 0.66, 0.69, 0.65],
#     'MAE': [0.95, 1.02, 0.96, 1.93]
# }

# # Benchmark values (as previously defined)
# benchmark_data = {
#     'Metric': ['MSE', 'R^2', 'MAE'],
#     'Benchmark': [3.1677707006369427, 0.3023928811935258, 1.4105095541401274]
# }

# # Convert to DataFrame
# benchmark_df = pd.DataFrame(benchmark_data)
# epochs_df_extended = pd.DataFrame(epochs_data_extended)

# # Plot the bar chart with larger labels and pastel colors
# plt.figure(figsize=(14, 8))

# # Plot benchmark values with increased width and pastel color
# plt.bar(benchmark_df['Metric'], benchmark_df['Benchmark'], color='grey', alpha=0.7, width=0.25,
label='Benchmark')
```

```
# # Plot values for each training and test epoch with increased width and pastel colors

# width = 0.25  # Adjust width for more bars

# positions = [-1.5*width, -0.5*width, 0.5*width, 1.5*width]

# colors = ['#FFB6C1', '#87CEFA', '#FFDAB9', '#98FB98']  # Pastel colors


# for i, epoch in enumerate(epochs_df_extended['Epocs']):

#     plt.bar([x + positions[i % 4] for x in range(len(benchmark_df['Metric']))],

#             epochs_df_extended.iloc[i, 1:],

#             width=width, alpha=0.7, label=epoch, color=colors[i])


# # Adding labels and titles with increased font size

# plt.title('Model Performance Comparison: Training, Test, and Benchmark', fontsize=20)

# plt.ylabel('Metric Value', fontsize=18)

# plt.xticks(range(len(benchmark_df['Metric'])), benchmark_df['Metric'], fontsize=18)

# plt.legend(fontsize=14)


# # Show values on top of bars with larger font size

# for i, value in enumerate(benchmark_df['Benchmark']):

#     plt.text(i, value + 0.05, f'{value:.2f}', ha='center', fontsize=16)


# for i in range(len(epochs_df_extended)):

#     for j in range(len(benchmark_df['Metric'])):

#         plt.text(j + positions[i % 4], epochs_df_extended.iloc[i, j+1] + 0.05,

#                  f'{epochs_df_extended.iloc[i, j+1]:.2f}',

#                  ha='center', fontsize=16, color='black')
# plt.tight_layout()

# plt.show()


# # Data for the model after hyperparameter tuning

# tuned_train_data = {

#     'Epocs': ['Epoch 50 Train', 'Epoch 100 Train'],

#     'MSE': [1.78, 1.82],

#     'R^2': [0.74, 0.73],

#     'MAE': [0.92, 0.94]

# }


# tuned_test_data = {

#     'Epocs': ['Epoch 50 Test', 'Epoch 100 Test'],

#     'MSE': [1.56, 1.66],

#     'R^2': [0.72, 0.70],

#     'MAE': [0.89, 0.90]

# }


# # Data for the model without hyperparameter tuning

# untuned_data = {

#     'Epocs': ['Epoch 50 Untuned', 'Epoch 100 Untuned'],

#     'MSE': [1.56, 1.66],

#     'R^2': [0.72, 0.70],

#     'MAE': [0.89, 0.90]
# }


# # Convert to DataFrames
# benchmark_df = pd.DataFrame(benchmark_data)
# tuned_train_df = pd.DataFrame(tuned_train_data)
# tuned_test_df = pd.DataFrame(tuned_test_data)
# untuned_df = pd.DataFrame(untuned_data)


# # Setting up data for the grouped bar chart
# metrics = ['MSE', 'R^2', 'MAE']
# values = np.array([tuned_train_df.iloc[:, 1:].values,
#                   tuned_test_df.iloc[:, 1:].values,
#                   untuned_df.iloc[:, 1:].values])


# # Plotting the grouped bar chart
# fig, ax = plt.subplots(figsize=(14, 8))


# # Set positions for the groups of bars
# bar_width = 0.2
# bar_positions = np.arange(len(metrics))


# # Plotting each condition
# colors = ['#FFB6C1', '#87CEFA', '#FFDAB9', '#98FB98']  # Pastel colors
# for i, key in enumerate(tuned_train_df['Epocs']):
#     ax.bar(bar_positions + i * bar_width, tuned_train_df.iloc[i, 1:], width=bar_width, label=key,
color=colors[i])
```

```
# for i, key in enumerate(tuned_test_df['Epocs']):

#     ax.bar(bar_positions + (i + 2) * bar_width, tuned_test_df.iloc[i, 1:], width=bar_width, label=key,
color=colors[i + 2])


# for i, key in enumerate(untuned_df['Epocs']):

#     ax.bar(bar_positions + (i + 4) * bar_width, untuned_df.iloc[i, 1:], width=bar_width, label=key,
color='purple')


# # Plot benchmark values

# ax.bar(bar_positions + 6 * bar_width, benchmark_df['Benchmark'], width=bar_width, color='darkgrey',
alpha=0.7, label='Benchmark')


# # Adding labels and title with updated font size and weight

# ax.set_xlabel('Metrics', fontsize=16, color='black')

# ax.set_ylabel('Values', fontsize=16, color='black')

# ax.set_title('Model Performance Comparison: Before and After Hyperparameter Tuning', fontsize=18,
color='black')

# ax.set_xticks(bar_positions + (6 / 2) * bar_width)

# ax.set_xticklabels(metrics, fontsize=14, color='black')

# ax.legend(fontsize=12)


# # Show values on top of bars with larger font size and bold

# for i in range(len(tuned_train_df)):

#     for j in range(len(metrics)):

#         ax.text(bar_positions[j] + i * bar_width, tuned_train_df.iloc[i, j+1] + 0.03,

#                 f'{tuned_train_df.iloc[i, j+1]:.2f}', ha='center', fontsize=14, fontweight='bold', color='black')
# for i in range(len(tuned_test_df)):

#     for j in range(len(metrics)):

#         ax.text(bar_positions[j] + (i + 2) * bar_width, tuned_test_df.iloc[i, j+1] + 0.03,

#                 f'{tuned_test_df.iloc[i, j+1]:.2f}', ha='center', fontsize=14, fontweight='bold', color='black')


# for i in range(len(untuned_df)):

#     for j in range(len(metrics)):

#         ax.text(bar_positions[j] + (i + 4) * bar_width, untuned_df.iloc[i, j+1] + 0.03,

#                 f'{untuned_df.iloc[i, j+1]:.2f}', ha='center', fontsize=14, fontweight='bold', color='black')


# # Benchmark values

# for j in range(len(metrics)):

#     ax.text(bar_positions[j] + 6 * bar_width, benchmark_df['Benchmark'][j] + 0.03,

#             f'{benchmark_df["Benchmark"][j]:.2f}', ha='center', fontsize=14, fontweight='bold',
color='black')


# plt.tight_layout()

# plt.show()


# # Data from the table provided

# data_table = {
```

```
#   'Model': [
#       'Linear Regression', 'Lasso Regression', 'Ridge Regression', 'Random Forest',
#       'Linear Regression', 'Lasso Regression', 'Ridge Regression', 'Random Forest',
#       'Linear Regression', 'Lasso Regression', 'Ridge Regression', 'Random Forest'
#   ],
#   'Data Type': [
#       'Normalized', 'Normalized', 'Normalized', 'Normalized',
#       'PCA_Normalised', 'PCA_Normalised', 'PCA_Normalised', 'PCA_Normalised',
#       'Combined Features', 'Combined Features', 'Combined Features', 'Combined Features'
#   ],
#   'Train MSE': [
#       2.356475151, 3.83886804, 2.373523792, 0.414065694,
#       2.416807003, 2.716267444, 2.416812262, 0.433362912,
#       1.2548E-27, 3.83886804, 0.590478534, 0.429905711
#   ],
#   'Train R^2': [
#       0.474450711, 0.143842289, 0.470648463, 0.907653628,
#       0.460995291, 0.394208581, 0.460994118, 0.903349895,
#       1, 0.143842289, 0.868309422, 0.904120932
#   ],
#   'Train MAE': [
#       1.226343296, 1.609198301, 1.218164454, 0.517263119,
#       1.230191104, 1.335347507, 1.230359453, 0.528973061,
#       2.63291E-14, 1.609198301, 0.584380633, 0.531731187
#   ]
# }


# # Convert to DataFrame

# df = pd.DataFrame(data_table)


# # Colors to use

# colors = ['#FFB6C1', '#87CEFA', '#FFDAB9', '#98FB98', '#DDA0DD', '#FFA07A', '#20B2AA', '#778899',
'#B0E0E6', '#40E0D0', '#FF6347', '#FAFAD2']


# # Create Lollipop chart for Train Data

# metrics_train = ['Train MSE', 'Train R^2', 'Train MAE']

# x_train = np.arange(len(metrics_train))

# bar_width = 0.15


# fig, ax = plt.subplots(figsize=(14, 8))


# # Plot for Train Data with specified line thickness and color

# for i, (model, dtype) in enumerate(zip(df['Model'], df['Data Type'])):

#     markerline, stemline, baseline = ax.stem(x_train + i * bar_width, df.iloc[i][metrics_train], linefmt='-',
markerfmt='o', basefmt=" ")

#     plt.setp(stemline, color='yellow', linewidth=2)  # Thicker yellow stem

#     plt.setp(markerline, color=colors[i], markersize=10)  # Marker color as per the list


# ax.set_xlabel('Metrics', fontsize=12)

# ax.set_ylabel('Values', fontsize=12)

# ax.set_title('Train Performance Comparison Across Models and Data Types', fontsize=14)
# ax.set_xticks(x_train + len(df) * bar_width / len(metrics_train))
# ax.set_xticklabels(metrics_train, fontsize=12)
# ax.legend(fontsize=10, loc='upper left', bbox_to_anchor=(1,1))

# # Data from the table provided
# data_90 = {
#     'Model': [
#         'Linear Regression', 'Lasso Regression', 'Ridge Regression', 'Random Forest',
#         'Linear Regression', 'Lasso Regression', 'Ridge Regression', 'Random Forest',
#         'Linear Regression', 'Lasso Regression', 'Ridge Regression', 'Random Forest'
#     ],
#     'Data Type': [
#         'Normalized', 'Normalized', 'Normalized', 'Normalized',
#         'PCA_Normalised', 'PCA_Normalised', 'PCA_Normalised', 'PCA_Normalised',
#         'Combined Features', 'Combined Features', 'Combined Features', 'Combined Features'
#     ],
#     'Test MSE': [
#         3.738037252, 4.126315979, 3.841726181, 4.098917785,
#         3.8469101, 3.882314116, 3.847498717, 3.871634997,
#         46.38182246, 4.126315979, 8.227882187, 4.237413672
#     ],
#     'Test R^2': [
#         0.107495757, 0.014789241, 0.082738698, 0.02133091,
#         0.081500971, 0.073047809, 0.081360431, 0.075597585,
#         -10.07425383, 0.014789241, -0.964512194, -0.011736756
#     ],
#     'Test MAE': [
#         1.523204982, 1.596152513, 1.560579355, 1.627064055,
#         1.567569485, 1.578796722, 1.567622152, 1.588197028,
#         5.356497497, 1.596152513, 2.171554035, 1.647990514
#     ]
# }


# # Convert to DataFrame
# df = pd.DataFrame(data_90)


# # Colors to use
# colors = ['#FFB6C1', '#87CEFA', '#FFDAB9', '#98FB98', '#DDA0DD', '#FFA07A', '#20B2AA', '#778899',
'#B0E0E6', '#40E0D0', '#FF6347', '#FAFAD2']


# # Create Lollipop chart for Test Data
# metrics_test = ['Test MSE', 'Test R^2', 'Test MAE']
# x_test = np.arange(len(metrics_test))
# bar_width = 0.15


# fig, ax = plt.subplots(figsize=(14, 8))


# # Plot for Test Data with specified line thickness and color
# for i, (model, dtype) in enumerate(zip(df['Model'], df['Data Type'])):
#     markerline, stemline, baseline = ax.stem(x_test + i * bar_width, df.iloc[i][metrics_test], linefmt='-',
markerfmt='o', basefmt=" ")
```

```
#     plt.setp(stemline, color='yellow', linewidth=2)  # Thicker yellow stem

#     plt.setp(markerline, color=colors[i], markersize=10)  # Marker color as per the list


# ax.set_xlabel('Metrics', fontsize=12)

# ax.set_ylabel('Values', fontsize=12)

# ax.set_title('Test Performance Comparison Across Models and Data Types', fontsize=14)

# ax.set_xticks(x_test + len(df) * bar_width / len(metrics_test))

# ax.set_xticklabels(metrics_test, fontsize=12)

# ax.legend(fontsize=10, loc='upper left', bbox_to_anchor=(1,1))


# plt.tight_layout()

# plt.show()


# -*- coding: utf-8 -*-

"""

Created on Sat Aug 24 18:55:38 2024


@author: 5558022_neural_network


To be used in a seperate script


"""


import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout

from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

from sklearn.model_selection import GridSearchCV

import matplotlib.pyplot as plt

import numpy as np


# Load the Excel file

data = pd.read_excel('PredictionsData.xlsx')

data.copy = data


# Normalisation of the COVID variable

# Assuming 'COVID' is the column or a flag that indicates COVID-related rows

covid_data = data[data['COVID'] == 1]  # Filter rows related to COVID

non_covid_data = data[data['COVID'] == 0]  # Filter rows not related to COVID


# Normalize the COVID-related data (MinMax scaling in this example)

scaler_covid = MinMaxScaler()

covid_data_scaled = covid_data.copy()

covid_data_scaled.iloc[:, :-1] = scaler_covid.fit_transform(covid_data.iloc[:, :-1])  # Apply scaling to all
features except the target


# Merge the data back together

data_normalized = pd.concat([covid_data_scaled, non_covid_data])
```

```
# Standardize the features for the whole dataset (including the normalized COVID data)
scaler = StandardScaler()
X = data_normalized.drop('Grade', axis=1)
y = data_normalized['Grade']
X_scaled = scaler.fit_transform(X)

# Eastablishing the benchmark

# Define the columns for predicted Maths, English grades, and the target Engineering grade
predicted_math_col = 'Metric27'
predicted_english_col = 'Metric12'
engineering_grade_col = 'Grade'

# Benchmark Calculation - Simple Average of Maths and English
data['Simple_Average'] = data[[predicted_math_col, predicted_english_col]].mean(axis=1)
simple_avg_mse = mean_squared_error(data[engineering_grade_col], data['Simple_Average'])
simple_avg_r2 = r2_score(data[engineering_grade_col], data['Simple_Average'])
simple_avg_mae = mean_absolute_error(data[engineering_grade_col], data['Simple_Average'])

print(f'Benchmark - MSE: {simple_avg_mse}, R^2: {simple_avg_r2}, MAE: {simple_avg_mae}')

# Drop 'Simple_Average' from the data before further processing
data_normalized = data_normalized.drop(columns=['Simple_Average'])

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

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


# Define the neural network architecture

model = Sequential()

model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(32, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(1))


# Compile the model

model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mean_squared_error'])


# Display the model's architecture

model.summary()


# Train the neural network model on 50 epocs

history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.2, verbose=1)


# # Train the neural network model on 100 epocs

# history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.2, verbose=1)


# Evaluate the model on the test data

loss, mse = model.evaluate(X_test, y_test, verbose=1)
```

```
# Make predictions
y_pred = model.predict(X_test)

# Calculate R^2 score and MAE
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'R^2 Score: {r2}')
print(f'Mean Absolute Error: {mae}')

# Calculate training error metrics
y_pred_train = model.predict(X_train)
mse_train = mean_squared_error(y_train, y_pred_train)
r2_train = r2_score(y_train, y_pred_train)
mae_train = mean_absolute_error(y_train, y_pred_train)

print(f'Training Mean Squared Error: {mse_train}')
print(f'Training R^2 Score: {r2_train}')
print(f'Training Mean Absolute Error: {mae_train}')

# Further Neural Network with Grid Search

# Define the neural network architecture as a function
def create_model(optimizer='adam', dropout_rate=0.5):
    model = Sequential()

    model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))

    model.add(Dropout(dropout_rate))

    model.add(Dense(32, activation='relu'))

    model.add(Dropout(dropout_rate))

    model.add(Dense(1))

    model.compile(optimizer=optimizer, loss='mean_squared_error')

    return model


# Custom Keras Regressor class to use in GridSearchCV

from sklearn.base import BaseEstimator, RegressorMixin


class KerasRegressor(BaseEstimator, RegressorMixin):

    def __init__(self, build_fn, optimizer='adam', dropout_rate=0.5, epochs=50, batch_size=32, 
verbose=0):

        self.build_fn = build_fn

        self.optimizer = optimizer

        self.dropout_rate = dropout_rate

        self.epochs = epochs

        self.batch_size = batch_size

        self.verbose = verbose

        self.model_ = None


    def fit(self, X, y):

        self.model_ = self.build_fn(optimizer=self.optimizer, dropout_rate=self.dropout_rate)

        self.model_.fit(X, y, epochs=self.epochs, batch_size=self.batch_size, verbose=self.verbose)
        return self

    def predict(self, X):
        return self.model_.predict(X)

# Create the Keras regressor
model = KerasRegressor(build_fn=create_model, verbose=0)

# Define the grid search parameters
param_grid = {
    'optimizer': ['adam', 'rmsprop'],
    'batch_size': [32, 64],
    'epochs': [50, 100],
    'dropout_rate': [0.3, 0.5, 0.7]
}

# Perform grid search
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=1, cv=5)
grid_result = grid.fit(X_train, y_train)

# Print the best parameters and results
print(f"Best: {grid_result.best_score_} using {grid_result.best_params_}")

# Get the best model
best_model = grid_result.best_estimator_
```

```
# Evaluate the model on the test data

y_pred = best_model.predict(X_test)

r2 = r2_score(y_test, y_pred)

mse = mean_squared_error(y_test, y_pred)

mae = mean_absolute_error(y_test, y_pred)


print(f'Mean Squared Error: {mse}')

print(f'R^2 Score: {r2}')

print(f'Mean Absolute Error: {mae}')


# Evaluate the model on the training data

y_pred_train = best_model.predict(X_train)

mse_train = mean_squared_error(y_train, y_pred_train)

r2_train = r2_score(y_train, y_pred_train)

mae_train = mean_absolute_error(y_train, y_pred_train)


print(f'Training Mean Squared Error (After Hyperparameter Tuning): {mse_train}')

print(f'Training R^2 Score (After Hyperparameter Tuning): {r2_train}')

print(f'Training Mean Absolute Error (After Hyperparameter Tuning): {mae_train}')



# Plot training & validation loss values - Performance graph

plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val_loss'], label='Validation Loss')


plt.title('Neural Network Performance - Loss Over 100 Epochs')

plt.xlabel('Epoch')

plt.ylabel('Loss (Mean Squared Error)')

plt.legend(loc='upper right')

plt.grid(True)


plt.show()


# Neural network comparison with the benchmark


benchmark_mse = simple_avg_mse

benchmark_r2 = simple_avg_r2

benchmark_mae = simple_avg_mae


initial_mse = mse

initial_r2 = r2

initial_mae = mae


tuned_test_mse = mse  # After tuning, on the test set

tuned_test_r2 = r2    # After tuning, on the test set

tuned_test_mae = mae  # After tuning, on the test set
```

```
tuned_train_mse = mse_train  # After tuning, on the training set

tuned_train_r2 = r2_train    # After tuning, on the training set

tuned_train_mae = mae_train  # After tuning, on the training set


# Metrics to plot

metrics = ['MSE', 'R^2', 'MAE']


# Combine results

benchmark_values = [benchmark_mse, benchmark_r2, benchmark_mae]

initial_values = [initial_mse, initial_r2, initial_mae]

tuned_test_values = [tuned_test_mse, tuned_test_r2, tuned_test_mae]

tuned_train_values = [tuned_train_mse, tuned_train_r2, tuned_train_mae]


# Set up the bar chart

x = np.arange(len(metrics))  # the label locations

width = 0.2  # the width of the bars


fig, ax = plt.subplots(figsize=(12, 7))


rects1 = ax.bar(x - 1.5*width, benchmark_values, width, label='Benchmark')

rects2 = ax.bar(x - 0.5*width, initial_values, width, label='Initial Predictions')

rects3 = ax.bar(x + 0.5*width, tuned_test_values, width, label='Tuned Predictions (Test Set)')

rects4 = ax.bar(x + 1.5*width, tuned_train_values, width, label='Tuned Predictions (Training Set)')


# Add some text for labels, title, and custom x-axis tick labels, etc.
```

```
ax.set_xlabel('Metrics')

ax.set_title('Comparison of Benchmark, Initial, and Tuned Predictions with Training Error')

ax.set_xticks(x)

ax.set_xticklabels(metrics)

ax.legend()


# Attach a text label above each bar in *rects*, displaying its height.

def autolabel(rects):

    """Attach a text label above each bar in *rects*, displaying its height."""

    for rect in rects:

        height = rect.get_height()

        ax.annotate('{}'.format(round(height, 2)),

                    xy=(rect.get_x() + rect.get_width() / 2, height),

                    xytext=(0, 3),  # 3 points vertical offset

                    textcoords="offset points",

                    ha='center', va='bottom')


autolabel(rects1)

autolabel(rects2)

autolabel(rects3)

autolabel(rects4)


fig.tight_layout()


plt.show()
```

```
# # Save the final tuned model - this is to save the model to run for new excel with predicted scores.

# best_model.model_.save('trained_model.h5')
```