1. Introduction

Predicting student enrolment in sixth form is crucial for educational planning and resource allocation. This dissertation explores the development of predictive models to estimate the number of students enrolled each year. By analysing factors such as past enrolment data, academic performance, socio-economic indicators, and demographic trends, the research provides insights for optimising staffing, budgeting, and resource management. These predictions help schools better meet student needs and maintain high educational standards.

WMG Academy, like many educational institutions, faces the challenge of attracting and enrolling the most suitable students for its Sixth Form program. Sixth Form students often apply to multiple schools, creating a competitive environment for the institution (Gorard and Siddiqui, 2018). Understanding which students are likely to enrol can help target efforts to attract the best candidates. As students typically apply to multiple schools, the decision-making process can be complex, influenced by a variety of factors such as academic performance, extracurricular involvement, socio-economic background, and personal preferences (Hemsley-Brown and Oplatka, 2015b). Understanding these factors and how they influence enrolment decisions is crucial for WMG Academy to target and support prospective students effectively (Perna, 2006).

1.1 Background

In recent years, the landscape of education has become increasingly competitive, with students having more choices regarding where and what they study (Hemsley-Brown and Oplatka, 2015a). As a result, understanding the factors that influence student enrolment decisions has gained significant attention in educational research and practice (Perna, 2006).

Student enrolment prediction models are designed to anticipate the number of students enrolled in each academic period. These models are typically built using historical data on student applications, acceptances, actual enrolments, demographic information, academic performance, and other personal attributes (Thomas and Galambos, 2004). The primary goal is to identify the key factors that influence enrolment decisions and use these insights to build models that can predict future enrolment behaviour fairly.

1.2 Purpose of the Study

The primary aim of this study is to develop a predictive model that classifies Sixth Form applicants into three categories: very likely to enrol, very unlikely to enrol, and undecided. By identifying the characteristics and behaviours that correlate with these categories, WMG Academy can tailor its recruitment efforts to address undecided students' specific needs and concerns, thereby increasing the overall enrolment rate and ensuring a better match between students and the academy.

1.3 Research problem

The research problem addressed in this dissertation is the need to develop a robust and accurate predictive model for student enrolment at WMG Academy. This model must account for various factors, including the distance between a student's home and the academy, the subjects they choose, and additional demographic and academic variables. The complexity of these factors and their interactions poses a significant challenge in building an effective predictive model (Chen, 2013).

1.4 Objectives

The primary objective of this dissertation is to develop and evaluate predictive models for student enrolment at WMG Academy. The specific objectives are as follows:

1. To explore and preprocess the available data, ensuring it is suitable for modelling.
2. To develop and compare multiple machine learning models, including Logistic Regression, Decision Trees, Random Forest, Gradient Boosting, Support Vector Machines (SVM), and XGBoost.
3. To evaluate the models' performance using appropriate metrics, including accuracy, precision, recall, F1 score, and ROC-AUC.
4. To conduct hyperparameter tuning and cross-validation to enhance model performance and reduce overfitting.
5. To validate the models using a hold-out set and assess their generalisation capabilities after evaluating them on the unseen data.
6. Classification of the predictive probabilities in categories, “enrol”, “not enrolled” and “undecided”.

1.5 Scope

The scope of this research is limited to the data provided by WMG Academy, which includes information on student applications, subject choices, and demographic variables. The analysis will focus on predicting enrolment for a specific academic year, using historical data to train and validate the models. While the study will explore various machine learning techniques, it will not delve into more complex ensemble methods beyond those mentioned, due to the limited data available. The research will not consider external factors such as economic conditions, policy changes, or broader educational trends, as these are beyond the scope of the data provided.

1.6 Significance of the Study

This research is significant for several reasons. First, it contributes to the growing body of literature on predictive modelling in education by applying advanced machine-learning techniques to the problem of student enrolment. Second, it provides practical insights for WMG Academy, enabling the institution to make better decisions that improve resource allocation and strategic planning. By identifying the key factors that influence enrolment, the academy can tailor its recruitment strategies and enhance its appeal to prospective students (Maringe, 2006).

Moreover, the methodologies and findings from this study can be adapted and applied to other educational institutions facing similar challenges. As data-driven decision-making becomes increasingly important in education, the ability to accurately predict student enrolment will be an asset for institutions worldwide.

2. Literature Review

2.1 Introduction

This literature review provides an overview of current research on predictive modelling in education, with a focus on student enrolment. As institutions increasingly adopt data-driven decision-making, understanding the factors' influencing enrolment has become

essential. Accurate predictive models can help optimise resource allocation, refine recruitment strategies, and improve student outcomes.

2.2 Theoretical framework

The chapter explores theoretical frameworks guiding research, reviews key studies applying predictive techniques to enrolment data, and highlights factors influencing enrolment decisions. While not all studies focus on the sixth form, they are relevant to the discussion. Additionally, it examines challenges like data quality, model complexity, and generalisability.

One of the most influential frameworks in this field is Hossler and Gallagher (1987) model of student college choice. This model is particularly relevant as it offers a comprehensive understanding of the factors that influence a student's decision-making process when selecting an educational institution. The model outlines a three-stage process: predisposition, search, and choice. In the predisposition stage, students form their aspirations about higher education based on personal, social, and economic factors. During the search stage, they gather information about potential institutions, evaluating factors such as location, cost, academic offerings, and institutional reputation. Finally, in the choice stage, students decide where to enrol, balancing their preferences with practical considerations such as financial aid and acceptance offers (Hossler and Gallagher, 1987).

The relevance of Hossler and Gallagher (1987) model in predictive modelling lies in its ability to integrate multiple factors into a cohesive framework, allowing researchers to understand how these factors interact to influence enrolment decisions. By considering variables such as academic performance, socio-economic background, and personal preferences, this model aligns closely with the goals of predictive modelling in education, which seeks to forecast student enrolment by analysing these and other relevant factors (Hossler and Gallagher, 1987).

Another critical theory is Ajzen (1991) Theory of Planned Behaviour posits that individual behaviour is driven by behavioural intentions, shaped by attitudes, subjective norms, and perceived behavioural control. This theory is highly applicable to understanding students' decision-making process when choosing an educational institution. According to Ajzen (1991) a student's intention to enrol in a particular institution is influenced by their

attitudes towards the institution (e.g., perceived quality of education), the influence of important others (e.g., parents, peers), and their perceived ability to succeed in that institution (Ajzen, 1991). This framework is particularly useful in predictive modelling as it highlights the importance of psychological and social factors, which can be quantified and incorporated into predictive models to enhance their accuracy.

2.3 Predictive modelling in education

Predictive modelling in education has become an essential tool for understanding student enrolment patterns, particularly in contexts like sixth-form education. The sixth form represents a critical phase in a student's academic trajectory, where they specialise in subjects aligned with their career aspirations. Predictive models allow educational institutions to make data-driven decisions, ensuring efficient resource allocation, curriculum planning, and student support (Kalim, 2023).

Techniques in predictive modelling

Predictive models leverage statistical techniques and machine learning algorithms to forecast future enrolment based on historical and real-time data. Common techniques include regression analysis, decision trees, and advanced machine-learning approaches like random forests and neural networks (Yi and Duval-Couetil, 2021). These models analyse various factors, such as students' academic performance, socio-economic background, and personal preferences, which are crucial in predicting enrolment. For instance, students with strong academic performance and supportive family environments are often more likely to enrol in sixth form, while socio-economic barriers may hinder others (Baker and Britton, 2024) .

Application in sixth form education

In sixth form education, predictive modelling is applied to identify students likely to continue their studies, as well as those at risk of dropping out or who cannot decide what to do. This helps schools focus on targeted interventions to increase enrolment and retention rates. Models can predict which subjects are in higher demand, helping schools adjust their curriculum and allocate appropriate resources (Collom, 2023). By anticipating enrolment trends, schools can also ensure that adequate teaching staff and facilities are available, preventing resource shortages.

For example, predictive models can incorporate variables such as prior academic performance and attitudes towards education, which are vital indicators of whether a student will continue into sixth form (Mansor et al., 2021). These models also consider external factors, such as financial incentives or societal influences like peer pressure or family expectations, which heavily impact students' decisions.

Challenges and considerations

While predictive modelling offers numerous advantages, there are challenges to its effective implementation. One challenge is the availability and accuracy of data, as predictive models rely on comprehensive datasets to make accurate forecasts. Furthermore, models may unintentionally reinforce inequalities if they are not carefully designed to account for socio-economic disparities (Baker and Britton, 2024).

Additionally, financial aid programs, such as the Tennessee Reconnect Grant, have shown that economic support can significantly influence enrolment decisions (Collom, 2023). This suggests that incorporating financial and policy variables into predictive models could enhance their effectiveness in predicting sixth-form enrolment, particularly among disadvantaged students.

2.4 Factors influencing student enrollment

Student enrolment in sixth form is shaped by a combination of academic, socio-economic, and institutional factors, each of which plays a critical role in a student's decision to continue their education. Understanding these factors is key for educational institutions to develop strategies that encourage enrolment and improve retention.

Academic performance

One of the most significant factors influencing sixth form enrolment is academic performance during secondary education. Students who perform well in their earlier years are more likely to continue into sixth form, as they typically seek to further their academic achievements (Taylor et al., 2022). High-performing students often see sixth form as a necessary step toward university or other higher education opportunities.

Socio-Economic background

Socio-economic factors are equally important in influencing student decisions to enrol in sixth form. Students from more affluent families are better equipped to continue their education due to fewer financial barriers (Woodward, 2022). On the other hand, students from lower-income families may face challenges, such as needing to enter the workforce earlier or being unable to afford the costs associated with further education. Financial support programs can play a crucial role in mitigating these challenges, thereby increasing enrolment among disadvantaged groups (Mansor et al., 2021).

Demographic factors

Demographic factors, including age, gender, and ethnicity, can also influence sixth-form enrolment. For instance, studies have shown that certain demographic groups may face unique challenges when it comes to accessing further education. In some regions, male students may be more likely to enter the workforce directly after secondary school, while female students may be encouraged to continue their education, depending on cultural or societal expectations (Kalim, 2023).

Geographical proximity

The geographical location of a sixth-form institution can significantly impact student enrolment decisions. Students are more likely to attend institutions that are closer to their homes, as longer commutes can lead to higher travel costs and increased time commitments, which might discourage enrolment (Woodward, 2022). For students from lower-income backgrounds, the added expense of transportation can be a major barrier, making local institutions a more attractive option. Schools located far from students' homes often struggle to attract enrolments unless they offer substantial support, such as transportation assistance or accommodation options

2.5 Challenges in predictive modelling for enrollment

While predictive modelling offers significant potential, it also presents challenges. One of the primary concerns is data limitations. Kotsiantis et al. (2004) discuss how issues with data quality and availability can significantly hinder the development and accuracy of predictive models in educational settings. In their study on predicting students' performance in distance learning environments, they highlight the impact of incomplete or inconsistent data on model outcomes, underscoring the need for high-quality datasets to ensure reliable predictions (Kotsiantis et al., 2004).

"Another challenge is overfitting, where models perform well on training data but fail to generalise to new data. Lawrence and Giles (2000) emphasize the importance of balancing model complexity with generalisation to avoid overfitting. In their work, they discuss how excessively complex models can capture noise in the training data, leading to poor performance on unseen data. They recommend strategies such as cross-validation and regularisation to mitigate overfitting and ensure that models generalise well (Lawrence and Giles, 2000).

"Interpretability is another critical issue in predictive modelling. While more complex models, such as deep learning, often provide high accuracy, they typically lack transparency, making it challenging for decision-makers to comprehend the underlying reasoning behind predictions. Rudin (2019) advocates for the development and use of interpretable models, especially in critical decision-making contexts. Rudin (2019) argues that while accuracy is essential, it should not be prioritised at the expense of interpretability, as the ability to understand and trust a model's predictions is crucial for effective decision-making in many applications (Rudin, 2019).

In summary, predictive modelling for enrolment is fraught with challenges stemming from the complexity of educational resource allocation, the diversity of student types, and the limitations of different predictive algorithms. While advanced machine learning models show promise, the variability of data and the dynamic nature of educational environments continue to complicate accurate forecasting. The literature suggests that despite the advancements in predictive analytics, there remains a need for further research and development to overcome these challenges.

2.6 Gaps in the literature

Despite extensive research on predictive modelling and enrolment, significant gaps remain in predicting Sixth Form enrolment. Limited studies have explored advanced methods like Gradient Boosting and XGBoost, with most focusing on higher education. Challenges faced by specialised institutions, requiring models tailored to specific student interests, have been overlooked. This dissertation addresses these gaps by applying advanced machine learning techniques and analysing key factors influencing Sixth Form enrolment. Additionally, it highlights the need for context-specific models, as generic

ones may miss nuances. The study aims to improve the generalisability and accuracy of predictive models, enhancing enrolment management and strategic planning.

2.7 Summary

This literature review has provided a comprehensive overview of the key theories, methodologies, and findings related to predictive modelling in education. It has highlighted the factors influencing student enrolment and discussed the challenges associated with building accurate and interpretable predictive models. The identified gaps in the literature underscore the need for further research in this area, particularly the application of advanced machine learning techniques in the context of Sixth Form enrolment at specialised institutions.

3. Methodology

The methodology section of this dissertation outlines the systematic approach undertaken to investigate and predict Sixth Form student enrolment at WMG Academy. The primary goal is to develop predictive models that can forecast student enrolment, thereby providing actionable insights for strategic decision-making and resource allocation. To achieve this, a comprehensive data analysis was performed, encompassing data preprocessing, and the application of advanced machine learning techniques.

3.1 Research design

The study begins with the collection and inspection of the dataset, focusing on data quality and key feature distributions. Data preprocessing addresses missing values, normalisation, and encoding to ensure the data is model-ready. The research applies multiple machine learning algorithms, including Logistic Regression, Decision Trees, Random Forests, SVM, and XGBoost, chosen for their ability to handle complex data patterns. Ensemble methods like Random Forest and XGBoost are used to enhance robustness and predictive power. Model performance is evaluated using metrics such as accuracy, precision, recall, F1 score, and AUC-ROC, with cross-validation ensuring generalisability. GridSearchCV is employed for hyperparameter tuning, while statistical techniques like variance inflation factor (VIF) and correlation analysis validate model

reliability. This methodical, iterative design ensures the models are both accurate and interpretable, offering valuable insights for enrolment management at WMG Academy

3.2 Data collection

Source of data

The data used for this dissertation was provided in two anonymised datasets (2023 and 2024) by WMG Academy. The predictive model was trained using 2023 data and tested on 2024 data, allowing for an evaluation of the model's ability to predict future outcomes, such as student enrolment for the next year. This method is particularly interesting because it reflects a real-world scenario where future predictions are critical. However, one challenge is that the datasets from 2023 and 2024 may differ significantly due to changes in external factors, such as economic conditions, policy shifts, or evolving student preferences. These differences can impact the model's accuracy and its ability to generalise across different periods. These datasets have included key information about student demographics, specific subjects that they have selected, geographic distance from the academy, and their enrolment status.

The datasets were assumed to be representative of the typical enrolment patterns at WMG Academy, making the findings generalisable within this context. The inclusion of both training and test datasets ensures that the predictive models can be evaluated for accuracy and generalisability across different student cohorts. This dataset provided a robust foundation for developing predictive models to predict students' enrolment at WMG Academy.

Description of dataset

The training dataset contains 264 rows and 19 columns while the test data contains 541 rows and 19 columns. The variables include:

- **Distance**: The original data contained zip codes and a tool was developed to turn zip codes into distances to preserve anonymity. Then the geographic distance of each student from the WMG academy was originally represented as a string (e.g., "5 km").

- ***A* and *B***: Categorical features related to the student's profile (such as home-schooled or not or gender).

- ***C1* to *C14***: A series of binary variables representing specific choices or selections made by the students during the enrolment process. These variables represent student choices about the subjects they wish to study, with students typically expected to make up to three subject choices. Each choice is indicated by a checkmark (✓).

- ***Additional Subjects***: This categorical feature indicates whether students expressed interest in or pursued additional subjects or qualifications, represented by checkmarks (✓). If a student did not provide this information, the data is recorded as missing. This variable was treated appropriately and discussed in detail in the data preprocessing step.

- ***Enroled?***: The target variable, a variable indicating whether the student enrolled in the Sixth Form. The enrolment of the student is represented by (✓) and the blanks represents either the student is not enrolled or has not decided whether to enrol or not.

3.3 Data pre-processing

Data preprocessing is an essential step in ensuring the dataset is cleaned and transformed appropriately for analysis. In this study, both the training and test datasets underwent identical preprocessing steps to ensure consistency across model training and evaluation. This process involved handling missing values, transforming categorical variables, encoding binary features, and normalising numerical columns. A consistent preprocessing approach across datasets is crucial to avoid discrepancies that could lead to biased model performance.

3.3.1. Transformation of variables

1. Processing *'Distance'* column

The *Distance* column initially contained non-numeric values in the form of strings, appended with the suffix " km," which prevented its direct use in calculations. To address this:

- **Suffix removal and conversion to numeric**: The " km" suffix was removed from all values, and the column was converted to a numeric data type, allowing for proper numerical operations.
- **Imputation of missing values**: Any missing values in the *Distance* column were filled with the mean value of the column. This approach ensured that missing entries did not introduce bias while maintaining the overall distribution of the data.

This preprocessing step ensured that both the training and test datasets had a consistent numeric representation of the *Distance* feature, which is critical for further modelling.

2. Processing *'Feature A'*

Feature A presented a mix of missing and non-missing values, requiring transformation to maintain data integrity. The original feature was categorical, and during the anonymisation process, the categorical variable was assigned specific numerical values. In the anonymised data, it wasn't clear whether the missing values were genuinely missing or if the student chose not to provide the information. Hence, the feature was processed as a binary feature, as described below:

- **Binary feature creation**: Two new binary features were derived from Feature A: *A_Numeric* and *A_Missing*. The *A_Numeric* feature indicated the presence of data (1 if a value was present, 0 otherwise), while *A_Missing* captured the absence of data (1 for missing, 0 otherwise).
- **Dropping the original column**: After the binary features were created, the original A column was removed to avoid the multicollinearity between new features and original feature.

This transformation allowed the model to interpret the presence or absence of data in a meaningful way, contributing to potential patterns related to missing information.

3. Handling *'Feature B'*

Feature B contained missing values and the specific entry "Prefer not to say," which needed careful handling:

- **Imputation of missing values**: Missing values were imputed using the mode (the most frequent value), ensuring that the distribution of categorical information remained intact.
- **Replacing 'Prefer not to say'**: Instances of "Prefer not to say" were replaced by the mode to maintain consistency across the dataset, avoiding ambiguity and ensuring that the column was fully prepared for analysis.

4. Encoding categorical features (*C1-C14*)

The columns *C1* through *C14* represented binary choices made by students regarding the subjects they wanted to study, with up to three subject choices typically expected. These columns were encoded as follows:

- **Binary encoding**: Checkmarks (✓) indicating subject selection were transformed into binary values (1 for selected, 0 otherwise). This encoding was applied consistently across both the training and test datasets, ensuring the categorical nature of these columns was preserved while making them compatible with machine learning models.

5. Processing the *'Additional Subjects'* column

The Additional Subjects column indicated whether students opted for additional qualifications, but it included missing values and categorical entries such as "Yes" and "No." The column was transformed into binary features:

- **Creation of binary features**:

- *Subject_Yes*: This feature indicated whether students had opted for additional subjects, represented by any non-null value other than "No".
- *Subject_No*: This feature indicated whether students explicitly marked "No" for additional subjects. However, there were no markings of "No" in the test data hence the column was not created in the test data. This did not impact the analysis because of multicollinearity which is discussed in detail further.
- *Subject_Missing*: This feature flagged missing entries in the Additional Subjects column.

After creating these binary features, the original *Additional Subjects* column was dropped.

6. Converting the *'Enroled?'* column into binary format

The target variable, *Enroled?*, was originally represented by checkmarks (✓) indicating whether a student had enrolled. For use in classification tasks, this column needed to be converted into a binary format:

- **Binary conversion**: The checkmark (✓) was converted into 1 for enrolled students, and all other values were set to 0, indicating non-enrolment.

3.3.2. Exploratory data analysis (EDA)

Descriptive analysis

The analysis focuses on key features such as *Distance*, *Feature B*, and the target variable *Enroled?,* as well as engineered features *like A_Numeric* and *A_Missing*. The goal is to highlight similarities and differences between the two datasets, which may influence model performance and evaluation. For complete summary statistics, refer to Table. A1.

Comparison of training and test data summary statistics

Mean Values:

- **Distance**: The average distance in the training set is 10.90 km, which is higher than the test set's mean of 8.79 km. This suggests a slightly different distribution of distances between the two datasets, with students in the test set tending to live closer to the academy.

- **B**: The mean of column B is 0.79 in the training set, while the test set has a lower mean of 0.65. This indicates that the proportion of instances classified as '1' in this column is higher in the training data, which may affect model performance if not accounted for.

- **Enroled?**: In the training set, approximately 54% of students are enrolled (Enroled? mean = 0.54), whereas the test set has a significantly lower mean of 0.23, indicating that only around 23.5% of students in the test set are enrolled. This difference in enrolment rates may impact the model's generalisability.

Comparison of *Distance* distribution between training and test data

The distribution of the *Distance* feature shows noticeable differences between the training and test datasets, as illustrated in Fig.3.1. Most of the *Distance* values in the training set are concentrated between 5 and 15 km, with a peak around 10 km, suggesting that most students in the training data live relatively close to the academy. In contrast, the test dataset shows a stronger concentration of distances below 10 km, with most students living within this range. This indicates that students in the test set generally live even closer to the academy compared to those in the training set.

Both datasets show a drop-off in the number of students living farther away, but this decline is more pronounced in the test data, where most students live within a narrower range of distances. While the training set contains several students who live as far as 40 km from the academy, the test set contains fewer long-distance students but includes a significant outlier with a distance of 220 km. This outlier in the test set is much higher than the maximum value in the training set and represents a notable difference in the overall distribution of distances hence this outlier was dropped in the further analysis.

In terms of variability, the test dataset has a higher standard deviation for Distance (17.03 km) compared to the training set (5.23 km), which reflects greater geographical

dispersion among students in the test set. However, despite this wider range, the concentration of students living close to the academy is much stronger in the test data.

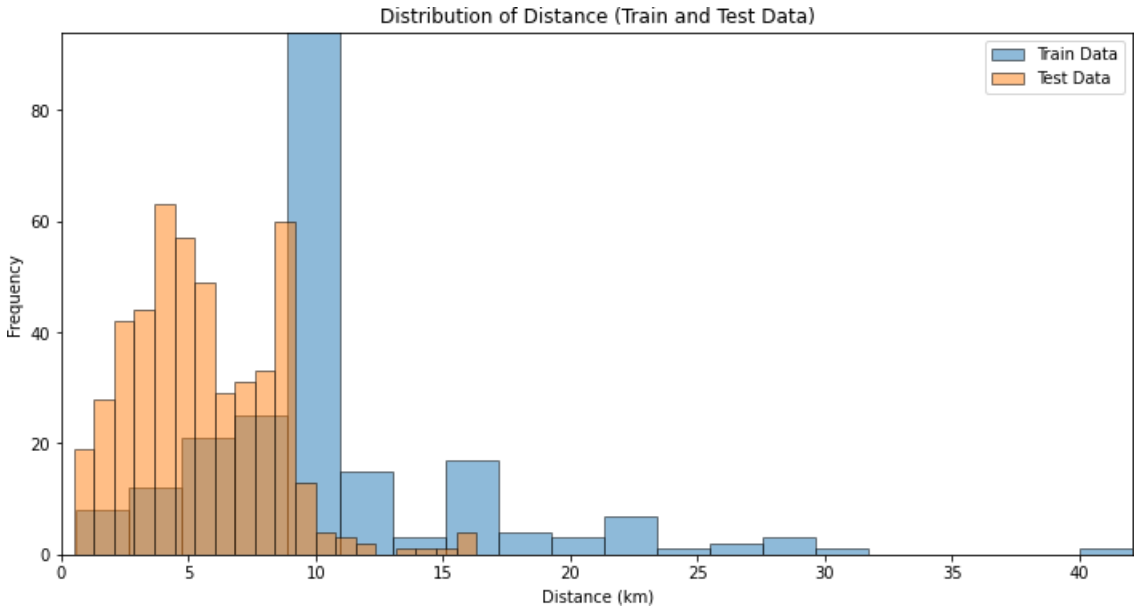

Fig.3.1 Distribution of the *Distance* for train and test data

Correlation analysis

Understanding the relationships between variables in the dataset is critical for building reliable machine learning models. A correlation analysis was performed to evaluate the degree of linear relationships between the independent variables and to detect any multicollinearity that might affect model performance. The full matrix heatmap can be found in the Fig. A1, Fig.A2.

1. Correlation between Independent variables and dependent variable (Enroled?)

Train data: In the 2023 training dataset, subject choices strongly predict enrolment. Subject_No shows a negative correlation (-0.67), meaning students not taking additional subjects were less likely to enrol, while Subject_Yes has a positive correlation (0.67), indicating higher enrolment likelihood. Other factors, like A_Missing (-0.15) and Distance (-0.10), show weaker negative correlations, suggesting that missing data and greater distance slightly reduce the likelihood of enrolment.

Test data: In the 2024 test dataset, Subject_No (-0.67) and Subject_Yes (0.67) maintain their strong correlations with enrolment, reaffirming the importance of subject choices. A_Missing shows a slightly stronger negative correlation (-0.20), indicating a lower likelihood of enrolment for students with missing data. C5 (0.53) and C10 (0.65) also become more influential predictors. Additionally, the negative correlation between Distance and enrolment strengthens to -0.17, suggesting students living farther away are less likely to enrol.

2. Consistency between *C5* and *C7*

The relationship between C5 and C7 remained strong in both training and test datasets, with a high positive correlation of 0.663 in the training data, slightly decreasing to 0.534 in the test data. This consistent correlation suggests a robust link between the two variables, making them key features for modelling. Their stable co-variation across datasets indicates they contribute significantly to the model's predictive power in both training and unseen data.

3. Changes in the relationship between *C6* and *C10*

The relationship between C6 and C10 varied between datasets. In the training data, they showed a moderate positive correlation of 0.452, which increased to 0.648 in the test data. This stronger correlation in the test set suggests a more pronounced interaction between these features in unseen data. The variability highlights the need for models to account for these stronger interactions, as they may play a more critical role in predicting outcomes than initially indicated.

4. Moderate correlation between *C9* and *C5*

The relationship between C9 and C5 remained stable but showed slight variations across datasets. In the training data, they had a moderate positive correlation of 0.423, which decreased to 0.357 in the test data. While the relationship held, the reduced strength suggests it may vary depending on the test data's distribution. This minor variability underscores the need to account for such interactions in model design to prevent performance drops when transitioning from training to test data.

3.3.3. Multicollinearity between variables

The Variance Inflation Factor (VIF) was employed to assess the degree of multicollinearity between features in both the training and test datasets. VIF scores offer insights into potential redundancy among variables, with values above 5 generally indicating high multicollinearity, which can affect model performance by introducing noise or making it difficult to isolate the independent effects of each feature. This section compares the multicollinearity results between the training and test datasets, highlighting key differences and their implications for model building.

1. Multicollinearity in the training data

In the training dataset, multicollinearity was detected in several key features, most notably in C2 and C5, both of which exhibited relatively high VIF scores (refer Table. 2):

- **A_Numeric, A_Missing, Subject_Yes, and Subject_No:** These features showed infinite VIF scores due to perfect collinearity. This is expected, as A_Numeric and A_Missing are binary complements, and Subject_Yes and Subject_No are mutually exclusive categories. While this type of multicollinearity is not problematic in terms of feature interpretation, redundant features (e.g., A_Missing or Subject_No) should be removed from the model to avoid unnecessary duplication of information hence they have been dropped in further analysis.

- **C2:** This feature had the highest VIF score in the training data, indicating strong multicollinearity with other features, particularly with C5 and C13, as noted in the correlation analysis. A VIF score above 5 suggests that C2 may introduce some redundancy in the model, which could affect its ability to determine the independent effect of this feature.

- **C5:** This feature also demonstrated a high VIF score, indicating multicollinearity with other features, particularly C7 and C9. Although the VIF score is slightly below 5, it is close enough to warrant consideration during model building to ensure that collinear relationships do not negatively affect model performance.

- **C7, C12, and C13**: These features exhibited moderate VIF scores (between 2.5 and 3), indicating some multicollinearity but not to a level that would necessarily affect model interpretation or performance. Nevertheless, these features should still be monitored during model evaluation to ensure they do not introduce excessive noise.

- **Remaining features:** Features such as Distance, B, C1, C3, and C4 had relatively low VIF scores (<2), indicating little to no multicollinearity. These features are unlikely to introduce redundancy and can be confidently included in the model. For full VIF score refer to Table. A2 in appendix.

Variable	VIF Score
A_Numeric, A_Missing, Subject_Yes, Subject_No	Infinity
C2	5.302
C5	4.443
C7, C12, and C13	2.5 to 3

Table. 2 Top VIF score in training data

2. Multicollinearity in the test data

In contrast to the training data, the test dataset showed significantly lower multicollinearity overall (refer Table. 3):

- **A_Numeric, A_Missing, Subject_Yes :** As in the training data, these features exhibited infinite VIF scores in the test data due to their complementary binary nature. This result is expected, and it can be managed by removing one of the redundant features (e.g., A_Missing or Subject_No) in both datasets to avoid unnecessary duplication.

- **C2**: In the test data, the VIF score for C2 was substantially lower than in the training data, dropping from 5.302 to 1.443. This reduction suggests that C2 has less overlap with other features in the test dataset, making it less redundant and easier to interpret in this context. This difference could lead to improved model performance when applying the model to unseen data.

- **C5**: Similarly, the VIF score for C5 was reduced in the test data, falling from 4.443 in the training data to 1.873. This decrease suggests that C5 is less

collinear with other features in the test dataset, which may reduce the risk of overfitting and improve the clarity of its contribution to the model.

- **C6 and C10**: These features maintained moderate VIF scores in both datasets, with values of approximately 1.860 and 1.813 in the test set, respectively. Although these features were somewhat collinear in the training data, their VIF scores remained below 2 in the test data, indicating a manageable level of multicollinearity that is unlikely to cause significant issues.

- **Remaining Features**: Most other features in the test data, such as Distance, B, C1, C3, and C4, had VIF scores below 2, indicating minimal multicollinearity. These features do not exhibit any significant redundancies and can be safely included in the model without further adjustments. For full VIF score refer to Table. A3 in appendix.

Variable	VIF Score
A_Numeric, A_Missing, Subject_Yes	Infinity
C2	1.443
C5	1.873
C7, C12, and C13	2.5 to 3

Table. 3 Top VIF score in test data

Comparison between training and test data

When comparing the multicollinearity results between the training and test datasets, some notable differences emerged:

- **Reduction in multicollinearity for key features**: Features such as C2 and C5, which had high VIF scores in the training data (5.302 and 4.443, respectively), exhibited much lower VIF scores in the test data (1.443 and 1.873). This reduction suggests that these features are less collinear with other features in the test data, which could lead to better model performance and clearer interpretation when applying the model to unseen data.

- **Consistency in infinite VIF scores**: Features with binary or mutually exclusive categories (such as *A_Numeric, A_Missing or Subject_Yes* and *Subject_No*) consistently showed infinite VIF scores in both datasets. This was expected due

to their perfect collinearity, and as in the training data, redundant features should be removed from the model to prevent duplication.

- **Overall lower multicollinearity in the test data**: In general, the test data exhibited lower multicollinearity compared to the training data. Most features in the test set had VIF scores below 2, indicating minimal collinearity and suggesting that the relationships between features are more balanced in the test data. This may result in more straightforward feature interpretation and improved generalisability of the model when applied to unseen data.

3.3.4. Feature selection for analysis

In the project, the primary reason for feature selection was to address multicollinearity. In the training data, certain features exhibited perfect multicollinearity, as indicated by their VIF scores of inf. Specifically, the binary features *A_Numeric* and *A_Missing*, and *Subject_Yes* and *Subject_No* were perfectly collinear. Since these features provide identical information in a complementary manner, retaining both would be redundant. To resolve this *Subject_No* and *A_Missing* were dropped. This reduced redundancy in the feature set while preserving the overall information needed for model training.

Similarly, in the test data, the feature *A_Missing* exhibited perfect collinearity with *A_Numeric*. To handle this, *A_Missing* was dropped from the test data to maintain consistency with the training dataset and eliminate multicollinearity.

By dropping these redundant features, it was ensured that the model is not impacted by multicollinearity, which can lead to inflated coefficients and reduced interpretability. This careful selection of features enhances model performance by simplifying the data without sacrificing valuable information.

3.4 Classification models

3.4.1. Model selection

The selection of machine learning models was influenced by several factors, including the ability to handle both categorical and numerical data and effectiveness in binary classification tasks. Since the dataset no longer contains missing values (due to

preprocessing steps like binary encoding of categorical data), models were chosen based on their performance in handling transformed features, robustness in binary classification, and interpretability.

The following models were selected for this study:

- **Logistic Regression**: Logistic Regression works well with encoded features and provides an easily interpretable relationship between the input features and the likelihood of student enrolment. Categorical data, such as *A_Numeric* and *Additional Subjects*, were binary encoded, which made them suitable for input into the Logistic Regression model, ensuring that the dataset was fully prepared for this linear approach.

- **Decision Tree Classifier**: Decision Trees are highly interpretable and can naturally handle a mix of categorical and numerical data. After preprocessing and encoding the categorical variables, Decision Trees were well-suited to work with the transformed binary features (e.g., *A_Numeric*). The Decision Tree's ability to split based on these binary flags ensured that it could create decision paths effectively for predicting student enrolment.

- **Random Forest Classifier**: Random Forest, an ensemble of decision trees, was selected for its robustness in handling both categorical and numerical features, especially after the encoding process. With the binary encoding of categorical variables, Random Forest's ability to build multiple decision trees based on different subsets of data makes it highly effective in capturing complex patterns without needing to worry about missing data.

- **XGBoost Classifier**: XGBoost was selected for its powerful performance in binary classification tasks. XGBoost uses gradient boosting techniques to iteratively improve the model's accuracy by minimizing errors from prior iterations. After preprocessing, with categorical variables binary-encoded and all missing data handled, XGBoost was highly effective in working with the clean dataset. Its advanced handling of both categorical and numerical data can make it a strong candidate for improving predictive performance in student enrolment classification.

- **Support Vector Machine (SVM)**: SVM was chosen for its ability to find optimal boundaries in high-dimensional spaces. With the dataset fully preprocessed and all missing values transformed into binary features (e.g., *A_Missing, Additional Subjects*), SVM was able to process the dataset without issue. Its ability to handle complex relationships between students' features and enrolment outcomes made it an excellent choice for binary classification.

3.4.2. Data splitting and model training

To ensure that the models developed in this study could generalise well to unseen dataset for the year 2024 (test dataset), the dataset for the year 2023 (training dataset) was first split into a training set and a holdout set. This process allowed the models to learn patterns from the training set and then be evaluated on the holdout set, providing an initial assessment of their performance before testing on the unseen dataset for the year 2024.

Holdout set and training/test split

Before evaluating the models on the test data, a holdout set was used to assess performance. The holdout set, a subset of the training data, was not used for model training but served as an unbiased evaluation tool to prevent overfitting and ensure the models could generalise to new data. The dataset was split, with 80% used for training and 20% reserved as the holdout set. Models were trained on the 80% portion, then evaluated on the 20% holdout to gauge initial performance. Afterward, the models were tested on the pre-processed test data, using the same evaluation metrics as the holdout set for consistency. This approach ensured robust, generalizable performance across both the holdout and unseen test data.

3.4.3. Model evaluation

Model evaluation is a critical step in assessing the performance and reliability of machine learning models. It involves using various metrics to quantify how well a model makes predictions on unseen data. The goal of model evaluation is to ensure that the model not only performs well on the training data but also generalises effectively to new, unseen data. All the metrics were the same for both the holdout set and the test data

and they were calculated with the help of confusion metrics for each model. The metrics include:

1. Accuracy: It measures the proportion of correctly predicted instances out of the total instances. It is useful for understanding the overall performance of the model.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

2. Precision: It measures the proportion of true positive predictions out of all positive predictions made by the model. It is particularly important when the cost of false positives is high.

$$Precision = \frac{TP}{TP + FP}$$

3. Recall (Sensitivity): Recall is also called sensitivity; it measures the proportion of true positive predictions out of all actual positives in the dataset. It is crucial when the cost of false negatives is high.

$$Recall = \frac{TP}{TP + FN}$$

4. F1 Score: The F1 Score is the harmonic mean of precision and recall, providing a balanced measure of the model's performance, especially in cases of class imbalance.

$$F1Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

5. AUC-ROC: AUC-ROC measures the area under the Receiver Operating Characteristic curve, which plots the true positive rate against the false positive rate at various threshold settings. It provides an aggregate measure of performance across all classification thresholds.

Where:

- **True Positive (TP):** The model correctly predicts a positive class (e.g., correctly predicting a student will enrol).

- **False Positive (FP):** The model incorrectly predicts a positive class (e.g., predicting a student will enrol when they do not).
- **True Negative (TN):** The model correctly predicts a negative class (e.g., correctly predicting a student will not enrol).
- **False Negative (FN):** The model incorrectly predicts a negative class (e.g., predicting a student will not enrol when they do).

3.4.4. Model evaluation on a hold-out set

The evaluation of the models on the holdout dataset was carried out using standard performance metrics, including Accuracy, Precision, Recall, and the F1-Score. Additionally, the models were evaluated using ROC-AUC curves to understand the trade-off between true positive and false positive rates refer to Table 3.1 and refer to Fig A3 in appendix for confusion metrics.

Logistic Regression and XGBoost emerged as the top-performing models based on both Accuracy and F1-Score. These models demonstrated a balanced trade-off between precision and recall, with Logistic Regression having the highest Recall, making it highly effective in identifying positive instances. Random Forest showed strong Recall and performed well overall, with an F1-Score of 0.7797, indicating it effectively balanced precision and recall but fell short compared to the top models.

SVM had decent Precision but struggled in Recall, resulting in a slightly lower F1-Score of 0.7059 compared to other models. Decision Tree had the lowest performance overall, with an Accuracy and the lowest F1-Score, indicating that it did not generalise as well on the holdout set.

	Decision tree	Random Forest	Logistic regression	XGBoost	SVM
Accuracy	0.6981	0.7547	0.8113	0.8113	0.717
Precision	0.7037	0.7188	0.7429	0.7576	0.75
Recall	0.7037	0.8519	0.963	0.9259	0.6667
F1-Score	0.7037	0.7797	0.8387	0.8333	0.7059

Table.3.1 Performance metrics of models on the hold-out set

The ROC curve, as illustrated in Fig.3.2, provides a graphical representation of the trade-off between the true positive rate and the false positive rate for each model. The

Area Under the Curve (AUC) is a useful indicator of overall model performance, with values closer to 1 indicating better performance.

XGBoost had the highest AUC of 0.91, indicating strong performance in distinguishing between classes. Logistic Regression also performed well, with an AUC of 0.88, confirming its effectiveness, particularly in high recall scenarios. Random Forest had an AUC of 0.84, reflecting solid model performance, while SVM also showed good performance with an AUC of 0.83. Decision Tree had the lowest AUC at 0.69, confirming its weaker performance compared to other models.

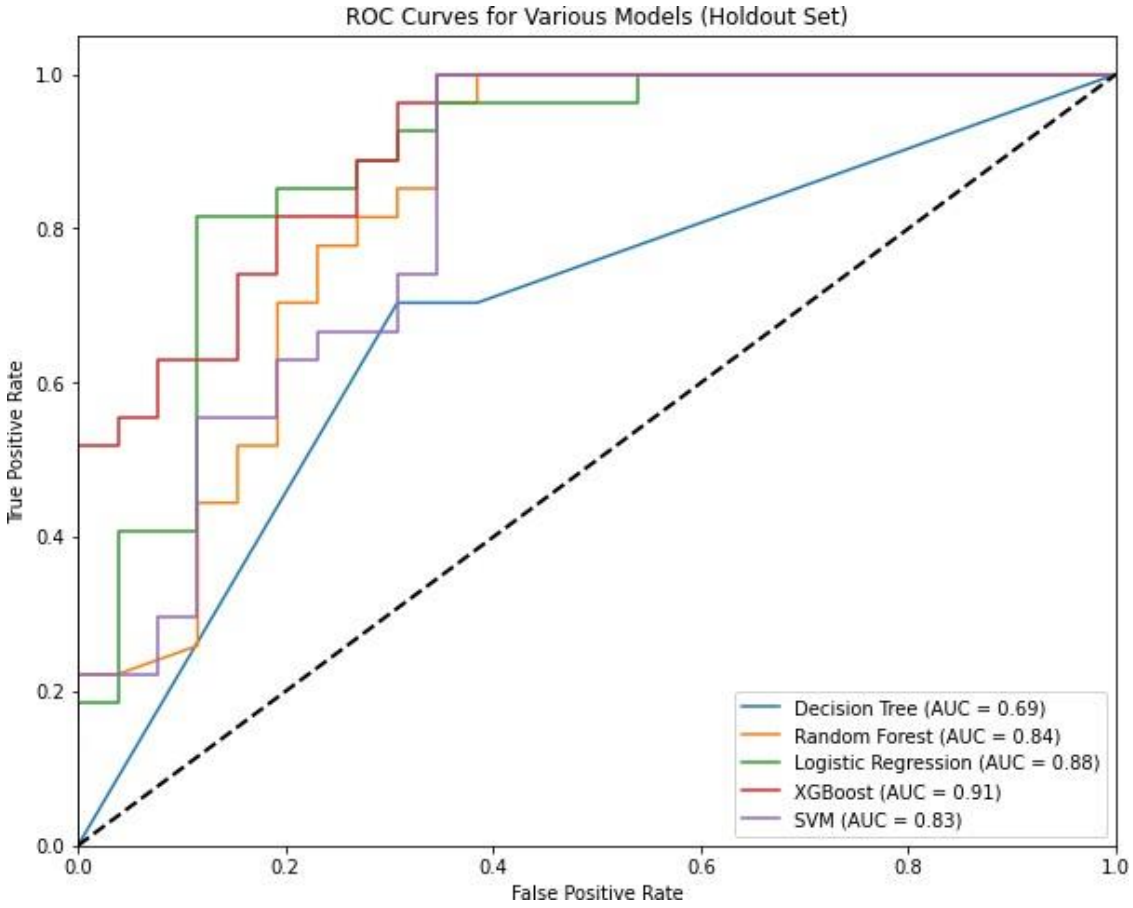

Fig.3.2 ROC curve of models on the hold-out set

Overall, the evaluation shows that Logistic Regression and XGBoost are the most balanced and effective models based on their high accuracy, recall, and F1-score, as well as strong AUC values from the ROC curves.

3.4.5. SMOTE and Hyperparameter tuning

During the analysis, it was found that the performance metrics on the holdout set were doing exceptionally well, but the model performance degraded on the test data. The dataset was then checked for class imbalance, and it was found that the class distribution before Synthetic Minority Over-sampling Technique (SMOTE) was as follows:

- Class **1**: 142 samples
- Class **0**: 122 samples

This indicates a slight class imbalance in the data, with class 1 having more samples than class 0. Although the imbalance is not extreme, it could still affect the model's generalisation, leading to overfitting to the holdout set. To address this issue, SMOTE was applied to balance the classes, aiming to improve the model's performance on the test data however, this did not impact the model's performance.

Another strategy was used in which model parameters were tuned and performance metrics were optimised on the holdout set, the models were then evaluated on the final test set, which was never seen by the model during training, further discussed in the result section. This ensured an accurate assessment of how well the models would perform on unseen data and provided insight into their true predictive capabilities. The effect of hyper tuning is discussed in detail in the result section.

This approach allowed for both an intermediate evaluation using the holdout set and a final evaluation using the test set, ensuring that the models were not only trained effectively but also evaluated thoroughly before being applied to new data.

4. Results

In this chapter, the performance of various machine learning models in predicting student enrolment is evaluated. First, the models' performance on the test data is compared with their earlier performance on the holdout set, focusing on key metrics such as accuracy, precision, recall, F1-score, and AUC-ROC. Following this, the models undergo hyperparameter tuning to optimise their performance, and the results after tuning are analysed and compared to the pre-tuning results. This chapter also discussed the categorisation of students on the predicted probabilities i.e. how the students have been categorised in different categories within the interest of the WMG academy. This analysis aims to identify the most effective model and demonstrate the impact of hyperparameter tuning on improving prediction accuracy.

4.1 Model performance on test data before hyperparameter tuning

Before applying hyperparameter tuning, the machine learning models were evaluated on the test dataset to assess their initial performance. The classifications models were tested and one-on-one performance comparison of models on holdout set and test data, using accuracy, precision, recall, F1-score, and AUC-ROC curves, are presented below. The combined AUC-ROC curve refers to Fig.4.1. Additionally, these outcomes are compared with the results from the holdout set to provide a comprehensive understanding of the model's performance on unseen data. The confusion metrics of the model can be found in Fig. A3.

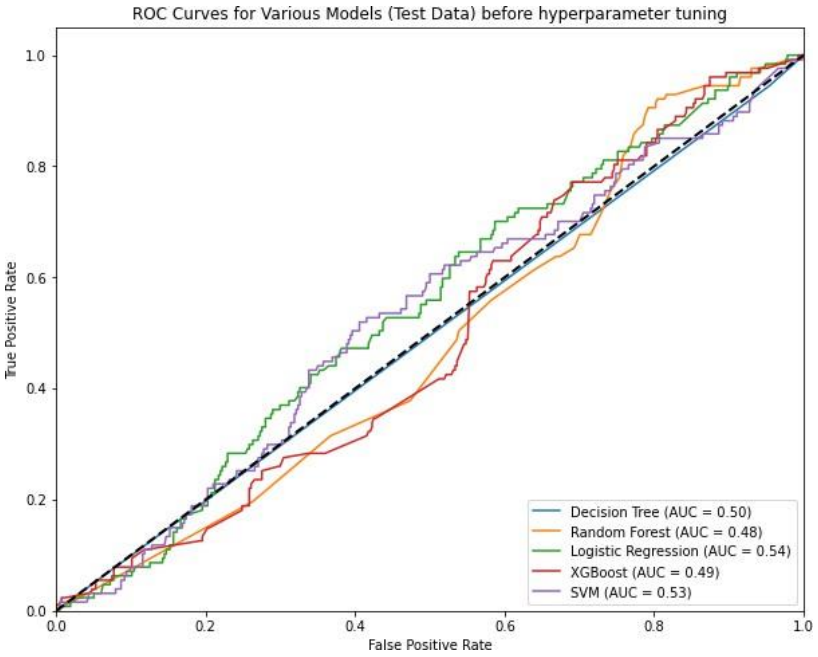

Fig.4.1 ROC curves of models on the test data

1. Decision Tree performance

The Decision Tree model demonstrated a sharp decline in accuracy on the test data, dropping from 0.6981 on the holdout set to only 0.2569 on the test data refer Table.4.1. This significant reduction indicates that the Decision Tree may have overfitted the training data, making it less effective at generalising to new data. The precision also decreased, from 0.7037 to 0.2330, reflecting a higher rate of false positives in the test data.

However, the recall saw a major increase, from 0.7037 in the holdout set to 0.9449 in the test set. This indicates that while the model struggled with accuracy and precision, it was highly sensitive, successfully identifying almost all students who enrolled. Despite the high recall, the poor precision resulted in a lower F1-score of 0.3738, which is much lower than the holdout set score of 0.7037.

The AUC-ROC score of 0.50 further underscores the model's poor performance on the test set, indicating that the model performed no better than random chance when distinguishing between enrolled and non-enrolled students.

	Holdout Set	Decision Tree
Accuracy	0.6981	0.2569
Precision	0.7037	0.233
Recall	0.7037	0.9449
F1-Score	0.7037	0.3738
AUC-ROC	0.69	0.50

Table 4.1. Performance metrics of Decision Tree on test data before hyper parameter tuning

2. Random Forest performance

The Random Forest model also experienced a substantial drop in accuracy, from 0.7547 in the holdout set to 0.2847 on the test data refer Table. 4.2. This suggests that the model's generalisation capability was limited when applied to new data. The precision decreased from 0.7188 to 0.2421, like the Decision Tree model, indicating an increase in false positive predictions.

Interestingly, the recall remained very high on the test data, increasing slightly to 0.9606 compared to 0.8519 in the holdout set. This suggests that the model was effective at identifying enrolled students, though it struggled to balance this sensitivity with precision, leading to a lower F1-score of 0.3867.

The AUC-ROC score dropped from 0.84 on the holdout set to 0.48 on the test set, signalling a considerable decline in the model's discriminatory power when applied to unseen data.

	Holdout Set	Random Forest
Accuracy	0.7547	0.2847
Precision	0.7188	0.2421
Recall	0.8519	0.9606
F1-Score	0.7797	0.3867
AUC-ROC	0.84	0.48

Table 4.2. Performance metrics of Random Forest on test data before hyper parameter tuning

3. Logistic Regression performance

Logistic Regression showed a more moderate decline in performance compared to the other models refer Table 4. The accuracy dropped from 0.8113 on the holdout set to 0.3937 on the test data, though it remained higher than the Decision Tree and Random Forest models. The precision saw a significant drop from 0.7429 to 0.2531, indicating that the model struggled with false positive predictions in the test data.

The recall, while lower than the holdout set value of 0.9630, remained relatively high at 0.8110, meaning that the model continued to perform well in identifying students who enrolled. However, the drop in precision led to a lower F1-score of 0.3858, a marked decrease from 0.8387 in the holdout set.

The AUC-ROC score of 0.54 suggests that Logistic Regression performed slightly better than random chance at distinguishing between classes on the test data, though its discriminatory power was reduced compared to the holdout set score of 0.88.

	Holdout Set	Logistic Regression
Accuracy	0.8113	0.3937
Precision	0.7429	0.2531
Recall	0.963	0.811
F1-Score	0.8387	0.3858
AUC-ROC	0.88	0.54

Table 4.3. Performance metrics of Logistic Regression on test data before hyper parameter tuning

4. XGBoost performance

XGBoost was one of the strongest performers on the holdout set, with an accuracy of 0.8113 and an AUC-ROC score of 0.91 refer Table. 4.4. However, its performance dropped significantly on the test data. The accuracy fell to 0.3050, and the precision dropped to 0.2485, suggesting an increase in false positives when applied to unseen data.

Despite this, XGBoost achieved the highest recall of 0.9685 on the test data, meaning that it successfully identified almost all students who enrolled. The strong recall resulted in a relatively higher F1-score of 0.3955, though still lower than the 0.8333 achieved on the holdout set.

The AUC-ROC score for XGBoost on the test data was 0.49, indicating that the model struggled to effectively differentiate between enrolled and non-enrolled students in this dataset.

	Holdout Set	XGBoost
Accuracy	0.8113	0.305
Precision	0.7576	0.2485
Recall	0.9259	0.9685
F1-Score	0.8333	0.3955
AUC-ROC	0.83	0.49

Table 4.4. Performance metrics of XGBoost on test data before hyperparameter tuning

5. Support Vector Machine (SVM) performance

SVM demonstrated moderate performance on the holdout set, with an accuracy of 0.7170 and an AUC-ROC score of 0.83 refer Table. 4.5. However, on the test data, the model's performance declined, with accuracy dropping to 0.2773. The precision decreased significantly to 0.2317, indicating a higher rate of false positives compared to the holdout set.

The recall, however, improved on the test data, increasing to 0.8976 from 0.6667 on the holdout set, suggesting that SVM was successful at identifying most of the enrolled students. Despite the improvement in recall, the overall balance between precision and recall was not optimal, resulting in a lower F1-score of 0.3683.

The AUC-ROC score for SVM on the test data was 0.53, suggesting only a slight improvement over random guessing, compared to the much higher 0.83 on the holdout set.

	Holdout Set	SVM
Accuracy	0.717	0.2773
Precision	0.75	0.2317
Recall	0.6667	0.8976
F1-Score	0.7059	0.3683
AUC-ROC	0.83	0.53

Table 4.5. Performance metrics of SVM on the test data before hyperparameter tuning

4.2 Performance of models after hyper parameter tuning

The models have different parameters which were tuned to improve the generalisability of the models on the test data. Table 4.6 shows the parameters of all the models.

Models	Hyperparamters	Values
Decision Tree	max_depth	10, 20, 30, None
	min_samples_split	2, 5, 10
	min_samples_leaf	1, 2, 4
Random Forest	n_estimators	100, 200, 500, 1000
	max_depth	10, 20, 30, None
	min_samples_split	2, 5, 10
	min_samples_leaf	1, 2, 4
	max_features	auto, sqrt, log2
Logistic Regression	C	0.01, 0.1, 1, 10, 100
	Penalty	l1, l2
	Solver	liblinear, saga
SVM	C	0.01, 0.1, 1, 10, 100
	Kernel	linear, rbf, poly
	Gamma	scale, auto
XGBoost	learning_rate	0.01, 0.1, 0.3
	max_depth	3, 6, 9
	n_estimators	100, 200, 500
	Subsample	0.6, 0.8, 1.0

Table. 4.6 Hyper parameters of models

Performance on the hold-out set

The performance of the tuned models on the holdout set remained relatively consistent with their pre-tuning results, as the models already exhibited strong performance in this phase. The minimal change suggests that the models were well-calibrated on the holdout data even before hyperparameter tuning. For detailed performance metrics of the tuned models, refer to Table 4.7, and for confusion matrices, see Figure A4 in the Appendix.

	Decision tree	Random Forest	Logistic regression	XGBoost	SVM
Accuracy	0.7925	0.8302	0.8113	0.8302	0.8491
Precision	0.7667	0.75	0.7429	0.75	0.7714
Recall	0.8519	1	0.963	1	1
F1-Score	0.807	0.8571	0.8387	0.8571	0.871

Table. 4.7 Performance metrics of SVM on the hold-out data after hyper parameter tuning

Performance on the preprocessed test data

Once the hyperparameters were optimised, the models were evaluated again on the pre-processed test data, and ROC (Receiver Operating Characteristic) curves were plotted for each model refer Fig. 4.2. These curves, combined with the AUC scores, provide a detailed view of each model's ability to distinguish between students who are enrolled and those who are not.

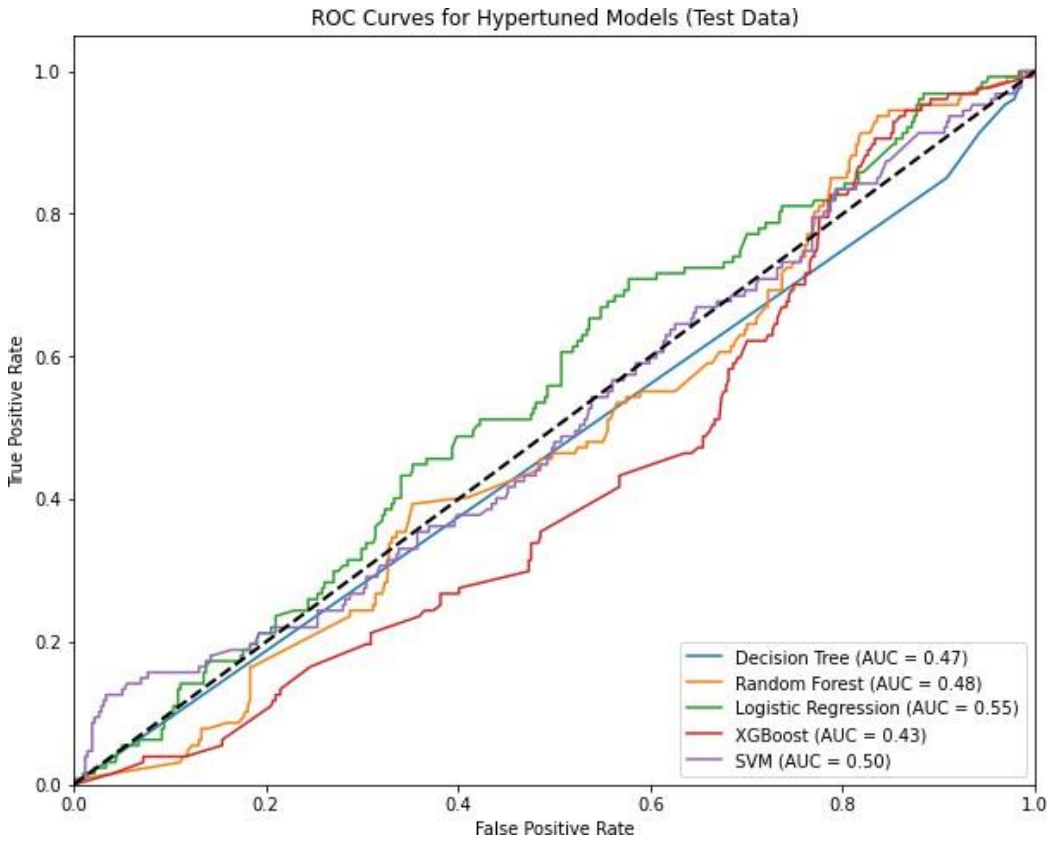

Fig. 4.2 ROC curve of the models on the test-data after hyper parameter tuning

Here's the breakdown of the ROC and AUC results after tuning:

1. Logistic Regression: After tuning, Logistic Regression had the highest AUC of all models at 0.55. This score suggests that Logistic Regression was the best at distinguishing between students likely to enrol and those unlikely to enrol, even though the performance was modest overall. Logistic Regression showed consistent improvements across accuracy and precision, especially when tuned and evaluated at a threshold of 0.7. This model successfully balanced between minimizing false positives and achieving high recall.

2. Random Forest: Random Forest improved slightly, with an AUC of 0.48. While the overall discriminative power remained modest, the model performed better when optimised. Hyperparameter tuning helped Random Forest increase its accuracy from 0.2847 to 0.3383, particularly at the 0.7 threshold, where the model's F1-score reached 0.4013. The ROC curve shows a slight improvement, but the Random Forest model still faced challenges in handling the ambiguity between classes.

3. SVM: SVM performed better than the Decision Tree and XGBoost models, with an AUC of 0.50. However, it still exhibited limitations in classifying students correctly, and the model struggled with false positives. The overall accuracy for SVM improved post-tuning, but not to the same extent as the Logistic Regression model. At the 0.7 threshold, the SVM's accuracy was 0.2588, and while it achieved a respectable recall, precision remained low, indicating a high number of false positives.

4. XGBoost: Surprisingly, XGBoost, which typically performs well in many scenarios, had an AUC of only 0.43, indicating that it struggled significantly with this dataset even after tuning. The ROC curve confirms that XGBoost did not perform well in separating the two classes. Despite tuning, XGBoost had low accuracy and precision, and its ROC curve shows that the model was only slightly better than random guessing.

5. Decision Tree: The Decision Tree model's AUC remained low at 0.47, reflecting that the model's performance was still suboptimal. Even after tuning, it struggled with generalisation and overfitting. Post-tuning, the Decision Tree's accuracy was 0.2588, and although it had a high recall at the 0.7 threshold, precision remained low, indicating a propensity for false positives.

4.3 Thresholding and classification of results

Given the poor initial results of the models on the test data, the impact of varying classification thresholds was explored (refer to Table A4 in the appendix). This adjustment helped refine the point at which students were classified as "enrolled" or "not enrolled." It was particularly relevant because the standard 0.5 threshold is not always ideal for all models. Instead, an optimal threshold was calculated for each model, as shown in Table 4.8. This optimal threshold represents the point where the difference between the True Positive Rate (TPR) and False Positive Rate (FPR) is maximised, offering the best trade-off between accuracy and false positive reduction.

Model	Optimal Thresholds
Decision Tree	0.5
Random Forest	0.7205
Logistic Regression	0.6595
XGBoost	0.6426
SVM	0.9353

Table. 4.8 Optimal thresholds for models

After careful evaluation across multiple thresholds (0.3, 0.4, 0.5, 0.6, and 0.7), a final 0.7 threshold was selected for classification. This threshold yielded the highest accuracy for most models (refer to Table 4.9), particularly for Logistic Regression (accuracy 0.4972) and Random Forest (accuracy 0.3383). By applying this higher threshold, the models focused more on reducing false positives (students incorrectly classified as enrolled), which had been a significant issue at lower thresholds.

	Decision Tree	Random Forest	Logistic Regression	XGBoost	SVM
Accuracy	0.2588	0.3383	0.4972	0.3272	0.2588
Precision	0.2292	0.2548	0.2698	0.2473	0.2365
Recall	0.9134	0.9449	0.6693	0.9134	0.9685

F1-Score	0.3665	0.4013	0.3846	0.3893	0.3802

Table. 4.9 Model performance (threshold = 0.7) on the test data after hyper parameter tuning

The 0.7 threshold improved the overall model performance by reducing false positives. The Logistic Regression model demonstrated significant improvements, achieving a balanced F1-Score of 0.3846 and precision of 0.2698. Similarly, Random Forest exhibited enhanced precision and recall, resulting in an F1-Score of 0.4013. These metrics indicate that by focusing on the 0.7 threshold, the models were better equipped to identify students more accurately, reducing the number of false positives and misclassifications.

Categorisation process

In addition to thresholding, a categorisation process was introduced to more effectively handle ambiguous predictions. Students with predicted probabilities between 0.25 and 0.7 were classified as "Undecided." This method helped refine the classification process by addressing borderline cases with greater precision. It proved particularly beneficial for the Logistic Regression model, which was ultimately selected as the most reliable due to its consistent performance across multiple metrics. This final model demonstrated superior accuracy and balance in managing uncertain predictions, making it the optimal choice for classifying student enrolment outcomes.

For instance, the "Undecided" category was used to classify students whose predicted probabilities were close to 0.5, making it difficult for the models to confidently assign them as "enrolled" or "not enrolled." By marking these cases as "Undecided," the models were able to avoid overconfident misclassifications, improving overall precision and reducing false positives. This categorisation method was instrumental in refining the final predictions, ensuring a more balanced and reliable output, especially when combined with the 0.7 threshold.

4.4 Results summary

The results of this study provided a comprehensive evaluation of various machine learning models in predicting student enrolment at WMG Academy, focusing on two

phases: initial performance before hyperparameter tuning and performance after optimisation.

In the initial evaluation, all models experienced a significant drop in accuracy and precision when tested on unseen data compared to their performance on the holdout set. This decline is likely due to the limited dataset size, which restricted the models' ability to generalise effectively. Additionally, class imbalance contributed to a higher rate of false positives, particularly in models like Decision Tree, Random Forest, and XGBoost. Although models such as Logistic Regression and XGBoost maintained high recall rates, their low precision resulted in reduced F1-scores, indicating difficulty in distinguishing between enrolled and non-enrolled students. The modest AUC-ROC scores across the models further reflect their initial lack of strong discriminatory power.

After hyperparameter tuning, most models showed notable improvements. Logistic Regression emerged as the best-performing model, with an AUC of 0.55 on the test data. Random Forest and SVM also displayed improvements, particularly in accuracy and precision, though they continued to face challenges with class imbalance and borderline cases. Interestingly, XGBoost, despite its strong performance on the holdout set, underperformed post-tuning, with a lower AUC of 0.43, highlighting its difficulties in separating the two classes in the test set.

One crucial aspect of post-tuning optimisation was the exploration of optimal classification thresholds. By adjusting the threshold from the default 0.5 to 0.7, the models—particularly Logistic Regression and Random Forest—were able to significantly reduce false positives, striking a better balance between precision and recall. Additionally, the introduction of an "Undecided" classification for ambiguous predictions improved the models' ability to manage uncertainty in borderline cases. This categorisation process helped refine the decision-making process and reduced misclassifications, making the predictions more reliable.

For WMG Academy, the insights gained from these models are valuable in informing strategic enrolment decisions. The ability to predict student enrolment with a higher degree of accuracy allows the academy to better allocate resources, optimise course planning, and anticipate staffing needs. Moreover, reducing false positives—students predicted to enrol but who do not—helps the academy minimise wasted resources. The

final tuned models, particularly Logistic Regression, provide actionable insights that can support more informed, data-driven enrolment strategies at WMG Academy.

5. Conclusions

This dissertation aimed to develop predictive models to forecast student enrolment in the sixth form at WMG Academy, using advanced machine learning techniques. By analysing key factors such as academic performance, socio-economic background, subject preferences, and geographic proximity, the research sought to provide valuable insights that could assist the academy in optimising resource allocation and improving recruitment strategies.

Several machine learning models, including Logistic Regression, Random Forest, Decision Trees, XGBoost, and Support Vector Machines, were trained on 2023 data and tested on 2024 data to assess their predictive performance. The models underwent evaluation using various metrics like accuracy, precision, recall, F1 score, and AUC-ROC. Results showed that Logistic Regression emerged as the most balanced and effective model, particularly after hyperparameter tuning, which improved its performance by addressing class imbalance and optimising classification thresholds.

However, the dissertation highlighted key challenges in predictive modelling, such as data imbalance, differences between training and testing datasets, and overfitting, which affected the models' generalisation to unseen data. Although hyperparameter tuning and threshold adjustments improved model performance, challenges like poor generalisation of models like XGBoost indicate the need for further research to enhance accuracy, particularly in handling complex datasets.

In conclusion, the dissertation demonstrated the value of predictive modelling in student enrolment but also underscored the complexities of applying these models in real-world educational settings. By using insights gained from these models, WMG Academy can better plan for future enrolments and allocate resources more effectively. Further refinement of models and data collection methods could enhance the predictive capabilities and help institutions like WMG Academy make more informed, data-driven decisions.

6. References

AJZEN, I. 1991. The theory of planned behavior. *Organizational behavior and human decision processes,* 50**,** 179-211.

BAKER, D. J. & BRITTON, T. 2024. Hate crimes and Black college student enrolment. *Education Finance and Policy,* 19**,** 187-217.

CHEN, X. 2013. STEM Attrition: College Students' Paths into and out of STEM Fields. Statistical Analysis Report. NCES 2014-001. *National Center for Education Statistics*.

COLLOM, G. D. 2023. A quasi-experimental investigation of adult student enrolment responses to the Tennessee Reconnect Grant. *Community College Journal of Research and Practice,* 47**,** 478-493.

GORARD, S. & SIDDIQUI, N. 2018. Grammar schools in England: A new analysis of social segregation and academic outcomes. *British Journal of Sociology of Education,* 39**,** 909-924.

HEMSLEY-BROWN, J. & OPLATKA, I. 2015a. *Higher education consumer choice*, Springer.

HEMSLEY-BROWN, J. & OPLATKA, I. 2015b. University choice: what do we know, what don't we know and what do we still need to find out? *International Journal of Educational Management,* 29**,** 254-274.

HOSSLER, D. & GALLAGHER, K. S. 1987. Studying Student College Choice: A Three-Phase Model and the Implications for Policymakers. *College and University,* 62.

KALIM, U. 2023. The impact of school resources on student enrolment: an empirical investigation of different category public schools in Pakistan. *SN Social Sciences,* 3**,** 85.

KOTSIANTIS, S., PIERRAKEAS, C. & PINTELAS, P. 2004. PREDICTING STUDENTS'PERFORMANCE IN DISTANCE LEARNING USING MACHINE LEARNING TECHNIQUES. *Applied Artificial Intelligence,* 18**,** 411-426.

LAWRENCE, S. & GILES, C. L. Overfitting and neural networks: conjugate gradient and backpropagation. Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium, 2000. IEEE, 114-119.

MANSOR, A. N., NASARUDDIN, M. Z. I. M. & A. HAMID, A. H. 2021. The effects of school climate on sixth form teachers' self-efficacy in Malaysia. *Sustainability,* 13**,** 2011.

MARINGE, F. 2006. University and course choice: Implications for positioning, recruitment and marketing. *International journal of educational management,* 20**,** 466-479.

PERNA, L. W. 2006. Studying college access and choice: A proposed conceptual model. *Higher education: Handbook of theory and research.* Springer.

RUDIN, C. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature machine intelligence,* 1**,** 206-215.

TAYLOR, P. H., REID, W. A. & HOLLEY, B. J. 2022. *The English sixth form: A case study in curriculum research*, Routledge.

THOMAS, E. H. & GALAMBOS, N. 2004. What satisfies students? Mining student-opinion data with regression and decision tree analysis. *Research in Higher Education,* 45**,** 251-269.

WOODWARD, P. 2022. Location and locational capital: an examination of factors influencing choice of higher education applications by working-class students in a sixth-form college. *Educational Review,* 74**,** 1119-1136.

YI, S. & DUVAL-COUETIL, N. 2021. Interdisciplinary entrepreneurship education: Exploring 10-year trends in student enrolment, interest and motivation. *Entrepreneurship Education and Pedagogy,* 4**,** 100-118.

7. Appendix

7.1 Tables

Variables	count	mean	std	min	25%	50%	75%	max
Distance_Train	264	10.8976	5.232	0.6	8.875	10.8976	10.8976	42.1
B_Train	264	0.787879	0.409587	0	1	1	1	1
C1_Train	264	0.018939	0.136757	0	0	0	0	1
C2_Train	264	0.401515	0.491136	0	0	0	1	1
C3_Train	264	0.034091	0.181807	0	0	0	0	1
Enroled?_Train	264	0.537879	0.49951	0	0	1	1	1
A_Numeric_Train	264	0.560606	0.497256	0	0	1	1	1
A_Missing_Train	264	0.439394	0.497256	0	0	0	1	1
Subject_Yes_Train	264	0.30303	0.460441	0	0	0	1	1
Distance_Test	541	8.78824	17.0267	0.5	3.7	5.5	8.78824	220
B_Test	541	0.654344	0.476022	0	0	1	1	1
C1_Test	541	0.053604	0.225444	0	0	0	0	1
C2_Test	541	0.079482	0.270741	0	0	0	0	1
C3_Test	541	0.018484	0.134819	0	0	0	0	1
Enroled?_Test	541	0.23475	0.424235	0	0	0	0	1
A_Numeric_Test	541	0.624769	0.484631	0	0	1	1	1
A_Missing_Test	541	0.375231	0.484631	0	0	0	1	1
Subject_Yes_Test	541	0.066543	0.24946	0	0	0	0	1

Table.A1 Summary statistics of the training and test datasets

Features	VIF
Distance	1.155882
B	1.230271
C1	1.351402
C2	5.302155
C3	1.561931
C4	1.222175
C5	4.443623
C6	1.670346
C7	2.50521
C8	1.306033
C9	1.718532
C10	1.816271
C11	1.806568
C12	2.768342
C13	2.566093
C14	1.876816

A_Numeric	inf
A_Missing	inf
Subject_Yes	inf
Subject_No	inf
Subject_Missing	inf

Table A2. VIF score for the training data

Feature	VIF
Distance	1.083864
B	1.152595
C1	1.05963
C2	1.443183
C3	1.18343
C4	1.058527
C5	1.873137
C6	1.86004
C7	1.629617
C8	1.270788
C9	1.335866
C10	1.813558
C11	1.121505
C12	1.284203
C13	1.321097
C14	1.333935
A_Numeric	inf
A_Missing	inf
Subject_Yes	inf
Subject_Missing	inf

Table A3. VIF score for the test data

Model	Threshold	Accuracy	Precision	Recall	F1-Score
Decision Tree	0.3	0.244	0.2369	1	0.3831
	0.4	0.2458	0.2374	1	0.3837
	0.5	0.2458	0.2374	1	0.3837
	0.6	0.2421	0.2315	0.9606	0.3731
	0.7	0.2588	0.2292	0.9134	0.3665
Random Forest	0.3	0.2366	0.2352	1	0.3808
	0.4	0.2384	0.2356	1	0.3814

	0.5	0.2477	0.2368	0.9921	0.3824
	0.6	0.3124	0.2485	0.9528	0.3941
	0.7	0.3383	0.2548	0.9449	0.4013
Logistic Regression	0.3	0.268	0.2418	0.9921	0.3889
	0.4	0.3087	0.2495	0.9685	0.3968
	0.5	0.3604	0.2436	0.8189	0.3755
	0.6	0.4713	0.2655	0.7087	0.3863
	0.7	0.4972	0.2698	0.6693	0.3846
XGBoost	0.3	0.2348	0.2348	1	0.3802
	0.4	0.2957	0.246	0.9685	0.3923
	0.5	0.305	0.2475	0.9606	0.3935
	0.6	0.3087	0.2485	0.9606	0.3948
	0.7	0.3272	0.2473	0.9134	0.3893
SVM	0.3	0.2421	0.2335	0.9764	0.3769
	0.4	0.2458	0.2344	0.9764	0.378
	0.5	0.2477	0.2338	0.9685	0.3767
	0.6	0.2514	0.2347	0.9685	0.3779
	0.7	0.2588	0.2365	0.9685	0.3802

Table. A4 Model performances on the test data after hyper parameter tuning for different thresholds

7.2 Figures

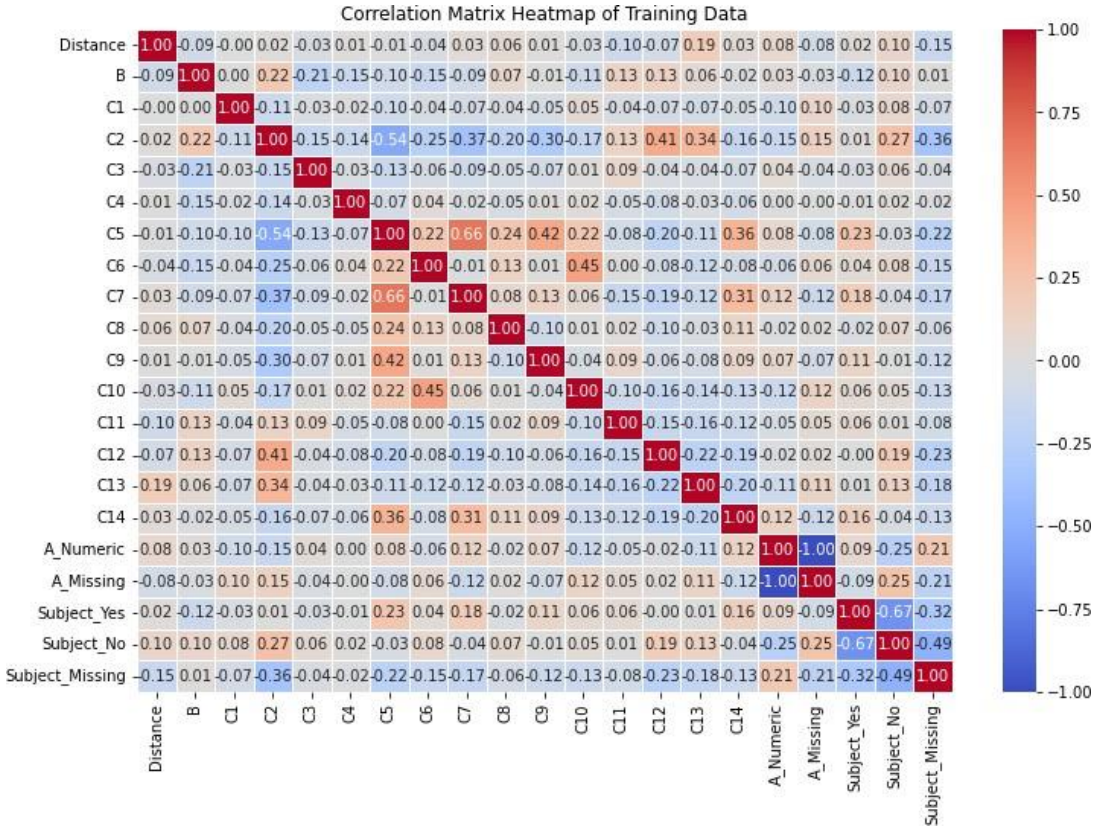

Fig.A1 Correlation matrix heatmap of the training dataset

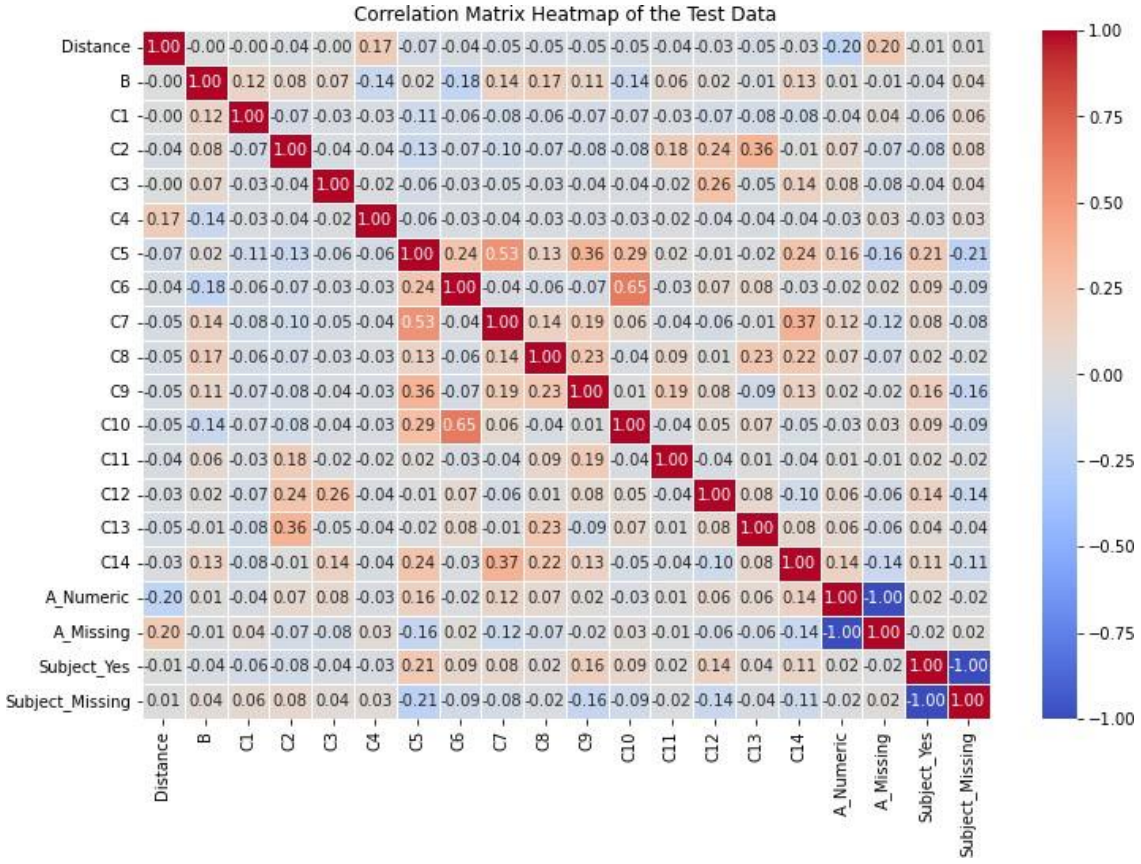

Fig A2. Correlation matrix heatmap of the test dataset

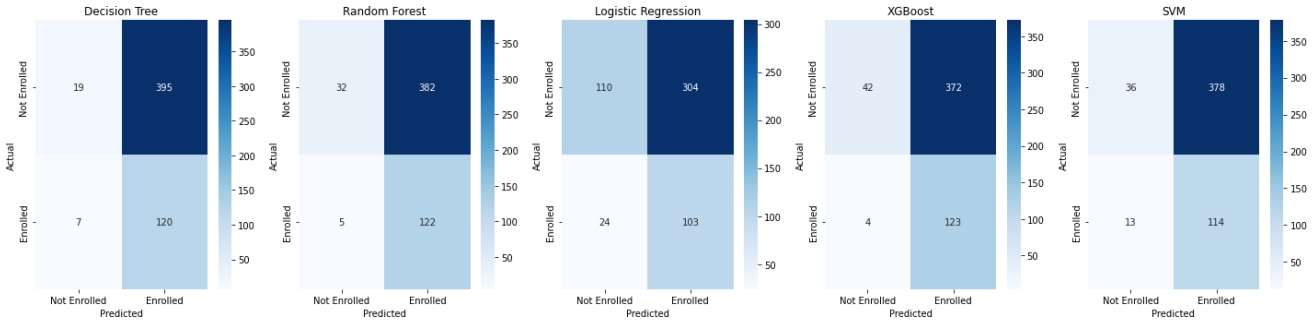

Fig A3. Confusion metrics of models on test data before hyperparameter tuning

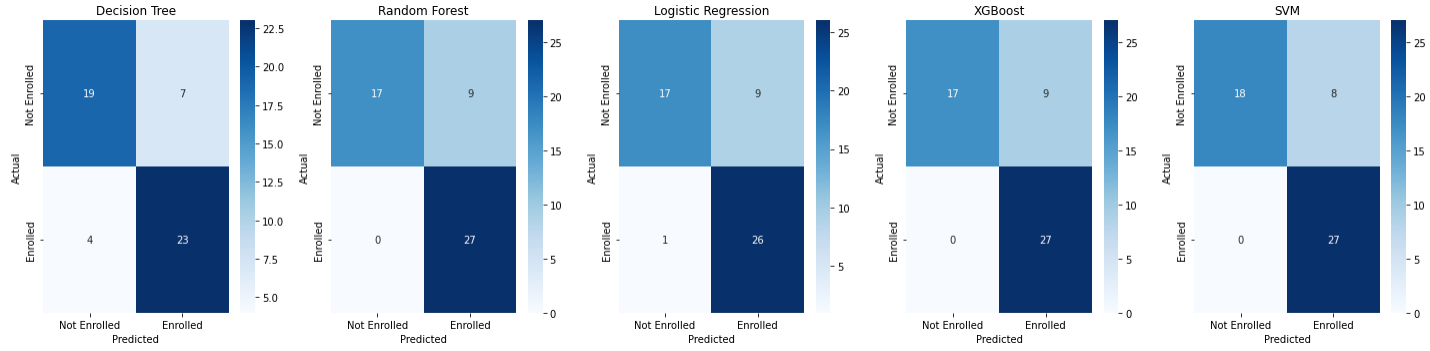

Fig A4. Confusion metrics of models on hold-out data after hyperparameter tuning

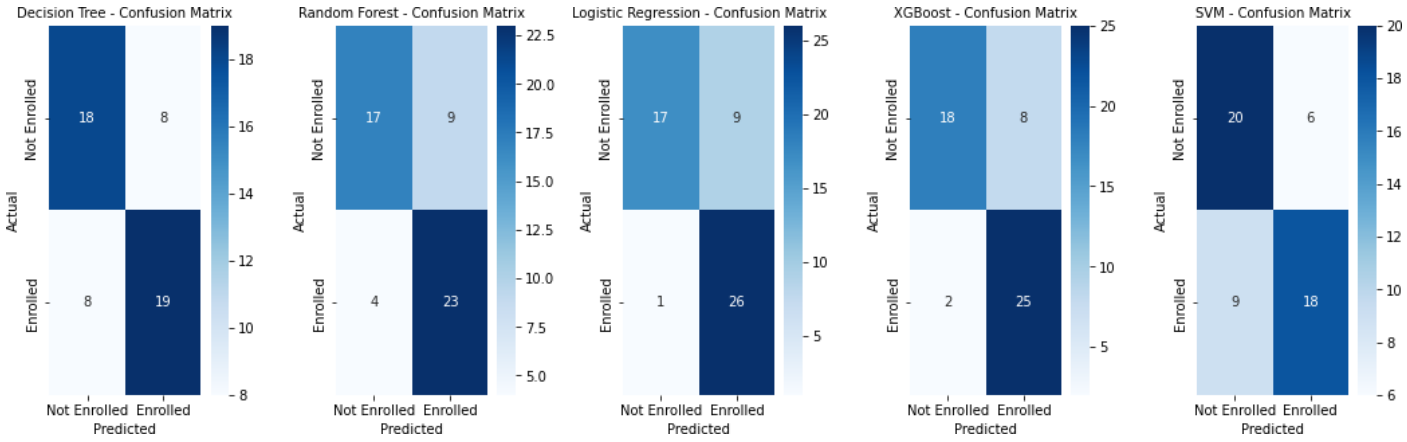

Fig A5. Confusion matrix of models on the hold-out set

7.3 Python Script for Machine Learning Models

```
# Import necessary libraries

import pandas as pd

import matplotlib.pyplot as plt

import statsmodels.api as sm

import seaborn as sns

from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```
from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

from sklearn.linear_model import LogisticRegression

from sklearn.svm import SVC

import xgboost as xgb

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
classification_report, confusion_matrix

import numpy as np

from imblearn.over_sampling import SMOTE

from sklearn.metrics import roc_curve, auc

from sklearn.model_selection import RandomizedSearchCV



# Load the Excel file

file_path = '/Users/Lenovo/Downloads/Enrolment Data Anonymised.xlsx'

xls = pd.ExcelFile(file_path)


# Load the data from the first sheet

df = pd.read_excel(xls, sheet_name='Sheet1')


# 1. Checking for missing values

missing_values = df.isnull().sum()

print("Missing Values:\n", missing_values)
```

```
# 2. Statistics of the data
statistics = df.describe(include='all')
print("\nStatistics:\n", statistics)

# 3. Data types and structure
df.info()

# 4. First few rows of the data
print("\nFirst Few Rows of the Data:\n", df.head())

# Handle 'Distance' column: Remove ' km' suffix and convert the Distance column to numeric
df['Distance'] = df['Distance'].str.replace(' km', '').astype(float)
df['Distance'].fillna(df['Distance'].mean(), inplace=True) # Impute missing values with the mean
print("\nDistance Column After Cleaning:\n", df['Distance'].head())

# Feature A: Create 'A_Numeric' and 'A_Missing' and drop original 'A' column
df['A_Numeric'] = df['A'].apply(lambda x: 1 if pd.notnull(x) else 0)
df['A_Missing'] = df['A'].apply(lambda x: 1 if pd.isnull(x) else 0)
df = df.drop(columns=['A']) # Drop the original 'A' column

# Feature B: Impute missing values and handle 'Prefer not to say'
df['B'].fillna(df['B'].mode()[0], inplace=True) # Impute missing values in 'B' with mode
df['B'] = df['B'].replace('Prefer not to say', df['B'].mode()[0])

# Features C1-C14: Replace checkmarks ('✓') with 1 and all other values with 0
```

```
columns_to_encode = ['C1', 'C2', 'C3', 'C4', 'C5', 'C6', 'C7', 'C8', 'C9', 'C10', 'C11', 'C12', 'C13', 'C14']

df[columns_to_encode] = df[columns_to_encode].applymap(lambda x: 1 if x == '✓' else 0)

print("\nEncoded C1-C14 Columns:\n", df[columns_to_encode].head())


# Additional Subjects: Create binary columns 'Yes', 'No', and 'Missing'

df['Subject_Yes'] = df['Additional Subjects'].apply(lambda x: 1 if pd.notnull(x) and x != 'No' else 0)

df['Subject_No'] = df['Additional Subjects'].apply(lambda x: 1 if x == 'No' else 0)

df['Subject_Missing'] = df['Additional Subjects'].apply(lambda x: 1 if pd.isnull(x) else 0)


# Drop the original 'Additional Subjects' column after creating binary columns

df = df.drop(columns=['Additional Subjects'])

print("\nAdditional Subjects Binary Columns Created and Original Dropped:\n", df[['Subject_Yes', 'Subject_No', 'Subject_Missing']].head())


# Convert 'Enroled?' column to binary: 1 for '✓', 0 for others

df['Enroled?'] = df['Enroled?'].apply(lambda x: 1 if x == '✓' else 0)


# Displaying summary statistics after cleaning

summary_stats = df.describe(include='all')

print("\nSummary Statistics After Cleaning:\n", summary_stats)


# Correlation metic and VIF score to check for multicollinearity


# Step 1: Compute the correlation matrix (without the target variable 'Enroled?')

correlation_matrix = df.drop(columns=['Enroled?']).corr()
```

```
# Step 2: Display the correlation matrix
print("\nCorrelation  Matrix:\n", correlation_matrix)

# Step 3: Export the correlation matrix to an Excel file
output_file_path = 'Correlation_Matrix.xlsx'
correlation_matrix.to_excel(output_file_path)

# Step 3: Plot the correlation  matrix using a heatmap
plt.figure(figsize=(12,  8))
sns.heatmap(correlation_matrix,  annot=True,  cmap='coolwarm',  fmt=".2f",  linewidths=0.5)
plt.title("Correlation Matrix Heatmap")
plt.show()

# Step  4: Prepare  the  feature  set (exclude the  target variable  'Enroled?')  for  VIF calculation
X_VIF = df.drop(columns=['Enroled?'])

# Step 5: Add a constant to the model for VIF calculation
X_with_constant = sm.add_constant(X_VIF)

# Step 6: Calculate VIF for each feature
vif_data = pd.DataFrame()
vif_data['Feature']  = X_VIF.columns
vif_data['VIF']  = [variance_inflation_factor(X_with_constant.values,  i + 1) for i in
range(len(X_VIF.columns))]
```

```
# Step 7: Display the VIF values

print("\nVariance Inflation Factor (VIF) for Each Feature:\n", vif_data)


# Drop one of the perfectly multicollinear features

df = df.drop(columns=['Subject_No', 'A_Missing'])# Drop one of the perfectly multicollinear features


# check the column names to ensure 'Subject_No' and 'A_Missing' does not exist

print(df.columns)



# ---------------------------------------

# Data Visualization (EDA)

# ---------------------------------------


# 1. Histogram for 'Distance' on training data

plt.figure(figsize=(8, 5))

plt.hist(df['Distance'], bins=10, edgecolor='black')

plt.title('Distribution of Distance of Train Data')

plt.xlabel('Distance (km)')

plt.ylabel('Frequency')

plt.show()


# 2. Pairplot for correlations between features on training data

sns.pairplot(df, vars=['Distance', 'A_Numeric', 'C1', 'C5', 'Enroled?'], hue='Enroled?', palette="coolwarm")

plt.show()
```

```
# ---------------------------------------
# Prepare the features (X) and target (y)
X = df.drop(columns=['Enroled?']) # Features
y = df['Enroled?'] # Target variable

# Check class distribution before SMOTE
print("Class distribution before SMOTE:")
print(y.value_counts())

# # Apply SMOTE to balance the classes
smote = SMOTE(random_state=42)
X_smote, y_smote = smote.fit_resample(X, y)

# # Check class distribution after SMOTE
print("Class distribution after SMOTE:")
print(y_smote.value_counts())

# Step 1: Split the data into training and holdout sets (80% training, 20% holdout)
X_train, X_holdout, y_train, y_holdout = train_test_split(X, y, test_size=0.2, random_state=42)

# 1. Train a Decision Tree Classifier
model_dt = DecisionTreeClassifier(random_state=42, class_weight='balanced')
model_dt.fit(X_train, y_train)
print("Decision Tree model trained.")
```

```
# 2. Train a Random Forest Classifier
model_rf = RandomForestClassifier(random_state=42)
model_rf.fit(X_train, y_train)
print("Random Forest model trained.")

# 3. Train a Logistic Regression Model
model_lr = LogisticRegression(random_state=42, max_iter=1000, class_weight='balanced')
model_lr.fit(X_train, y_train)
print("Logistic Regression model trained.")

# 4. Train a Gradient Boosting Classifier (XGBoost)
model_xgb = xgb.XGBClassifier(random_state=42, eval_metric='mlogloss')
model_xgb.fit(X_train, y_train)
print("XGBoost model trained.")

# 5. Train a Support Vector Machine (SVM)
model_svm = SVC(probability=True, random_state=42)
model_svm.fit(X_train, y_train)
print("SVM model trained.")

# ------------------------------------
# Evaluate Models on the Holdout Set
# ------------------------------------

# List of models to evaluate
models = {
```

```
    'Decision Tree': model_dt,
    'Random Forest': model_rf,
    'Logistic Regression': model_lr,
    'XGBoost': model_xgb,
    'SVM': model_svm
}

# Iterate through models and evaluate each one on the holdout set
for model_name, model in models.items():
    # Predict on the holdout set
    y_holdout_pred = model.predict(X_holdout)

    # Calculate Accuracy
    accuracy = accuracy_score(y_holdout, y_holdout_pred)

    # Calculate Precision, Recall, and F1-Score (for binary classification)
    precision = precision_score(y_holdout, y_holdout_pred)
    recall = recall_score(y_holdout, y_holdout_pred)
    f1 = f1_score(y_holdout, y_holdout_pred)

    # Print performance metrics
    print(f"--- {model_name} Performance on Holdout Set ---")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1-Score: {f1:.4f}")
```

```
    # Confusion Matrix
    conf_matrix = confusion_matrix(y_holdout, y_holdout_pred)

    # Plot confusion matrix
    plt.figure(figsize=(6, 4))
    sns.heatmap(conf_matrix, annot=True, fmt='g', cmap='Blues', xticklabels=['Not Enroled', 'Enroled'], yticklabels=['Not Enroled', 'Enroled'])
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title(f'{model_name} - Confusion Matrix (Holdout Set)')
    plt.show()

# Create subplots with the number of rows and columns corresponding to the number of models
fig, axes = plt.subplots(1, len(models), figsize=(15, 5))  # Adjust figsize as needed

# Iterate through models and plot confusion matrices in each subplot
for ax, (model_name, model) in zip(axes, models.items()):
    # Predict on the holdout set
    y_holdout_pred = model.predict(X_holdout)

    # Generate confusion matrix
    conf_matrix = confusion_matrix(y_holdout, y_holdout_pred)

    # Plot confusion matrix on the current subplot
    sns.heatmap(conf_matrix, annot=True, fmt='g', cmap='Blues',
```

```
            xticklabels=['Not Enroled', 'Enroled'],
            yticklabels=['Not Enroled', 'Enroled'], ax=ax)

    # Set title and labels for the subplot
    ax.set_title(f'{model_name} - Confusion Matrix', fontsize=10)
    ax.set_xlabel('Predicted')
    ax.set_ylabel('Actual')

# Adjust layout for better spacing between plots
plt.tight_layout(pad=2.0)  # Use 'pad' to add space between subplots

plt.show()

# Adjust layout for better spacing between plots
plt.tight_layout()
plt.show()

# Generate ROC curves for all models
plt.figure(figsize=(10, 8))

# Iterate through models and calculate ROC curve and AUC
for model_name, model in models.items():
    # Predict probabilities for the positive class (1) on the holdout set
    y_prob = model.predict_proba(X_holdout)[:, 1]

    # Calculate the ROC curve
```

```
    fpr, tpr, _ = roc_curve(y_holdout, y_prob)

    # Calculate the AUC (Area Under the Curve)
    roc_auc = auc(fpr, tpr)

    # Plot the ROC curve
    plt.plot(fpr, tpr, label=f'{model_name} (AUC = {roc_auc:.2f})')

# Plot diagonal line for random classifier (chance level)
plt.plot([0, 1], [0, 1], 'k--', lw=2)

# Set plot labels and title
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves for Various Models (Holdout Set)')
plt.legend(loc='lower right')
plt.show()

# Load the test data
test_file_path = '/Users/Lenovo/Downloads/Enrolment Data anonymised New.xlsx'
test_df = pd.read_excel(test_file_path)

# Define a function for preprocessing
def preprocess_data(df):
```

```
# 1. Handle 'Distance' column
df['Distance'] = df['Distance'].str.replace(' km', '').astype(float)
df['Distance'].fillna(df['Distance'].mean(), inplace=True)

# 2. Feature A: Create 'A_Numeric' and 'A_Missing', then drop original 'A' column
df['A_Numeric'] = df['A'].apply(lambda x: 1 if pd.notnull(x) else 0)
df['A_Missing'] = df['A'].apply(lambda x: 1 if pd.isnull(x) else 0)
df = df.drop(columns=['A'], errors='ignore')  # Drop the original 'A' column

# 3. Feature B: Impute missing values and handle 'Prefer not to say'
df['B'].fillna(df['B'].mode()[0], inplace=True)
df['B'] = df['B'].replace('Prefer not to say', df['B'].mode()[0])

# 4. Encode C1-C14 columns (replace '✓' with 1, others with 0)
columns_to_encode = ['C1', 'C2', 'C3', 'C4', 'C5', 'C6', 'C7', 'C8', 'C9', 'C10', 'C11', 'C12', 'C13',
'C14']
df[columns_to_encode] = df[columns_to_encode].applymap(lambda x: 1 if x == '✓' else 0)

# 5. Additional Subjects: Create 'Yes', 'No', and 'Missing' columns, then drop original
column
df['Subject_Yes'] = df['Additional Subjects'].apply(lambda x: 1 if pd.notnull(x) and x != 'No'
else 0)
df['Subject_Missing'] = df['Additional Subjects'].apply(lambda x: 1 if pd.isnull(x) else 0)
df = df.drop(columns=['Additional Subjects'], errors='ignore')

# 6. Convert 'Enroled?' column: '✓' to 1, NaN or missing to 0
df['Enroled?'] = df['Enroled?'].apply(lambda x: 1 if x == '✓' else 0)
```

```
    return df

# Apply the preprocessing to the test data
df_test_processed = preprocess_data(test_df)

# Verify the processed test data
print(df_test_processed.head())

# Generate summary statistics for the test data
summary_stats_test = df_test_processed.describe()

# Display the summary statistics
print("\nSummary Statistics for the Processed Test Data:\n", summary_stats_test)

# Visulisation

# 1. Histogram for 'Distance' for the test data
plt.figure(figsize=(8, 6))
plt.hist(df_test_processed['Distance'], bins=20, edgecolor='black')
plt.title('Distribution of Distance of Test Data')
plt.xlabel('Distance (km)')
plt.ylabel('Frequency')
plt.show()

# Calculate Q1 (25th percentile) and Q3 (75th percentile)
```

```
Q1 = df_test_processed['Distance'].quantile(0.25)

Q3 = df_test_processed['Distance'].quantile(0.75)


# Calculate the Interquartile Range (IQR)

IQR = Q3 - Q1


# Define the acceptable range for data (1.5 times the IQR is a common rule)

lower_bound = Q1 - 1.5 * IQR

upper_bound = Q3 + 1.5 * IQR


# Remove outliers from the Distance column

df_test_processed_no_outliers = df_test_processed[(df_test_processed['Distance'] >=
lower_bound) &

                                (df_test_processed['Distance'] <= upper_bound)]


# Verify the changes

print(df_test_processed_no_outliers['Distance'].describe())


# Re-plot the histogram for the 'Distance' column after removing the outliers

plt.figure(figsize=(8, 6))

plt.hist(df_test_processed_no_outliers['Distance'], bins=20, edgecolor='black')

plt.title('Distribution of Distance of Test Data (After Removing Outliers)')

plt.xlabel('Distance (km)')

plt.ylabel('Frequency')

plt.show()
```

```
# Create a combined histogram for both test and train data distance distributions

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


# Plot histogram for the train data (using df)

plt.hist(df['Distance'], bins=20, alpha=0.5, label='Train Data', edgecolor='black')


# Plot histogram for the test data (after removing outliers)

plt.hist(df_test_processed_no_outliers['Distance'], bins=20, alpha=0.5, label='Test Data',
edgecolor='black')


# Set the same scale for x-axis and y-axis

plt.xlim(0, max(df['Distance'].max(), df_test_processed_no_outliers['Distance'].max()))

plt.ylim(0, max(df['Distance'].value_counts().max(),
df_test_processed_no_outliers['Distance'].value_counts().max()))


# Set labels and title

plt.title('Distribution of Distance (Train and Test Data)')

plt.xlabel('Distance (km)')

plt.ylabel('Frequency')


# Add a legend

plt.legend()


# Show the plot

plt.show()
```

```
# Step 1: Compute the correlation matrix (without the target variable 'Enroled?')
correlation_matrix_test = df_test_processed.drop(columns=['Enroled?']).corr()

# Step 2: Display the correlation matrix
print("\nCorrelation Matrix for Test Data:\n", correlation_matrix_test)

# Step 3: Plot the correlation matrix using a heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix_test, annot=True, cmap='coolwarm', fmt=".2f",
linewidths=0.5)
plt.title("Correlation Matrix Heatmap of the Test Data")
plt.show()

# Step 4: Prepare the feature set (exclude the target variable 'Enroled?') for VIF calculation
X_test = df_test_processed.drop(columns=['Enroled?'])

# Step 5: Add a constant to the model for VIF calculation
X_test_with_constant = sm.add_constant(X_test)

# Step 6: Calculate VIF for each feature
vif_data_test = pd.DataFrame()
vif_data_test['Feature'] = X_test.columns
vif_data_test['VIF'] = [variance_inflation_factor(X_test_with_constant.values, i + 1) for i in
range(len(X_test.columns))]

# Step 7: Display the VIF values
print("\nVariance Inflation Factor (VIF) for Each Feature in Test Data:\n", vif_data_test)
```

```
# Drop redundant features with high multicollinearity
df_test_processed = df_test_processed.drop(columns=['A_Missing'])

# Testing the model on the unseen data-01

# Ensure the test data has the same features and order as the training data
X_test = df_test_processed.drop(columns=['Enroled?'])

# Get the feature names used during training
trained_feature_names = X_train.columns  # X_train is your feature set from training

# Reindex the test data to ensure the features match those used in training
X_test_aligned = X_test.reindex(columns=trained_feature_names, fill_value=0)

# Create a DataFrame for storing prediction probabilities
pred_probs = pd.DataFrame()

# Get prediction probabilities for each model
pred_probs['Decision_Tree_Prob'] = model_dt.predict_proba(X_test_aligned)[:, 1]
pred_probs['Random_Forest_Prob'] = model_rf.predict_proba(X_test_aligned)[:, 1]
pred_probs['Logistic_Regression_Prob'] = model_lr.predict_proba(X_test_aligned)[:, 1]
pred_probs['XGBoost_Prob'] = model_xgb.predict_proba(X_test_aligned)[:, 1]
pred_probs['SVM_Prob'] = model_svm.predict_proba(X_test_aligned)[:, 1]

# Add the actual 'Enroled?' values to the DataFrame
```

```
pred_probs['Actual_Enroled'] = df_test_processed['Enroled?']

# Save the prediction probabilities to an Excel file
output_file = '/Users/Lenovo/test_predictions.xlsx'
pred_probs.to_excel(output_file, index=False)

print(f"Prediction probabilities saved to {output_file}")

# Generate confusion matrices for each model
models = {
    'Decision Tree': model_dt,
    'Random Forest': model_rf,
    'Logistic Regression': model_lr,
    'XGBoost': model_xgb,
    'SVM': model_svm
}

for model_name, model in models.items():
    # Predict classes on the test data
    y_pred = model.predict(X_test_aligned)

    # Confusion Matrix
    conf_matrix = confusion_matrix(df_test_processed['Enroled?'], y_pred)

    # Calculate performance metrics
    accuracy = accuracy_score(df_test_processed['Enroled?'], y_pred)
```

```
    precision = precision_score(df_test_processed['Enroled?'], y_pred)
    recall = recall_score(df_test_processed['Enroled?'], y_pred)
    f1 = f1_score(df_test_processed['Enroled?'], y_pred)

    # Print performance metrics
    print(f"\n--- {model_name} Performance on Test Data ---")
    print(f'Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1-Score: {f1:.4f}")

    # Plot confusion matrix
    plt.figure(figsize=(6, 4))
    sns.heatmap(conf_matrix, annot=True, fmt='g', cmap='Blues', xticklabels=['Not Enroled',
'Enroled'], yticklabels=['Not Enroled', 'Enroled'])
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title(f'{model_name} - Confusion Matrix (Test Data)')
    plt.show()

# Create a figure with subplots for each model's confusion matrix
fig, axes = plt.subplots(1, len(models), figsize=(20, 5))

# Iterate through the models and plot their confusion matrix
for ax, (model_name, model) in zip(axes, models.items()):
    # Predict classes on the test data
```

```
    y_pred = model.predict(X_test_aligned)

    # Generate the confusion matrix
    conf_matrix = confusion_matrix(df_test_processed['Enroled?'], y_pred)

    # Plot the confusion matrix on the corresponding subplot
    sns.heatmap(conf_matrix, annot=True, fmt='g', cmap='Blues', ax=ax,
                xticklabels=['Not Enroled', 'Enroled'], yticklabels=['Not Enroled', 'Enroled'])

    # Set title and labels for the subplot
    ax.set_title(f'{model_name}')
    ax.set_xlabel('Predicted')
    ax.set_ylabel('Actual')

# Adjust layout to prevent overlapping
plt.tight_layout()
plt.show()

# Create a figure for the ROC curves
plt.figure(figsize=(10, 8))

# Iterate through models to compute ROC curve and AUC
for model_name, model in models.items():
    # Predict probabilities for the positive class (1) on the test data
    y_prob = model.predict_proba(X_test_aligned)[:, 1]
```

```
    # Calculate the ROC curve
    fpr, tpr, _ = roc_curve(df_test_processed['Enroled?'], y_prob)

    # Calculate the AUC (Area Under the Curve)
    roc_auc = auc(fpr, tpr)

    # Plot the ROC curve
    plt.plot(fpr, tpr, label=f'{model_name} (AUC = {roc_auc:.2f})')

# Plot diagonal line for random classifier
plt.plot([0, 1], [0, 1], 'k--', lw=2)

# Set plot labels and title
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves for Various Models (Test Data) before hyperparameter tuning')
plt.legend(loc='lower right')
plt.show()



# Define hyperparameter grids for each model

# 1. Decision Tree
```

```
param_grid_dt = {
    'max_depth': [10, 20, 30, None], # Max depth of the tree
    'min_samples_split': [2, 5, 10], # Minimum samples required to split a node
    'min_samples_leaf': [1, 2, 4] # Minimum samples required at a leaf node
}

# 2. Random Forest
param_grid_rf = {
    'n_estimators': [100, 200, 500, 1000], # Number of trees
    'max_depth': [10, 20, 30, None], # Max depth of trees
    'min_samples_split': [2, 5, 10], # Minimum samples to split a node
    'min_samples_leaf': [1, 2, 4], # Minimum samples at a leaf node
    'max_features': ['auto', 'sqrt', 'log2'] # Number of features to consider for the best split
}

# 3. Logistic Regression
param_grid_lr = {
    'C': [0.01, 0.1, 1, 10, 100], # Regularization strength
    'penalty': ['l1', 'l2'], # Regularization method
    'solver': ['liblinear', 'saga'] # Solver to use
}

# 4. SVM
param_grid_svm = {
    'C': [0.01, 0.1, 1, 10, 100], # Regularization parameter
    'kernel': ['linear', 'rbf', 'poly'], # Kernel type
```

```
    'gamma': ['scale', 'auto']  # Kernel coefficient
}

# 5. XGBoost
param_grid_xgb = {
    'learning_rate': [0.01, 0.1, 0.3],  # Learning rate
    'max_depth': [3, 6, 9],  # Maximum depth of the trees
    'n_estimators': [100, 200, 500],  # Number of boosting rounds
    'subsample': [0.6, 0.8, 1.0]  # Subsample ratio of training instances
}

# Hyperparameter tuning with RandomizedSearchCV

# 1. Decision Tree
dt_random = RandomizedSearchCV(
    estimator=DecisionTreeClassifier(random_state=42),
    param_distributions=param_grid_dt,
    n_iter=50,
    cv=5,
    verbose=2,
    random_state=42,
    n_jobs=-1
)
dt_random.fit(X_train, y_train)
best_dt = dt_random.best_estimator_
```

```
# 2. Random Forest
rf_random  = RandomizedSearchCV(
   estimator=RandomForestClassifier(random_state=42),
   param_distributions=param_grid_rf,
   n_iter=50,
   cv=5,
   verbose=2,
   random_state=42,
   n_jobs=-1
)
rf_random.fit(X_train, y_train)
best_rf = rf_random.best_estimator_

# 3. Logistic Regression
lr_random = RandomizedSearchCV(
   estimator=LogisticRegression(random_state=42,  class_weight='balanced',
max_iter=1000),
   param_distributions=param_grid_lr,
   n_iter=50,
   cv=5,
   verbose=2,
   random_state=42,
   n_jobs=-1
)
lr_random.fit(X_train, y_train)
best_lr = lr_random.best_estimator_
```

```
# 4. SVM
svm_random = RandomizedSearchCV(
    estimator=SVC(random_state=42,  probability=True),
    param_distributions=param_grid_svm,
    n_iter=50,
    cv=5,
    verbose=2,
    random_state=42,
    n_jobs=-1
)
svm_random.fit(X_train,  y_train)
best_svm = svm_random.best_estimator_

# 5. XGBoost
xgb_random = RandomizedSearchCV(
    estimator=xgb.XGBClassifier(random_state=42,   use_label_encoder=False,
eval_metric='mlogloss'),
    param_distributions=param_grid_xgb,
    n_iter=50,
    cv=5,
    verbose=2,
    random_state=42,
    n_jobs=-1
)
xgb_random.fit(X_train,  y_train)
```

```
best_xgb = xgb_random.best_estimator_

# Evaluate all the best models on the holdout set
models = {
    'Decision Tree': best_dt,
    'Random Forest': best_rf,
    'Logistic Regression': best_lr,
    'SVM': best_svm,
    'XGBoost':  best_xgb
}

    # Evaluate on holdout set
for model_name, model in models.items():
    # Predict on the holdout set
    y_holdout_pred = model.predict(X_holdout)

    # Calculate performance metrics
    accuracy = accuracy_score(y_holdout, y_holdout_pred)
    precision = precision_score(y_holdout, y_holdout_pred)
    recall = recall_score(y_holdout, y_holdout_pred)
    f1 = f1_score(y_holdout, y_holdout_pred)

    # Print performance metrics
    print(f"\n--- {model_name} Performance on Holdout Set ---")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
```

```
    print(f"Recall: {recall:.4f}")
    print(f"F1-Score: {f1:.4f}")

    # Confusion Matrix
    conf_matrix = confusion_matrix(y_holdout, y_holdout_pred)

    # Plot confusion matrix
    plt.figure(figsize=(6, 4))
    sns.heatmap(conf_matrix, annot=True, fmt='g', cmap='Blues', xticklabels=['Not Enroled', 'Enroled'], yticklabels=['Not Enroled', 'Enroled'])
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title(f'{model_name} - Confusion Matrix (Holdout Set)')
    plt.show()

# Create subplots to show confusion matrices side by side for all models
fig, axes = plt.subplots(1, len(models), figsize=(20, 5))

# Iterate through models and plot their confusion matrix in each subplot
for ax, (model_name, model) in zip(axes, models.items()):
    # Predict classes on the holdout set
    y_holdout_pred = model.predict(X_holdout)

    # Generate the confusion matrix
    conf_matrix = confusion_matrix(y_holdout, y_holdout_pred)
```

```
    # Plot the confusion matrix on the current subplot
    sns.heatmap(conf_matrix, annot=True, fmt='g', cmap='Blues',
                xticklabels=['Not Enroled', 'Enroled'],
                yticklabels=['Not Enroled', 'Enroled'], ax=ax)

    # Set title and labels for the subplot
    ax.set_title(f'{model_name}')
    ax.set_xlabel('Predicted')
    ax.set_ylabel('Actual')

# Adjust layout for better spacing
plt.tight_layout()
plt.show()



# Assuming the test data is already preprocessed and stored in df_test_processed
# X_test: Features from the test data
X_test = df_test_processed.drop(columns=['Enroled?'])  # Features from the test data

# Use the hypertuned models to predict on the test data
# y_test_actual: Actual labels for the test data
y_test_actual = df_test_processed['Enroled?']

# Create a DataFrame to store the prediction probabilities and actual values
pred_probs = pd.DataFrame()
```

```
# Get prediction probabilities for each hypertuned model
pred_probs['Decision_Tree_Prob'] = best_dt.predict_proba(X_test)[:, 1]
pred_probs['Random_Forest_Prob'] = best_rf.predict_proba(X_test)[:, 1]
pred_probs['Logistic_Regression_Prob'] = best_lr.predict_proba(X_test)[:, 1]
pred_probs['XGBoost_Prob'] = best_xgb.predict_proba(X_test)[:, 1]
pred_probs['SVM_Prob'] = best_svm.predict_proba(X_test)[:, 1]

# Add the actual 'Enroled?' values to the DataFrame
pred_probs['Actual_Enroled'] = y_test_actual

# Save the prediction probabilities to an Excel file for further analysis
output_file = '/Users/Lenovo/test_predictions_after_hyperparameter_tuning.xlsx'
pred_probs.to_excel(output_file, index=False)

print(f"Prediction probabilities saved to {output_file}")

# Ensure the test data has the same features and order as the training data
# Use the feature names from the training data (X_train.columns)
X_test_aligned = X_test.reindex(columns=X_train.columns, fill_value=0) # Align columns and fill missing ones with 0

# Now, use X_test_aligned in the model predictions
for model_name, model in models.items():
    # Predict classes on the test data
    y_pred = model.predict(X_test_aligned) # Use X_test_aligned instead of X_test
```

```
    # Confusion Matrix
    conf_matrix = confusion_matrix(y_test_actual, y_pred)

    # Calculate performance metrics
    accuracy = accuracy_score(y_test_actual, y_pred)
    precision = precision_score(y_test_actual, y_pred)
    recall = recall_score(y_test_actual, y_pred)
    f1 = f1_score(y_test_actual, y_pred)

    # Print performance metrics
    print(f"\n--- {model_name} Performance on Test Data ---")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1-Score: {f1:.4f}")

    # Plot confusion matrix
    plt.figure(figsize=(6, 4))
    sns.heatmap(conf_matrix, annot=True, fmt='g', cmap='Blues', xticklabels=['Not Enroled',
'Enroled'], yticklabels=['Not Enroled', 'Enroled'])
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title(f'{model_name} - Confusion Matrix (Test Data)')
    plt.show()

# Create subplots to show confusion matrices side by side
```

```
fig, axes = plt.subplots(1, len(models), figsize=(20, 5))

# Iterate through models and plot their confusion matrix in each subplot
for ax, (model_name, model) in zip(axes, models.items()):
    # Predict classes on the test data
    y_pred = model.predict(X_test_aligned)

    # Generate confusion matrix
    conf_matrix = confusion_matrix(y_test_actual, y_pred)

    # Plot the confusion matrix on the current subplot
    sns.heatmap(conf_matrix, annot=True, fmt='g', cmap='Blues',
                xticklabels=['Not Enroled', 'Enroled'],
                yticklabels=['Not Enroled', 'Enroled'], ax=ax)

    # Set title and labels for each subplot
    ax.set_title(f'{model_name} - Confusion Matrix')
    ax.set_xlabel('Predicted')
    ax.set_ylabel('Actual')

# Adjust the layout to prevent overlap
plt.tight_layout()
plt.show()

# Generate ROC curves for all hypertuned models
plt.figure(figsize=(10, 8))
```

```
# Dictionary  to store AUC values for each model
auc_values = {}

# Iterate through models and calculate ROC curve and AUC
for model_name, model in models.items():
    # Predict probabilities for the positive class (1) on the test data
    y_prob = model.predict_proba(X_test_aligned)[:,  1]

    # Calculate the ROC curve
    fpr, tpr, _ = roc_curve(y_test_actual, y_prob)

    # Calculate the AUC (Area Under the Curve)
    roc_auc = auc(fpr, tpr)
    auc_values[model_name] = roc_auc

    # Plot the ROC curve
    plt.plot(fpr, tpr, label=f'{model_name}  (AUC = {roc_auc:.2f})')

# Plot diagonal line for random classifier
plt.plot([0, 1], [0, 1], 'k--', lw=2)

# Set plot labels and title
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')

plt.title('ROC Curves for Hypertuned Models (Test Data)')

plt.legend(loc='lower right')

plt.show()


from sklearn.metrics import precision_recall_curve, roc_curve


# List of models to evaluate

models = {

   'Decision Tree': best_dt, # Decision Tree doesn't output probabilities

   'Random Forest': best_rf,

   'Logistic Regression': best_lr,

   'XGBoost': best_xgb,

   'SVM': best_svm

}


# Dictionary to store model probabilities

model_probs = {}


# Get predicted probabilities for models that support them

model_probs['Decision Tree'] = models['Decision Tree'].predict_proba(X_test_aligned)[:, 1]

model_probs['Random Forest'] = models['Random Forest'].predict_proba(X_test_aligned)[:,
1]

model_probs['Logistic Regression'] = models['Logistic
Regression'].predict_proba(X_test_aligned)[:, 1]

model_probs['XGBoost'] = models['XGBoost'].predict_proba(X_test_aligned)[:, 1]

model_probs['SVM'] = models['SVM'].predict_proba(X_test_aligned)[:, 1]
```

```
# You can change the thresholds as needed
thresholds = [0.3, 0.4, 0.5, 0.6, 0.7]

# Evaluate for each model and threshold
for model_name, y_prob in model_probs.items():
    print(f"\n--- {model_name} Performance at Different Thresholds ---")
    for threshold in thresholds:
        # Apply threshold
        y_pred_threshold = (y_prob >= threshold).astype(int)

        # Calculate metrics
        accuracy = accuracy_score(y_test_actual, y_pred_threshold)
        precision = precision_score(y_test_actual, y_pred_threshold)
        recall = recall_score(y_test_actual, y_pred_threshold)
        f1 = f1_score(y_test_actual, y_pred_threshold)

        # Print results for each threshold
        print(f"\nThreshold: {threshold}")
        print(f"Accuracy: {accuracy:.4f}")
        print(f"Precision: {precision:.4f}")
        print(f"Recall: {recall:.4f}")
        print(f"F1-Score: {f1:.4f}")

# Plot ROC curve to find the optimal threshold
for model_name, y_prob in model_probs.items():
```

```
    fpr, tpr, roc_thresholds = roc_curve(y_test_actual, y_prob)

    # Plot the ROC curve
    plt.figure(figsize=(8, 6))
    plt.plot(fpr, tpr, label=model_name)
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curve for {model_name}')
    plt.legend(loc="best")
    plt.show()

    # Find the optimal threshold based on ROC curve (You can change this method if needed)
    optimal_idx = np.argmax(tpr - fpr)
    optimal_threshold = roc_thresholds[optimal_idx]
    print(f'Optimal threshold for {model_name}: {optimal_threshold:.4f}')


# Plot all ROC curves in a single plot with threshold 0.7 highlighted
plt.figure(figsize=(10, 8))

for model_name, y_prob in model_probs.items():
    # Generate ROC curve
    fpr, tpr, roc_thresholds = roc_curve(y_test_actual, y_prob)

    # Plot the ROC curve
    plt.plot(fpr, tpr, lw=2, label=f'{model_name} ROC Curve (AUC = {auc(fpr, tpr):.2f})')
```

```
    # Highlight the point for threshold 0.7 on the ROC curve
    if 0.7 in roc_thresholds:
        threshold_idx = np.where(roc_thresholds == 0.7)[0][0]
        plt.scatter(fpr[threshold_idx], tpr[threshold_idx], color='red', s=100,
label=f'{model_name} Threshold = 0.7')
    else:
        # Find the closest threshold if 0.7 isn't exactly present
        closest_idx = np.argmin(np.abs(roc_thresholds - 0.7))
        plt.scatter(fpr[closest_idx], tpr[closest_idx], color='red', s=100, label=f'{model_name}
Closest to Threshold 0.7')

# Plot diagonal line for random classifier
plt.plot([0, 1], [0, 1], 'k--', lw=2)

# Set plot labels and title
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves for All Models with Threshold 0.7 Highlighted')
plt.legend(loc='lower right')
plt.show()

# Step 1: Set thresholds for classification
enroled_threshold = 0.7
```

```
not_enroled_threshold = 0.25

# Step 2: Sort students based on their distance from 0.5 for undecided classification
def categorize_probabilities(probabilities, actual_labels, undecided_count=50):
    # Create a DataFrame to store results
    result_df = pd.DataFrame({
        'Probability': probabilities,
        'Actual_Enroled': actual_labels
    })

    # Step 3: Sort by distance from 0.5 (for undecided classification)
    result_df['Distance_From_0.5'] = abs(result_df['Probability'] - 0.5)

    # Step 4: First, assign all as "Undecided"
    result_df['Predicted_Category'] = 'Undecided'

    # Step 5: Categorize students as 'Enroled' or 'Not Enroled' based on thresholds
    result_df.loc[result_df['Probability'] >= enroled_threshold, 'Predicted_Category'] =
'Enroled'
    result_df.loc[result_df['Probability'] <= not_enroled_threshold, 'Predicted_Category'] =
'Not Enroled'

    # Step 6: Sort by distance from 0.5 and mark the 50 closest to 0.5 as 'Undecided'
    undecided_students = result_df[result_df['Predicted_Category'] == 'Undecided']
    undecided_students_sorted = undecided_students.sort_values(by='Distance_From_0.5')

    # Only keep the top 50 closest to 0.5 as undecided
```

```
    top_undecided_students = undecided_students_sorted.head(undecided_count).index

    # Step 7: Mark the remaining 'Undecided' students as either 'Enroled' or 'Not Enroled'
    result_df.loc[undecided_students_sorted.index.difference(top_undecided_students), 'Predicted_Category'] = (
        result_df.loc[undecided_students_sorted.index.difference(top_undecided_students), 'Probability']
        .apply(lambda x: 'Enroled' if x >= 0.5 else 'Not Enroled')
    )

    return result_df[['Probability', 'Predicted_Category', 'Actual_Enroled']]

# Step 8: Apply this function for each model's probabilities

# Ensure prediction probabilities for each model are correctly added
pred_probs = pd.DataFrame()

# Get prediction probabilities for each model
pred_probs['Decision_Tree_Prob'] = best_dt.predict_proba(X_test_aligned)[:, 1]
pred_probs['Random_Forest_Prob'] = best_rf.predict_proba(X_test_aligned)[:, 1]
pred_probs['Logistic_Regression_Prob'] = best_lr.predict_proba(X_test_aligned)[:, 1]
pred_probs['XGBoost_Prob'] = best_xgb.predict_proba(X_test_aligned)[:, 1]
pred_probs['SVM_Prob'] = best_svm.predict_proba(X_test_aligned)[:, 1]

# Add actual 'Enroled?' values for comparison
pred_probs['Actual_Enroled'] = y_test_actual
```

```
# Define the categorize_probabilities function

def categorize_probabilities(probabilities, actual_enroled, threshold=0.7,
undecided_count=50):

    categorized = pd.DataFrame({'Probability': probabilities, 'Actual_Enroled':
actual_enroled})


    # Set initial categories based on threshold

    categorized['Predicted_Category'] = categorized['Probability'].apply(

        lambda x: 'Enroled' if x >= threshold else ('Not Enroled' if x < 0.25 else 'Undecided')

    )


    # Sort the undecided students by distance from 0.5

    undecided_students = categorized[(categorized['Predicted_Category'] ==
'Undecided')].copy()

    undecided_students['Distance_From_0.5'] = abs(undecided_students['Probability'] - 0.5)


    # Sort undecided students and pick the closest to 0.5

    closest_to_undecided = undecided_students.nsmallest(undecided_count,
 'Distance_From_0.5')

    categorized.loc[closest_to_undecided.index, 'Predicted_Category'] = 'Undecided'


     return categorized


# Apply categorization for each model

for model_name in ['Decision_Tree_Prob', 'Random_Forest_Prob',
'Logistic_Regression_Prob', 'XGBoost_Prob', 'SVM_Prob']:

    print(f"\n--- Categorizing for {model_name} ---")

    categorized_results = categorize_probabilities(
```

```
    pred_probs[model_name],
    pred_probs['Actual_Enroled'],
    undecided_count=50  # Set the number of undecided students
  )

  # Save categorized results to a new column
  pred_probs[f'{model_name}_Category'] = categorized_results['Predicted_Category']

# Combine the original test data with the prediction probabilities and categories
full_output = pd.concat([df_test_processed.reset_index(drop=True), pred_probs], axis=1)

# Save the full output with test data, predictions, and categories to an Excel file
output_file_with_full_data =
'/Users/Lenovo/test_predictions_with_full_data_and_categories.xlsx'
full_output.to_excel(output_file_with_full_data, index=False)

print(f"Full test data with prediction probabilities and categories saved to
{output_file_with_full_data}")

# FInal model selected - Logistic regression. thershold was 0.7

# Step 1: Apply the 0.7 threshold to categorize as 'Enroled' or 'Not Enroled'
pred_probs['Logistic_Predicted_Labels'] = (pred_probs['Logistic_Regression_Prob'] >=
0.7).astype(int)

# Step 2: Calculate performance metrics based on the actual labels ('Actual_Enroled') and
predicted labels
```

```
accuracy = accuracy_score(pred_probs['Actual_Enroled'],
pred_probs['Logistic_Predicted_Labels'])

precision = precision_score(pred_probs['Actual_Enroled'],
pred_probs['Logistic_Predicted_Labels'])

recall = recall_score(pred_probs['Actual_Enroled'], pred_probs['Logistic_Predicted_Labels'])

f1 = f1_score(pred_probs['Actual_Enroled'], pred_probs['Logistic_Predicted_Labels'])


# Step 3: Print performance metrics

print("\n--- Logistic Regression Performance Metrics at 0.7 Threshold ---")

print(f"Accuracy: {accuracy:.4f}")

print(f"Precision: {precision:.4f}")

print(f"Recall: {recall:.4f}")

print(f"F1-Score: {f1:.4f}")


# Step 4: Generate the confusion matrix

conf_matrix = confusion_matrix(pred_probs['Actual_Enroled'],
pred_probs['Logistic_Predicted_Labels'])


# Step 5: Print the confusion matrix

print("\nConfusion Matrix for Logistic Regression (0.7 Threshold):")

print(conf_matrix)


# Optional: Plot the confusion matrix

plt.figure(figsize=(6, 4))

sns.heatmap(conf_matrix, annot=True, fmt='g', cmap='Blues', xticklabels=['Not Enroled',
'Enroled'], yticklabels=['Not Enroled', 'Enroled'])

plt.xlabel('Predicted')

plt.ylabel('Actual')
```

```
plt.title('Logistic Regression - Confusion Matrix (Test Data)')

plt.show()


# Step 1: Calculate the distance from 0.5 for each student in the logistic regression predictions

pred_probs['Logistic_Distance_from_0.5'] = abs(pred_probs['Logistic_Regression_Prob'] - 0.5)


# Step 2: Sort by distance from 0.5 to identify the "Undecided" students

pred_probs_sorted = pred_probs.sort_values(by='Logistic_Distance_from_0.5').reset_index(drop=True)


# Step 3: Categorize based on the threshold and closest 50 to 0.5 as "Undecided"

# Initialize the category as "Not Enroled" for all

pred_probs_sorted['Logistic_Category'] = 'Not Enroled'


# Step 4: Apply the thresholds

pred_probs_sorted.loc[pred_probs_sorted['Logistic_Regression_Prob'] >= 0.7, 'Logistic_Category'] = 'Enroled'

pred_probs_sorted.loc[pred_probs_sorted['Logistic_Regression_Prob'] <= 0.25, 'Logistic_Category'] = 'Not Enroled'


# Step 5: Set the closest 50 students to 0.5 as "Undecided"

pred_probs_sorted.loc[:49, 'Logistic_Category'] = 'Undecided'


# Step 6: Add this categorized result along with the prediction probabilities to the test data (before processing)

df_test_processed_with_preds = df_test_processed.copy()  # Keep the original test data structure intact
```

```
df_test_processed_with_preds['Logistic_Regression_Prob'] =
pred_probs_sorted['Logistic_Regression_Prob']

df_test_processed_with_preds['Logistic_Distance_from_0.5'] =
pred_probs_sorted['Logistic_Distance_from_0.5']

df_test_processed_with_preds['Logistic_Category'] = pred_probs_sorted['Logistic_Category']


# Step 7: Save the updated test data with logistic regression predictions and categories

output_file_with_logistic_categories =
'/Users/Lenovo/logistic_test_predictions_with_categories.xlsx'

df_test_processed_with_preds.to_excel(output_file_with_logistic_categories, index=False)


print(f"Logistic Regression prediction probabilities and categories saved to
{output_file_with_logistic_categories}")


output_file_with_logistic_categories =
'/Users/Lenovo/logistic_test_predictions_with_categories_v2.xlsx'

df_test_processed_with_preds.to_excel(output_file_with_logistic_categories, index=False)


print(f"Logistic Regression prediction probabilities and categories saved to
{output_file_with_logistic_categories}")


# Extra Graphs

# Histograms

plt.figure(figsize=(14, 6))


# Histogram for Training Data

plt.subplot(1, 2, 1)

plt.hist(df['Distance'], bins=10, edgecolor='black', color='blue', alpha=0.7)
```

```
plt.title('Distribution of Distance (Training Data)', fontsize=12) # Reduced title size
plt.xlabel('Distance (km)')
plt.ylabel('Frequency')

# Histogram for Test Data
plt.subplot(1, 2, 2)
plt.hist(df_test_processed['Distance'], bins=10, edgecolor='black', color='green', alpha=0.7)
plt.title('Distribution of Distance (Test Data)', fontsize=12)  # Reduced title size
plt.xlabel('Distance (km)')
plt.ylabel('Frequency')

# Show the side-by-side plots with adjusted title size
plt.tight_layout()
plt.show()
```