Team 13: Georgia Karen Lau, Liang Yuxin, Lim Zhen Yong, Tang Xiling, Ying Yunqian

1 Title

Predicting Student Dropout. In this project, we aim to evaluate multiple machine learning models to determine the most effective approach for predicting student's dropout. Using a data set containing various student attributes, we implement five different models: Decision Tree, XGBoost, Support Vector Machine (SVM), Neural Network and Random Forest with 5-fold cross-validation and choose the best model to predict the test data set.

2 Introduction

Predicting student dropout is a crucial challenge in education, as early identification of at-risk students enables timely interventions to improve retention rates and academic success. Singapore has seen a rise in out-of-school youth, with 17,000 aged 15–24 not in education, employment, or training in 2023, up from 2013 [Ang, 2024]. These youth face challenges in fulfilling their potential and risk social exclusion due to prolonged disengagement, as highlighted by Senior Parliamentary Secretary Eric Chua during the Youth Outreach Conference. Many factors, such as socioeconomic background, academic history, and engagement levels, can influence students' academic performance. By leveraging machine learning techniques, we aim to develop a predictive model that can assist educators in identifying students who are at risk of dropping out.

2.1 Data Description

This dataset is from the paper - Multi-Class Phased Prediction of Academic Performance and Dropout [Martins et al., 2023]. The data was collected for 4433 undergraduate students enrolled in the Polytechnic University of Portalegre between 2009 and 2017. It was compiled from multiple internal and external sources of the university, such as its academic service database, Portuguese General Directorate of Higher Education and Portugal National statistics portal (PORDATA). The data was used to identify at-risk students early so that timely interventions can be made. With 37 features, this data includes information related to students' academic, demographic, socio-economic and macroeconomics conditions. The detailed breakdown of the features can be seen in Appendix A.

3 Preliminary Analysis

3.1 EDA and Data Transformation Report for Categorical Variables

In the exploratory data analysis (EDA) phase of this project, we focused on visualising data, normalising numerical data and transforming categorical data into suitable format for model training. The dataset includes various demographic and academic attributes, many of which are categorical in nature. Proper handling of categorical variables is crucial to ensure the machine learning models can interpret the data effectively. Various data visualisations can be seen in Appendix B.

From the data visualisations we derived that:

- The proportion of non-dropout vs dropout is 67.88% and 32.12%, respectively.
- Most of the numerical features are close to normal. Some academic-related features like circular units (credited), circular units (without evaluation) have low variance, i.e. majority of the data are within narrow ranges. These features will be retained as we believe the minority values might provide useful information for our classification tasks.

- Some categorical features have many categories, such as occupation, qualification and course. Data transformation techniques are needed to transform these features for machine learning use cases.
- Some circular unit-related features show high correlation. Though correlated, we decided to retain them because these features represent different phases of academic performance that will impact students' outcomes significantly.

3.2 Data Loading and Inspection

The dataset was loaded from Google Drive using `pandas` library in Python. After loading the data, we performed an initial inspection of the dataset with `df.info()`, which provided an overview of the features and their data types. This helped us identify the categorical variables that needed to be transformed for further analysis and modeling.

3.3 Identification of Categorical Variables

The dataset contains several categorical variables representing student characteristics and academic information. These categorical variables are:

- Marital status
- Application mode
- Course
- Daytime/evening attendance
- Previous qualification
- Nationality
- Mother's qualification
- Father's qualification
- Mother's occupation
- Father's occupation
- Displaced
- Educational special needs
- Debtor
- Tuition fees up to date
- Gender
- Scholarship holder
- International

These variables, while useful for predicting student chances of dropout, are not in a format suitable for machine learning models, which require numerical inputs. Thus, a transformation process is required to convert these categorical variables into a format which fits for machine learning tasks.

3.4 Data Processing

We applied a grouping technique for categorical variables with more than four distinct categories. In cases where the number of categories exceeded four, the less frequent categories were grouped into an "others" group. This ensures that the model does not suffer from overfitting due to rare categories that have less contribution to the predictive power of the model. The function `process_categorical_variable` was applied to each categorical column in the dataset, iterating over the list of categorical variables. For each variable:

- The value counts of the variable were calculated.
- If the variable had more than four categories,the top four categories were retained, and the remaining categories were merged into an "others" category.
- This transformation reduces the dimensionality of the dataset and helps the model focus on the most important categories.

After processing categorical variables, we performed One-Hot Encoding to convert categorical variables into binary columns. This creates new binary columns for each possible category within the variable, where a 1 represents the presence of a particular category and a 0 represents its absence.

We used `pd.get_dummies()` to encode the categorical variables, with the `drop_first=True` argument to avoid multicollinearity by dropping the first category in each encoded variable (which acts as a reference category). This ensures that the model will have a sufficient number of features while avoiding redundant information.

For continuous data like "Previous qualification (grade)", "Admission grade", "Age at enrollment". "Unemployment rate", "Curricular units 1st sem (credited)", "Curricular units 1st sem (enrolled)", "Curricular units 1st sem (evaluations)", "Curricular units 1st sem (approved)", "Curricular units 1st sem (grade)", "Curricular units 1st sem (without evaluations)", "Curricular units 2nd sem (credited)", "Curricular units 2nd sem (enrolled)", "Curricular units 2nd sem (evaluations)", "Curricular units 2nd sem (approved)", "Curricular units 2nd sem (grade)", "Curricular units 2nd sem (without evaluations)", "Inflation rate", "GDP", and "Application order" which is a ordinal data, we applied minmax scaler to normalise the data to make it easier to compare and interpret features that originally had different ranges.

3.5 Transformation of the Target Variable

The target variable originally had three categories: **Graduate (**$\sim$ 50.83%**), Enrolled (**$\sim$ 17.05%), and **Dropout** ($\sim$ 32.12%). To simplify the classification task, we restructured it as a binary classification problem where "Graduate" and "Enrolled" were merged into a single "Non-dropout" category with value 0 and "Dropout" was encoded with value 1. This makes it easier to apply machine learning classification algorithms, as the target variable is now a binary outcome.

4 Full Results from Analysis

4.1 Training the Models

Before proceeding with our analysis, we applied Stratified K-fold Cross-Validation with k=5 for all models to ensure reliable and robust performance evaluation. Cross-validation is a crucial technique in machine learning that helps assess how well a model generalises to unseen data. Instead of relying on a single train-test split, which can lead to misleading results due to variance in data distribution, k-fold validation divides the dataset into k equally sized folds. The model is trained on k-1 folds and validated on the remaining fold, repeating this process k times with different validation sets. The results are then averaged for a more stable estimate of model performance. We specifically used stratified k-fold to ensure that each fold maintains the original class distribution. This is particularly important in imbalanced binary classification problems, such as predicting student dropout, where maintaining the class distribution is essential to avoid biased evaluation. To support this, we first split the dataset into 80% training and 20% testing sets using stratified splitting to maintain the target class proportion. We then applied k-fold cross-validation only on the training portion, while the test set was held out and used exclusively for final evaluation.

4.1.1 Baseline Model - Logistic Regression

We chose Logistic Regression as our baseline model due to its simplicity, speed, and interpretability, making it a standard choice for binary classification. It provides a solid reference point to assess whether more complex models, such as tree-based algorithms, actually deliver significant improvements. In educational contexts, logistic regression is often favoured in early stages of modeling due to its transparency and ability to explain relationships between variables and the outcome.

Logistic Regression is a generalised linear model that predicts the probability of a binary outcome. Instead of modeling the output directly, it models the log-odds of the probability, making it suitable for classification problems. It assumes a linear relationship between the independent variables and the log-odds of the dependent variable. It is particularly useful when interpretability and quick insights into variable importance are necessary. However, Logistic Regression model may fall short in capturing nonlinear relationships or feature interactions present in the data.

Mathematically, Logistic Regression estimates the probability of the positive class using the sigmoid function:

$$\mathbf{P}(y = 1|X) = \frac{1}{1 + e^{-(\beta_0+\beta_1 x_1+\cdots+\beta_n x_n)}}$$

These model parameters β_i are learned by maximum likelihood estimation, which finds the best coefficients to fit the observed labels. Since the output is a probability between 0 and 1, a threshold (commonly 0.5) is applied to make binary decisions. Logistic Regression assumes independence among features and the effect of each feature is additive and linear in log-odds space.

Analysis of the Results

The Logistic Regression model achieved solid performance across all metrics:

Accuracy	0.8689
Precision	0.8488
Recall	0.7059
F1 Score	0.7706

The relatively high precision suggests that the model is good at identifying the likelihood of student dropout, while the moderate recall indicates that it misses some students who dropped out. The F1 score provides a balanced metric and serves as the benchmark for evaluating the Decision Tree models. Overall, Logistic Regression performs well and sets a reliable baseline, but it may fall short in capturing nonlinear relationships or feature interactions present in the data.

4.1.2 Decision Tree Classifier Model

Decision Trees were chosen because they can capture nonlinear and complex interactions between features without requiring transformations or scaling. Given the diversity in the dataset, ranging from demographics to academic scores, Decision Trees offer a flexible and interpretable model that can adapt to varying feature types and distributions. They also provide clear rules that are useful for stakeholders in education to understand student risk factors.

A Decision Tree is a hierarchical model that splits data recursively based on feature values to predict an outcome. Each internal node represents a decision on a feature, each branch corresponds to an outcome of the decision, and each leaf node represents a final prediction. The goal is to partition the data into increasingly homogeneous subsets with respect to the target variable.

Decision Trees work by evaluating all possible splits at each node and selecting the one that maximises the reduction in impurity. The most common impurity metrics are:
- **Gini impurity:** measures the probability of incorrectly labeling a randomly chosen element.
- **Entropy (Information Gain):** measures the amount of information disorder or randomness in the data.

The tree grows until stopping criteria are met, such as reaching a maximum depth or minimum number of samples per split/leaf. Key parameters control this growth and affect generalisation:

max_depth:	limits how deep the tree can grow
min_samples_split:	the minimum number of samples required to split a node
min_samples_leaf:	the minimum number of samples required to be at a leaf node
criterion:	how to evaluate the quality of split (gini or entropy)
splitter:	strategy used to choose the split (best or random)

Analysis of the Results

The Decision Tree model was tested using more than 50 parameter configurations, and the top 5 performing models based on **F1 score** are summarised below:

	Accuracy	Precision	Recall	F1-score
Model 1: max_depth: 5, min_samples_split: 35, min_samples_leaf: 8, criterion: ‘gini’, max_features: None, splitter: ‘best’	**0.8601**	**0.8357**	**0.7032**	**0.7636**
Model 2: max_depth: 5, min_samples_split: 35, min_samples_leaf: 10, criterion: ‘gini’, max_features: None, splitter: ‘best’	0.8595	0.8312	0.7062	0.7636
Model 3: max_depth: 5, min_samples_split: 10, min_samples_leaf: 8, criterion: ‘gini’, max_features: None, splitter: ‘best’	0.8601	0.8368	0.7022	0.7635
Model 4: max_depth: 5, min_samples_split: 2, min_samples_leaf: 8, criterion: ‘gini’, max_features: None, splitter: ‘best’	0.8601	0.8368	0.7022	0.7635
Model 5: max_depth: 5, min_samples_split: 5, min_samples_leaf: 8, criterion: ‘gini’, max_features: None, splitter: ‘best’	0.8601	0.8368	0.7322	0.7635

Key insights on parameter effects:

1. Consistent optimal max_depth at 5:
   - max_depth = 5 is the most effective parameter choice across all top models, reinforcing the idea that shallow trees perform well on this dataset.
   - This setting balances model complexity and generalisation, preventing overfitting and promoting better performance on unseen data.

2. Regularisation impact with min_samples_split and min_samples_leaf:
    - Higher values for min_samples_split = 35 and min_samples_leaf = 8 have been effective in regularising the model, preventing it from fitting too closely to noise or outliers in the training data.
    - These parameters help avoid overly complex trees that would otherwise overfit, leading to a better model that generalises well.
3. Stable performance with criterion ='gini':
    - Gini consistently performs well across all top models, which suggests that it is an appropriate choice for splitting criteria in this dataset. Gini provides a good balance for binary classification, with its efficiency in splitting the data based on purity.
    - Entropy could have been considered, but Gini proved to be equally or more effective in this case.
4. Splitter = 'best' preference:
    - All of the best-performing models used splitter = 'best', emphasising that the greedy approach for choosing optimal splits leads to better accuracy and stability.
    - This method of selecting the best split at each node consistently outperforms the random split approach.
5. Consistent model performance:
    - The performance metrics (accuracy, precision, recall, and F1 score) for all models are very close, with small variations in precision and recall values, suggesting minimal difference in model configurations.
    - The F1 score consistently hovers around 0.763, indicating a balanced performance in terms of precision and recall.
6. Improvement through regularisation:
    - Increasing min_samples_split and min_samples_leaf to higher values (e.g., 35 and 8) likely helped regularise the model and reduce the chances of overfitting, leading to more stable results.
    - Lower values of min_samples_split (like 2) are still effective but show slightly less regularisation power compared to higher values.

4.1.3 XGBoost (Extreme Gradient Boosting) Model

XGBoost is an optimised machine learning algorithm based on gradient boosting and is widely used for structured/tabular data because of its efficiency, scalability, and superior performance in classification and regression tasks.

XGBoost is an ensemble learning technique that builds multiple weak decision trees sequentially, where each tree corrects the errors of its predecessor. It uses gradient boosting which improves prediction accuracy by minimising the residual errors, regularisation (L1 & L2) to reduce overfitting, and parallel processing for computational efficiency. For this specific task, XGBoost classifies students into two categories "Dropout" and "Not Dropout", based on various input features like past academic performance, financial situation, and personal demographics.

XGBoost algorithm uses loss function like log-loss for classification to ensure that it prioritises reducing incorrect predictions. It works as follows:

1. It initialises a base prediction (the mean of the target variable)
2. It calculates the difference (gradient) between the actual and predicted values
3. A new decision tree is built to model these residuals
4. The process repeats for multiple iterations, each time improving upon previous errors.

Different hyperparameters impact how XGBoost models the data:

learning_rate (default: 0.3):	Controls how much each tree contributes to the final prediction. A smaller value ensures better generalisation but requires more trees.
n_estimators (default: 100):	Number of boosting rounds. Increasing this can improve performance but may lead to overfitting.
max_depth (default: 6):	The depth of trees. Deeper trees capture complex patterns but may overfit.
min_child_weight (default: 1):	Minimum sum of instance weights required in a child node. Higher values prevent overfitting.
subsample (default: 1.0):	Fraction of samples used per tree. Lower values help in regularisation.
colsample_bytree (default: 1.0):	Fraction of features used per tree, reducing correlation between trees.
gamma (default: 0):	Minimum loss reduction required to make a split. Higher values make the model conservative.
lambda, alpha (default: 1, 0):	Regularisation terms to prevent overfitting.

Analysis of the Results

The model uses `RandomSearchCV()` with multiple hyperparameter configurations, and the top 5 performing models based on **F1 Score** are summarised below:

	Accuracy	Precision	Recall	F1 Score
Model 1: colsample_bytree = 1, gamma = 0.2, learning_rate = 0.05, max_depth = 5, min_child_weight = 3, n_estimators = 300, reg_alpha = 0, reg_lambda = 10, subsample = 0.8	**0.8740**	**0.8694**	**0.8364**	**0.8494**
Model 2: colsample_bytree = 1, gamma = 0, learning_rate = 0.1, max_depth = 3, min_child_weight = 1, n_estimators = 100, reg_alpha = 1, reg_lambda = 1, subsample = 0.8	0.8740	0.8704	0.8354	0.8491
Model 3: colsample_bytree = 0.8, gamma = 0.2, learning_rate = 0.1, max_depth = 5, min_child_weight = 3, n_estimators = 100, reg_alpha = 1, reg_lambda = 1, subsample = 0.8	0.8727	0.8699	0.8326	0.8469
Model 4: colsample_bytree = 0.8, gamma = 0.1, learning_rate = 0.05, max_depth = 3, min_child_weight = 1, n_estimators = 300, reg_alpha = 0, reg_lambda = 10, subsample = 0.8	0.8724	0.8690	0.8326	0.8468
Model 5: colsample_bytree = 0.8, gamma = 0.2, learning_rate = 0.05, max_depth = 3, min_child_weight = 3, n_estimators = 300, reg_alpha = 0, reg_lambda = 10, subsample = 0.8	0.8724	0.8693	0.8323	0.8467

Key insights on parameter effects:

- The top-performing model (Model 1) uses deeper tree (max_depth = 5) and more estimators (n_estimators = 300), indicating that a moderately deep tree with more boosting rounds captures patterns effectively. It strikes a balance between learning complexity and prediction stability, avoiding overfitting.
- A lower L1 regularisation (alpha = 0) and higher L2 regularisation (lambda = 10) help the model to generalise better to unseen data.
- A consitent setting of subsample (= 0.8) and colsample_bytree (= 1 or 0.8) across top models promotes diversity in boosting iterations, reducing the risk of overfitting. This combination enhances model robustness without sacrificing performance.
- A moderate gamma value (gamma = 0.2) helps control tree growth by requiring a minimum loss reduction for further partitioning. While a lower learning_rate (= 0.05) allows Model 1 to converge slowly and carefully, leading to improved final performance when combined with more trees.

4.1.4 Random Forest Model

Random Forest is an ensemble machine learning method composed of numerous decision trees. Each tree is constructed from random subsets of the training data and features. In classification tasks, each tree independently classifies input data, and the final decision of class is determined by majority voting among all trees. Compared to other algorithms, Random forest can create more robust and accurate results.

Random Forest naturally handles mixed data types and is suitable for our dataset which comprises numerical (e.g., Admission grades), categorical (e.g. Student's major), and binary features (e.g., Scholarship or not).

Furthermore, Random Forest is resilient to outliers. First, decision trees themselves are tolerant of outliers compared to linear models. They split the data space based on proportion-based metrics like entropy or Gini index. An outlier is less likely to skew the split as it is based on majority proportions. Second, each tree in Random Forest is built on a random subset of data and a random subset of features. This means that any outlier affects only a fraction of the individual trees rather than all of them.

Several of the hyperparameters used in this model are related either to the tree structure or the ensemble design. For example, max_depth, min_samples_split, and min_samples_leaf are tree-based parameters. They control how deep each tree can grow and how many samples are needed to split or form a leaf. These parameters help prevent overly complex trees that may overfit the data.

In contrast, n_estimators and max_features are ensemble-related parameters. n_estimators specifies the number of decision trees in the forest—more trees generally lead to better performance. max_features determines how many features are considered at each split. Using values like "sqrt" or "log2" introduces randomness that helps each tree learn different patterns, increasing the overall diversity and strength of the model. Thus, by combining tree-based learning with ensemble strategies, Random Forest takes advantage of both simplicity and stability, making it suitable for complex, real-world classification tasks like ours.

Different hyperparameters tuning for Random Forest:

n_estimators :	Number of trees, ranges from 100 to 200.
max_depth :	Maximum depth of the tree. None (not limited), 15 and 20 were selected.
min_samples_split and min_samples_leaf:	Minimum samples after splitting and minimum samples in a leaf enforces the tree to be simpler, reducing overfitting. Range 1 to 4 were tested.
max_features:	Number of features to be included at each split (each decision node) within every tree. Sqrt means number of features equal to square root of total number of available features, vice versa. None (not limited), square root and log2 were selected.

Analysis of the Result

The performance of the model is as follows:

	Accuracy	Precision	Recall	F1 Score
Model 1: n_estimator = 100, max_depth = None, min_samples_split = 1, min_samples_leaf = 1, max_features = sqrt	0.8695	0.8674	0.7022	0.7753
Model 2: n_estimator = 100, max_depth = None, min_samples_split = 2, min_samples_leaf = 2, max_features = log2	0.8653	0.8639	0.6921	0.7675
Model 3: n_estimator = 100, max_depth = None, min_samples_split = 4, min_samples_leaf = 4, max_features = None	**0.8718**	**0.8635**	**0.7153**	**0.7819**
Model 4: n_estimator = 150, max_depth = 20, min_samples_split = 3, min_samples_leaf = 3, max_features = sqrt	0.8689	0.8652	0.7032	0.7747
Model 5: n_estimator = 200, max_depth = 15, min_samples_split = 2, min_samples_leaf = 2, max_features = log2	0.8669	0.8762	0.6841	0.7675

4.1.5 Neural Network Model

Neural Network is chosen for their ability to handle complex tasks that other models may struggle with as it has the ability to model complex, nonlinear relationships in the data, which is crucial when the data are not linearly separable or straightforward. It also performs well even with large datasets. As data size increases, it can continue to improve their performance as more information becomes available, making the model a good choice for big data problems. While the relationship between the target group ("Dropout" or "Not Dropout") and features (e.g. Application order, Admission grade and Age at enrollment) may not be linear, Neural Network can be a good model to fit the data.

A Neural Network model is a computational framework inspired by the way biological brains process information which consists of layers of interconnected nodes, "neurons", which work together to process input data and produce outputs.

- **Neurons:** The basic units of a Neural Network. Each neuron receives input, processes them through a mathematical function (such as weighted sum, followed by non-linearity), and outputs a result.
- **Layers:** A Neural Network typically has multiple layers:
  - **Input layer:** This layer receives the raw data.
  - **Hidden layers:** Layers between input and output. These layers perform computations and learn representations of the data.
  - **Output layer:** This layer generates the final predictions or classifications.
- **Weights and Biases:** The connections between neurons have weights, which are learned during training. Each neuron has a bias term that helps to adjust the output.

- **Activation Functions:** Each neuron typically has an activation function (such as Sigmoid and Tanh) that introduces non-linearity, enabling the network to learn complex patterns.

The model learns by minimising a loss function through backpropagation and optimisation techniques such as gradient descent.

Neural Network is a kind of model that learns from data by adjusting weights and biases through backpropagation. The network is composed of layers of neurons that transform inputs to outputs through a series of weighted sums and activation functions. The network is trained using optimization techniques (gradient descent) to minimise loss function. Regularisation techniques are also applied to help prevent overfitting and improve generalisation to new data.

Hyperparameters tuned in this model are:

hidden_layer_sizes :	Defines the architecture of the neural network, specifically the number of neurons in each hidden layer. This parameter is a tuple where each element specifies the number of neurons in a layer. Options provided are 32, 64, 128, 256 neurons for the hidden layers.
activation :	Specifies the activation function for the hidden layer(s). The activation function determines how the weighted sum of inputs is transformed into the output. Options provided are: **'relu'**: Rectified Linear Unit activation function, commonly used in neural networks for its efficiency; **'tanh'**: Hyperbolic tangent activation function which is a popular activation function that outputs values in the range of [-1,1]; **'sigmoid'**: Sigmoid function which transforms any real-valued input into a range between 0 and 1, making it useful for models that need to output probabilities.
solver:	Defines the optimization algorithm used to minimize the loss function. Option provided is **'adam'**: A popular optimization algorithm that adapts the learning rate during training (default solver for *MLPClassifier*).

Analysis of the Results

The performance of the model is as follows:

	Accuracy	Precision	Recall	F1 score
Model 1 (Input layer: 64 neurons, relu activation, Hidden layer: 32 neurons, relu activation, Output layer: 1 neurons, sigmoid activation)	0.8572	0.8479	0.6841	0.7540
Model 2 (Input layer: 128 neurons, relu activation, Hidden layer: 64 neurons, relu activation, Output layer: 1 neurons, sigmoid activation)	0.8579	0.8095	0.7294	0.7666
Model 3 (Input layer: 64 neurons, sigmoid activation, Hidden layer: 32 neurons, sigmoid activation, Output layer: 1 neurons, sigmoid activation)	0.8471	0.8159	0.6821	0.7410
Model 4 (Input layer: 128 neurons, relu activation, Hidden layer 1: 64 neurons, relu activation, Hidden layer 2: 32 neurons, relu activation, Output layer: 1 neurons, sigmoid activation)	0.8446	0.7958	0.7042	0.7430
Model 5 (Input layer: 256 neurons, relu activation, Hidden layer: 64 neurons, relu activation, Hidden layer 2: 32 neurons, relu activation, Output layer: 1 neurons, sigmoid activation)	**0.8579**	**0.8099**	**0.7344**	**0.7687**

4.1.6 Support Vector Machines (SVM)

Support Vector Machines (SVM) are effective for classification tasks, especially when the data is high-dimensional or not easily separable. SVM aims to find an optimal hyperplane that maximises the margin between different classes. It is particularly useful for datasets with many features, like the one used in our study.

SVM was chosen for this model due to its good performance in high-dimensional spaces, effectiveness with numerical and categorical data, and flexibility through different kernels (linear, polynomial, RBF) to handle complex patterns.

SVM works by:

- Finding the optimal hyperplane that separates data points into different classes.
- Maximising the margin between the closest points of different classes (support vectors).
- Using kernel functions (linear, RBF, polynomial) to transform non-linearly separable data into higher dimensions where separation is easier.

The decision boundary is influenced by:

C (Regularisation Parameter):	controls the trade-off between maximizing margin and minimising classification error.
Kernel Type:	determines how the input space is transformed for separation.
Gamma (for RBF Kernel):	controls the influence of individual training examples.

Analysis of the Results

The performance of the model is as follows:

Parameter	Accuracy	Precision	Recall	F1 score
C=0.1, kernel=linear	0.8634	0.8646	0.8634	0.8582
C=1, kernel=linear	**0.8672**	**0.8677**	**0.8672**	**0.8629**
C=1, kernel = rbf, gamma = scale	0.8543	0.8578	0.8543	0.8468
C=10, kernel = rbf, gamma = scale	0.8424	0.8396	0.8424	0.8386
C=10, kernel = rbf, gamma = scale	0.8424	0.8396	0.8424	0.8386

A linear SVM with C=0.1 or C=1 is the best choice for this dataset. This suggests that the data is likely linearly separable in the feature space used, meaning a simple linear decision boundary is sufficient for classification.

4.2 Ensemble Learning Strategies for Multi-Model Integration

4.2.1 Baseline Ensemble Setup

For each type of model, we conducted parameter tuning using k-fold cross-validation to obtain the average accuracy, precision, recall, and F1 score as performance metrics. Based on the highest F1 score achieved across these runs, we selected the best-performing parameter set for each model to construct an ensemble. Four ensemble methods were applied—simple averaging, majority voting, weighted averaging of probabilities, and weighted majority voting—where the weights were assigned according to each model's F1 score: Decision Tree (0.7636), Random Forest (0.7819), SVM (0.8629), XGBoost (0.8494), and Neural Network (0.7687).

The best-performing parameters:

- **Decision Tree**: max_depth=5, min_samples_split=35, min_samples_leaf=8, criterion='gini', splitter='best'
- **Random Forest**: n_estimators=100, max_depth=None, min_samples_split=10, min_samples_leaf=4, max_features=None
- **Support Vector Machines**: C=1, kernel='linear', probability=True
- **XGBoost**: n_estimators=300, learning_rate=0.05, max_depth=5, min_child_weight=3, gamma=0.2, subsample=0.8, colsample_bytree=1, reg_alpha=0, reg_lambda=10, use_label_encoder=False, eval_metric='logloss'
- **Neural Network**: 4 layers-Dense(256, input_dim=X_train.shape[1], activation='tanh'), Dense(64, activation='tanh'), Dense(32, activation='tanh'), Dense(1, activation='sigmoid')

After determining the optimal parameters and corresponding weights, each model was retrained using the entire training set (i.e., the same data used in cross-validation, but now without reserving a fold for validation). The final ensemble evaluation was then conducted on the 20 percent hold-out test set from the original 80:20 train-test split.

The resulting performance metrics for the four ensemble methods—Averaging, Majority Voting, Weighted Probabilities, and Weighted Majority Voting—are as follows:

Averaging		Voting		Weighted probabilities		Weighted majority voting	
Accuracy	0.8773	Accuracy	0.8758	Accuracy	0.8780	Accuracy	0.8758
Precision	0.8626	Precision	0.8599	Precision	0.8630	Precision	0.8599
Recall	0.7354	Recall	0.7330	Recall	0.7377	Recall	0.7330
F1 Score	0.7939	F1 Score	0.7914	F1 Score	0.7955	F1 Score	0.7914

4.2.2 Ensemble Learning with SMOTE for Class Imbalance

Ensemble with SMOTE follows the same configuration described in Section 4.2.1. However, before retraining was performed, we applied the SMOTE technique to the training data to generate synthetic samples for the minority class, resulting in a balanced target class (50-50). SMOTE (Synthetic Minority Over-sampling Technique) works by creating new, synthetic data points through interpolation between existing minority class samples and their nearest neighbours. This helps address the issue of class imbalance, which can otherwise bias model performance. By enhancing the representation of the minority class, SMOTE contributes to improvements in accuracy, precision, and F1 score.

With the same configuration, the results for all four methods are:

Averaging		Voting		Weighted probabilities		Weighted majority voting	
Accuracy	0.8803	Accuracy	0.8855	Accuracy	0.8788	Accuracy	0.8855
Precision	0.8221	Precision	0.8345	Precision	0.8197	Precision	0.8345
Recall	0.8009	Recall	0.8033	Recall	0.7986	Recall	0.8033
F1 Score	0.8114	F1 Score	0.8186	F1 Score	0.8090	F1 Score	0.8186

5 Discussion and Summary

Through the development and evaluation of multiple ensemble strategies, our analysis provided valuable insights into both the data characteristics and the model behaviour under class imbalance. Using F1 Score as the primary evaluation metric allowed us to focus on balancing precision and recall, which is especially important given the skewed distribution in the target labels.

One of the clearest takeaways from our results was that weighted ensemble methods performed better than their unweighted counterparts. Specifically, weighted probability averaging with 79.55% F1 scores, outperformed simple averaging with 79.39% F1 scores. This performance gain can be attributed to the fact that better-performing base models were given greater influence in the final predictions. By assigning higher weights to classifiers that had stronger discriminative power, the ensemble was able to leverage individual model strengths more effectively.

Interestingly, we also found that simple ensemble methods were surprisingly competitive, particularly simple averaging. Despite its conceptual simplicity and lack of weighting, this approach still achieved reasonable performance. Its effectiveness seemed to stem from the complementary nature of the base models used, where one model favoured precision and another recall, their averaged outputs often balanced each other out, producing more stable and generalisable results.

Another key observation was the role of SMOTE in improving the performance for minority classes. After applying SMOTE to the training data, all ensemble methods demonstrated noticeable **in-**

creases by 1-2% in F1 Scores. The improvement was especially pronounced in the recall scores for underrepresented classes. This confirms that synthetic oversampling was effective in mitigating class imbalance, allowing models to better recognise patterns in the minority classes rather than defaulting to majority predictions.

Areas for Improvement

While the ensemble strategies implemented in this study yielded promising results, there remain several avenues to further enhance performance and gain deeper insights.

Firstly, one of the most promising directions is to optimise the ensemble weights more systematically. In our current weighted ensemble methods (weighted probability averaging and weighted majority voting), weights were assigned manually based on preliminary observations of individual model performance. While effective, a more rigorous approach, such as performing a grid search or using optimisation algorithms to learn optimal weights through cross-validation, could lead to improved predictive accuracy and F1 Scores.

Therefore, we recommend exploring confidence-based [Li et al., 2014] or dynamic ensemble weighting techniques [Do et al., 2020]. Instead of using fixed weights per model, predictions could be adjusted based on each model's confidence in a given prediction. This allows the ensemble to dynamically shift its trust toward models that are more certain or have historically performed better for specific classes or feature spaces.

Lastly, we recommend incorporating model interpretability tools, such as SHAP (SHapley Additive exPlanations) or LIME (Local Interpretable Model-Agnostic Explanations) [Hasan, 2024]. These tools can provide transparency into how each feature influences the predictions of individual base models and, more importantly, how those insights aggregate in the ensemble's final decision. Understanding which features drive performance can also feed back into the data pipeline, potentially guiding future feature engineering or collection efforts.

References

[Ang, 2024] Ang (2024). Around 17,000 youth in singapore are not in school, work or training. Accessed: 2025-03-17.

[Do et al., 2020] Do, D. T., Nguyen, T. T., Nguyen, T. K., Luong, V. A., Liew, A. W.-C., and McCall, J. (2020). Confidence in prediction: An approach for dynamic weighted ensemble. In *Intelligent Information and Database Systems*, pages 358–370. Springer.

[Hasan, 2024] Hasan, M. M. (2024). Understanding model predictions: A comparative analysis of shap and lime on various ml algorithms. *Journal of Scientific and Technological Research*, 5:17–26.

[Li et al., 2014] Li, L., Hu, Q., Wu, X., and Yu, D. (2014). Exploration of classification confidence in ensemble learning. *Pattern Recognition*, 47(9):3120–3131.

[Martins et al., 2023] Martins, M. V., Baptista, L., Machado, J., and Realinho, V. (2023). Multi-class phased prediction of academic performance and dropout in higher education. *Applied Sciences*, 13(8):4702.

A Appendix: Dataset Features

Table 1: Dataset Features Table

Index	Feature Name	Data Type	Description
1	Marital status	Nominal	Marital status of the student
2	Application mode	Nominal	Type of admission (e.g., national exam, direct entry).
3	Application order	Discrete	Rank of the student's application choice.
4	Course	Nominal	Course enrolled in.
5	Daytime/evening attendance	Binary	1 = Daytime program, 0 = Evening program.
6	Previous qualification	Nominal	Student's academic qualification before enrolling in university.
7	Previous qualification (grade)	Continuous	Average grade obtained in national exams.
8	Nationality	Nominal	Country of nationality.
9	Mother's qualification	Nominal	Highest education level of the mother.
10	Father's qualification	Nominal	Highest education level of the father.
11	Mother's occupation	Nominal	Job classification of the mother.
12	Father's occupation	Nominal	Job classification of the father.
13	Admission grade	Continuous	Final grade at the time of admission.
14	Displaced	Binary	1 = The student moved from another region to attend university, 0 = Local student.
15	Educational special needs	Binary	1 = Student has special education needs, 0 = No special needs.
16	Debtor	Binary	1 = Student has outstanding university fees, 0 = No debt.
17	Tuition fees up to date	Binary	1 = Student paid tuition on time, 0 = Delayed payments.
18	Gender	Binary	1 = Male, 0 = Female.
19	Scholarship holder	Binary	1 = Student receives a scholarship, 0 = No scholarship.
20	Age at enrollment	Discrete	Age of student at the time of enrollment.
21	International	Binary	1 = Student is from another country, 0 = Domestic student.
22	Curricular units 1st sem (credited)	Discrete	Number of curricular units credited in the first semester.
23	Curricular units 1st sem (enrolled)	Discrete	Number of curricular units enrolled in the first semester.
24	Curricular units 1st sem (evaluations)	Discrete	Number of curricular units evaluated in the first semester.
25	Curricular units 1st sem (approved)	Discrete	Number of curricular units approved in the first semester.

Index	Feature Name	Data Type	Description
26	Curricular units 1st sem (grade)	Continuous	Grade obtained in the first semester.
27	Curricular units 1st sem (without evaluations)	Discrete	Number of curricular units without evaluation in the first semester.
28	Curricular units 2nd sem (credited)	Discrete	Number of curricular units credited in the second semester.
29	Curricular units 2nd sem (enrolled)	Discrete	Number of curricular units enrolled in the second semester.
30	Curricular units 2nd sem (evaluations)	Discrete	Number of curricular units evaluated in the second semester.
31	Curricular units 2nd sem (approved)	Discrete	Number of curricular units approved in the second semester.
32	Curricular units 2nd sem (grade)	Continuous	Grade obtained in the second semester.
33	Curricular units 2nd sem (without evaluations)	Discrete	Number of curricular units without evaluation in the second semester.
34	Unemployment rate	Continuous	Portugal's unemployment rate at the time of enrollment.
35	Inflation rate	Continuous	Portugal's inflation rate at the time of enrollment.
36	GDP	Continuous	Portugal's GDP growth rate at the time of enrollment.
37	Target	Discrete	If the student graduated, enrolled, or dropped out.

B Appendix: Simple Statistical Analysis

Figure 1: Correlation Matrix of Numerical Variables

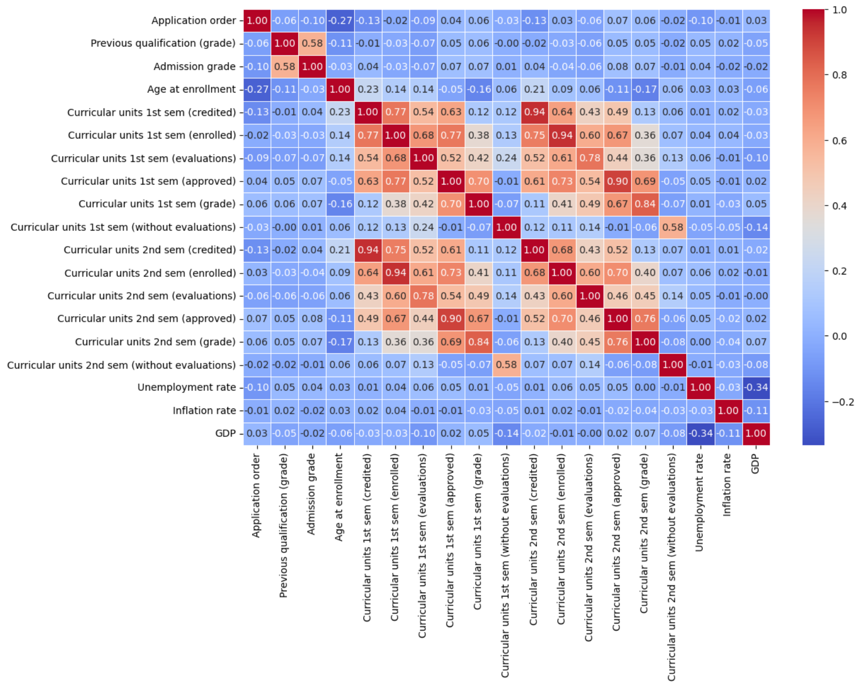


Figure 2: Histograms of Numerical Features

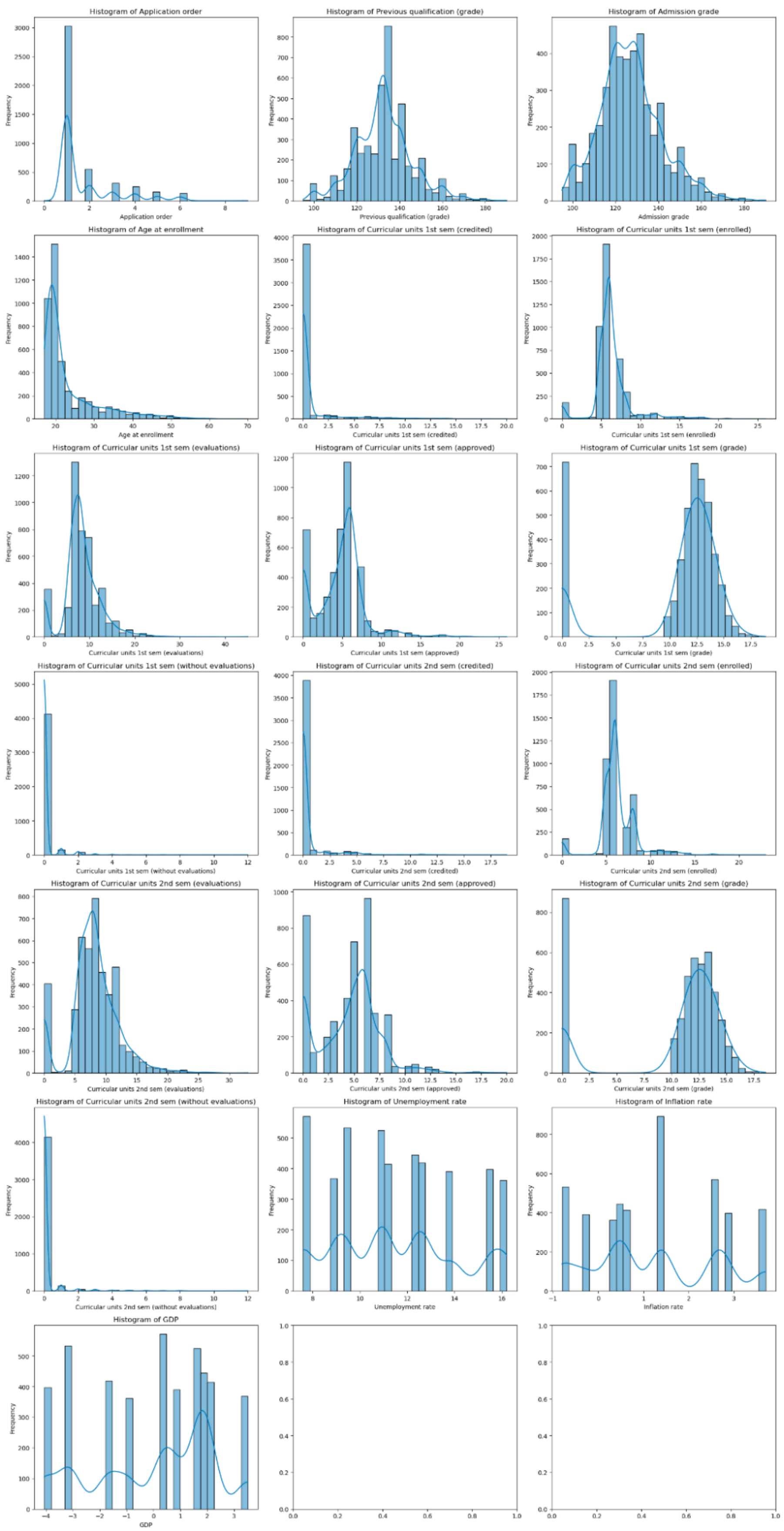
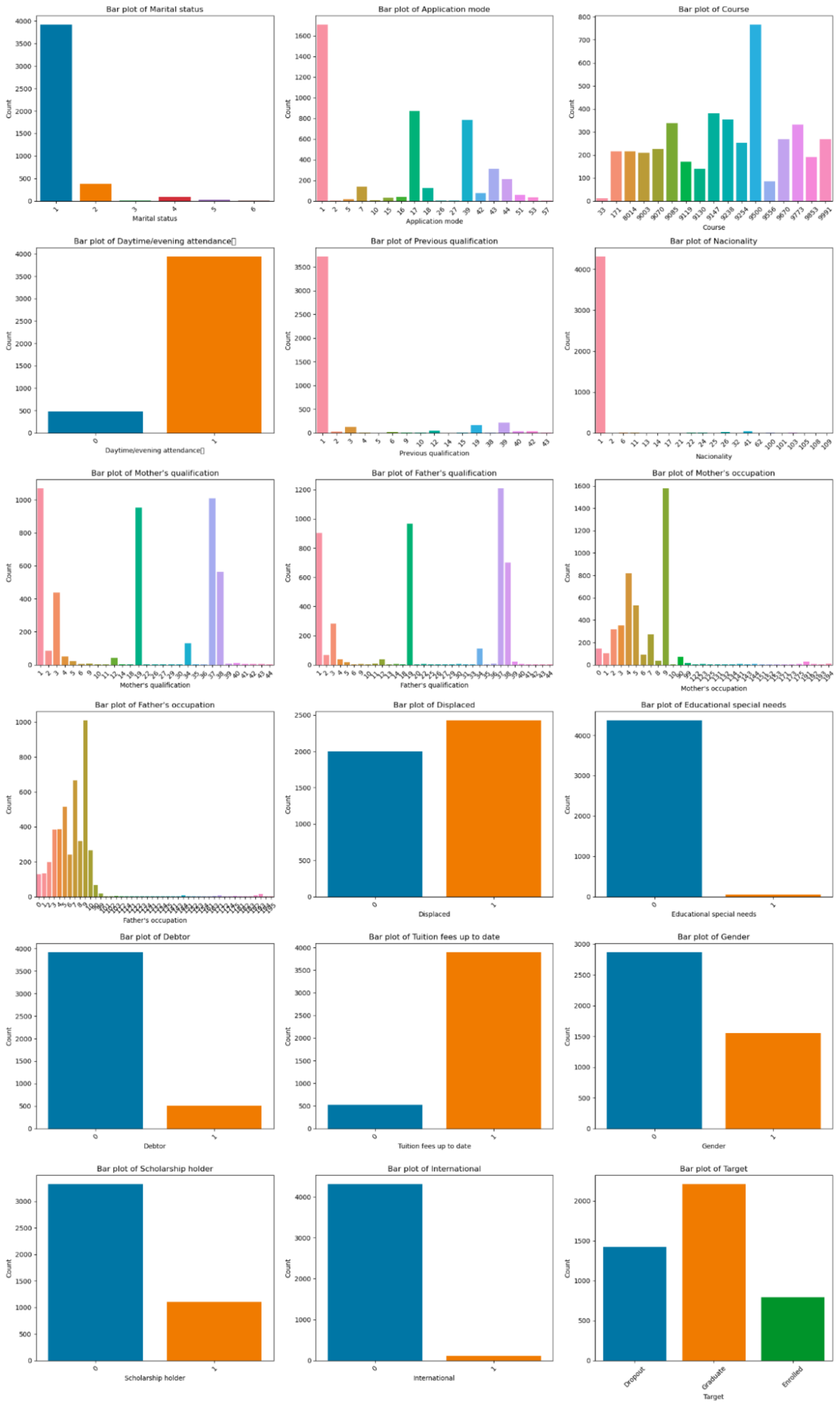


Figure 3: Barplots of Categorical Features


Bar plot of Marital status
Marital status
Bar plot of Application mode
Application mode
Bar plot of Course
Course
Bar plot of Daytime/evening attendance
Daytime/evening attendance
Bar plot of Previous qualification
Previous qualification
Bar plot of Nacionality
Nacionality
Bar plot of Mother's qualification
Mother's qualification
Bar plot of Father's qualification
Father's qualification
Bar plot of Mother's occupation
Mother's occupation
Bar plot of Father's occupation
Father's occupation
Bar plot of Displaced
Displaced
Bar plot of Educational special needs
Educational special needs
Bar plot of Debtor
Debtor
Bar plot of Tuition fees up to date
Tuition fees up to date
Bar plot of Gender
Gender
Bar plot of Scholarship holder
Scholarship holder
Bar plot of International
International
Bar plot of Target
Target
Dropout
Graduate
Enrolled
Count