Abstract

The increasing prevalence of student dropouts globally has proved to be a concerning and pressing matter for educational institutions. In the light of the growing development and use of machine learning models in the predictive analytics realm, this study aims to leverage on the three machine learning models namely, Extreme Gradient Boosting (XGBoost), Random Forest, and the Decision Tree model to help in predicting students whom are at risk of dropping out. Attributes that are effective predictors of students dropping out has also been identified. These attributes are namely, the number of curricular units approved in the second semester, the number of curricular units that are approved in the first semester, the average grade in the first and second semester, and lastly the status of the student's tuition fees (up-to-date or not). The dataset employed for the purpose of this study comprises of 4424 observations with 37 attributes (inclusive of the Target variable). Exploration of the impact of hyperparameter tuning is also another objective of the study. The best-performing model of this study is the XGBoost model with hyperparameters tuned using firstly the Random Search technique followed by the Grid Search technique. The model's Accuracy stood at 77.39% with a Dropout class-specific Precision and Recall of 0.8889 and 0.7324 respectively. AUC value of the model stood at 0.8922 - the highest among the other two models implemented.

Keywords: Predicting Student Dropouts, XGBoost, Random Forest, Decision Tree, Hyperparameter Tuning

Table of Contents

1.0 Introduction

1.1 Background of the Domain Discussed

With the advancement in the integration of machine learning and data mining technologies, analytics involving the prediction of students' academic performance or more precisely, the prediction of student at risk of dropping out has too been advancing and increasing in number. Often referred to as Educational Data Mining (EDM), such exploration of student academic data for predictive and subsequently prescriptive purposes has established itself as a growing area of research more so in the face of big data as well as the development of database management systems (Al-Razgan et al., 2013). The process of EDM involves applying data mining techniques to academic data with the intention of unravelling patterns and insights about the education domain. Areas of concern usually include student academic achievements, student retention rate, dropout rates, and failure rates to name a few. Data employed in these analyses often times comprises of information relating to the academic performance of the student, their demographics and socioeconomics information, macroeconomics indicators, educational trajectories, and the details of students that are available at the time of enrolment, to name a few.

1.2 Significance of the Study

Efforts to predict the academic outcomes of students and the prediction of students at risk of dropping out could not be emphasized any lesser for several reasons. Firstly, predictive analytics in this regard could provide means for academic institutions to promptly identify those who are at risk or are struggling academically, ultimately enabling for timely interventions and support to be made available. That is, by approaching the issue in a proactive manner, institutions will then be able to considerably improve the retention and success rates of students. Besides, the identification of factors that contribute to the academic regression and eventually the dropping out of students from schools could serve as valuable insights for all stakeholders within the education domain to better strategize and tailor to the betterment of students. Also, it is worth noting that keeping dropout rates low in fact, has broader societal benefits. That is, because more education is typically associated to improved employability and employment opportunities, economic growth would then ensue as a result of that. Lastly, and most likely the most apparent justification for predictive analytics is that by predicting dropouts, it ensures that resources are allocated in an effective manner, essentially ensuring that support is given to those who need it most.

1.3 Aim and Objectives

1.3.1 Aim

The aim of this study is to identify the best-performing model for predicting students' academic outcomes, specifically through a comprehensive exploration of the tuning of hyperparameters within each model.

1.3.2 Objectives

The aim of this study could be further broken down into several smaller objectives and the objectives are as follows:

1. To evaluate and subsequently identify the most reliable and accurate model for the prediction of the academic outcomes of students, more precisely, those who are at risk of dropping out.
2. To determine the most important indicators of the academic outcomes of students.
3. To demonstrate the influence of hyperparameter tuning on the performance of machine learning models when predicting the academic outcomes of students.
4. To provide tangible recommendations to academic institutions and all concerned stakeholders in terms of mitigating dropout rates and academic regression.

1.4 Scope of the Study

This section defines the boundaries of the study carried out. The boundaries are as outlined down below:

- **Data features:** The dataset employed within the study was obtained from Martins et al. (2023). While details of the dataset would be discussed in greater depth in the sections to come, the dataset is distinctive to the 4424 student entries it contained.
- **Domain of Study:** Fields of study outside the scope of those included in the dataset will not be examined in the study, essentially narrowing down the focus of the analysis solely on the fields of study of the students contained within the dataset only.
- **Temporal Scope:** Data is limited to the period between 2008 to 2019 where only records from the school year of 2008/2009 to that of 2018/2019 are contained within the dataset.
- **Modelling Techniques:** Machine Learning Models examined within the study are Extreme Gradient Boosting (XGBoost) model, Random Forest (RF) model, and Decision Tree (DT) model.

- **Evaluation Measures:** Metrics used to evaluate the models examined in the study are namely, Accuracy, Precision, Recall, and Area Under the Receiver Operating Characteristic Curve (AUC-ROC).
- **Software Tools:** Programming tools employed by the study is limited to RStudio.
- **Coding Language:** The programming language applied is the R Language.

2.0 Related Work

2.1 Review of Student Dropout Prediction Models

Martins, Baptista, Machado, and Realinho (2023) compared different machine learning algorithms in terms of their ability to predict student dropouts together with the ideal time to perform such prediction for optimal interventions. Evaluation of the algorithms were made for three different periods namely during enrolment, the first semester, and the second semester of the first academic year of the students. The dataset employed consisted of information about the academic, demographic, macroeconomic and socio-economic situations of students who have enrolled between 2009 to 2017 in one of the Polytechnic Universities in Portugal. Results from the study indicate that the Random Forest (RF) algorithm had the best predicting ability and that the earliest of such predictions should be made at the end of the first semester and no later.

Dissimilar to their most recent works, Realinho, Machado, Baptista and Martins (2022) in one of their earlier works have instead made comparisons between the features which are important to the different machine learning models evaluated when using the models to predict student dropouts. The dataset employed consisted of 4424 instances and 35 attributes (inclusive of the class label). Results from the study show that features do differ from models to models. The models evaluated consisted of the Random Forest (RF) model, Extreme Gradient Boosting (XGBOOST) model, Light Gradient Boosting Machine (LIGHTGBM) model and CatBoost (CATBOOST) model. Having discussed the above two studies, it is important to note that although the datasets in both studies seem to have the same attributes as the dataset employed in this study, there are actually not referring to the same dataset.

When evaluating the different performances of machine learning techniques in predicting student dropouts, Capuno, Ferrer, Manaloto and Villafria (2023) and Jiménez, Jesús, and Wong (2023) found that the Random Forest (RF) model had the best predicting ability among the other models compared within their respective study. Consistent with their findings, Dasi and Kanakala (2022) in their work concluded that models namely the Random Forest (RF), Logistic Regression, Decision Tree and Neural Network were among the best models of their study. No mention of dataset used was made for the study conducted by Capuno, Ferrer, Manaloto and Villafria (2023). Dataset for the study conducted by Dasi and Kanakala (2022) and Jiménez, Jesús, and Wong (2023) were from Kaggle (261 instances with 10 attributes) and from survey results conducted in several public and private universities in Peru respectively. 8

Another work which shares great similarities with the dataset used in this paper would be that of Martins, Tolledo, Machado, Baptista and Realinho (2021) where they too aim to identify students at risk of dropping out through the use of both classical machine learning algorithms as well as those of advanced boosting algorithms. Great similarities were said to exist because the dataset that they employed in fact contained attributes that are subsets of the current dataset of this paper. That is, the former dataset contained only students' information available at the time of enrolment, while the latter dataset, on top of the information obtained at the time of enrolment, does also contain additional information pertaining to the academic achievement of students post enrolment. Among the classical and state of art algorithms evaluated, the authors concluded that the Random Forest (RF) algorithm performed best among its own classical kind while the Extreme Gradient Boosting (XGBoost) algorithm championed the rest of its kind. Final remark from the authors was that boosting models do perform better than classical models when carrying out classification tasks.

2.2 Research Gap

While many of the existing literatures are driven in the same direction as that of this study, that is to essentially reduce academic dropouts and academic regression, there remained however, several research gaps which are left unaddressed and unexplored. These gaps are outlined as follows, and for each gap identified, this study would propose a specified approach to address them:

- **Gap in Hyperparameter Tuning:** Many literatures failed to comprehensively explore the impact of the tuning of the hyperparameter of the model on the performance of the model. Such research gap needs to be promptly addressed because the role of hyperparameter tuning is to optimize the settings a machine learning model, ultimately improving its performance and accuracy on a particular task. It ensures that there is a balance between the fitness (overfitting and underfitting) of the model and that of the generalizability of the model. Given that, this study aims to address the gap by carrying out a comprehensive hyperparameter optimization process for each of the three models examined, ultimately ensuring that the most optimal model configurations for the purpose of predicting academic outcomes are identified.
- **Lack of Documentation:** While many literatures were detailed about the performances of the machine learning models examined in their respective studies, current research

still falls short in providing detailed documentation of the hyperparameter tuning processes which they have carried out. Detailed documentation in this regard is especially important in ensuring that the analysis could be replicated by people other than the authors. That said, this study intends to closely detail the hyperparameter tuning process so that other academics or other interested parties would then be able to replicate the findings, obtain insights from the methodologies implemented, and ultimately to further propel the advancement of predictive modelling within the educational settings.

3.0 Methods

3.1 Description of the Dataset

3.1.1 Source of the Dataset

The dataset was obtained from a tertiary education institution where entries of the dataset consisted of the records of students from the school year of 2008/2009 to that of 2018/2019 (Martins et al. 2021). Students' profiles included those from various backgrounds specifically in terms of demographics, socioeconomics, academics. The various academic disciplines of the students within the dataset are namely, information technologies, business management, social work, journalism, nursing, education, design, as well as agronomy studies. All data are gathered from three sources – one of the sources being from the institution's Academic Management System (AMS), another one being from the General Directorate of Higher Education (DGES), and the third one being that from the Contemporary Portugal Database (PORDATA) (Realinho et al., 2022; Martins et al., 2023). The compiled and final dataset is accessible in a Comma Separated Value (CSV) file with UTF8 encodings.

3.1.2 Attributes and Target Labels of the Dataset

The dataset consisted of 4424 entries and a total of 37 attributes (including the Target variable). The 36 attributes (excluding the Target variable) could be further categorized into either of the 6 categories namely, Demographics, Socioeconomics, Macroeconomics, Academics Data Upon Enrolment, Academic Data at the Conclusion of the First Semester, and lastly, Academic Data at the Conclusion of the Second Semester. The dataset also features a wide range of data types, precisely, categorical data, discrete data, numerical data, and also those with binary properties. There are three classes to the Target variable, and they are, "Enrolled", "Graduate", and, "Dropout". A comprehensive overview of the attributes together with their respective data types and grouping could be found in the table below.

Attribute	Group	Type of Data
Unemployment rate	Macroeconomics Data	Numeric
GDP		Numeric
Inflation rate		Numeric
Debtor	Socioeconomics Data	Binary
Tuition fees up to date		Binary
Mother qualification		Discrete
Father qualification		Discrete
Mother occupation		Discrete
Father occupation		Discrete
Educational special needs		Binary
Scholarship holder		Binary
Gender	Demographics Data	Binary
Displaced		Binary
Nacionality		Discrete
Marital status		Discrete
Age at enrollment		Discrete
International		Binary
Application order	Academic Data Upon Enrolment	Discrete
Application mode		Discrete
Course		Discrete
Admission grade		Numeric
Previous qualification		Numeric
Previous qualification (grade)		Numeric
Daytime/evening attendance		Binary
Curricular units 1st sem (credited)	Academic Data at the Conclusion of the First Semester	Discrete
Curricular units 1st sem (enrolled)		Discrete
Curricular units 1st sem (evaluations)		Discrete
Curricular units 1st sem (approved)		Discrete
Curricular units 1st sem (grade)		Numeric

Curricular units 1st sem (without evaluations)		Discrete
Curricular units 2nd sem (credited)	Academic Data at the Conclusion of the Second Semester	Discrete
Curricular units 2nd sem (enrolled)		Discrete
Curricular units 2nd sem (evaluations)		Discrete
Curricular units 2nd sem (approved)		Discrete
Curricular units 2nd sem (grade)		Numeric
Curricular units 2nd sem (without evaluations)		Discrete
Target	Target	Categorical

3.2 Learning Techniques

3.2.1 Programming Tools and Language Employed in the Study

3.2.1.1 Programming Language

The R programming language was being employed in this study. The choice of the said programming language is due to its robust capabilities in statistical analysis and data visualization, both of which could be noted in the few sections to come. The language's extensive set of packages which are developed specifically for machine learning and data mining processes, was also another added reason for choosing R in this study. R's strong and supportive community, which is known for its active and communal nature, where resources are freely and extensively shared among the community, meant that solutions and best practices could be readily available, and that any recent developments in statistical analysis and methodologies could be too be easily assessed. Lastly, the fact that the language is an open-source programming language signifies that continued accessibility of the language is guaranteed throughout the duration of the study carried out in this report, without the need for acquiring any expensive software licenses.

3.2.1.2 Programming Tools

The programming tool engaged during this study is RStudio. Justifications for the choice of programming tool are stated as follows:

1. **RStudio:** RStudio on the other hand, was used extensively throughout the study to conduct the necessary data analysis. Processes like initial data explorations, data preprocessing, Exploratory Data Analysis (EDA), and the implementation and evaluation of the machine learning models of the study were all performed within the RStudio's environment. Predominantly designed as an Integrated Development Environment (IDE) for the R programming language, the choice of RStudio was thereby driven by its compatible complement to the study's selection of the R language as previously mentioned above. By leveraging on the user-friendly interface of RStudio in maximizing the capabilities of the R programming languages, the workflow of this study could hence be streamlined and made cohesive.

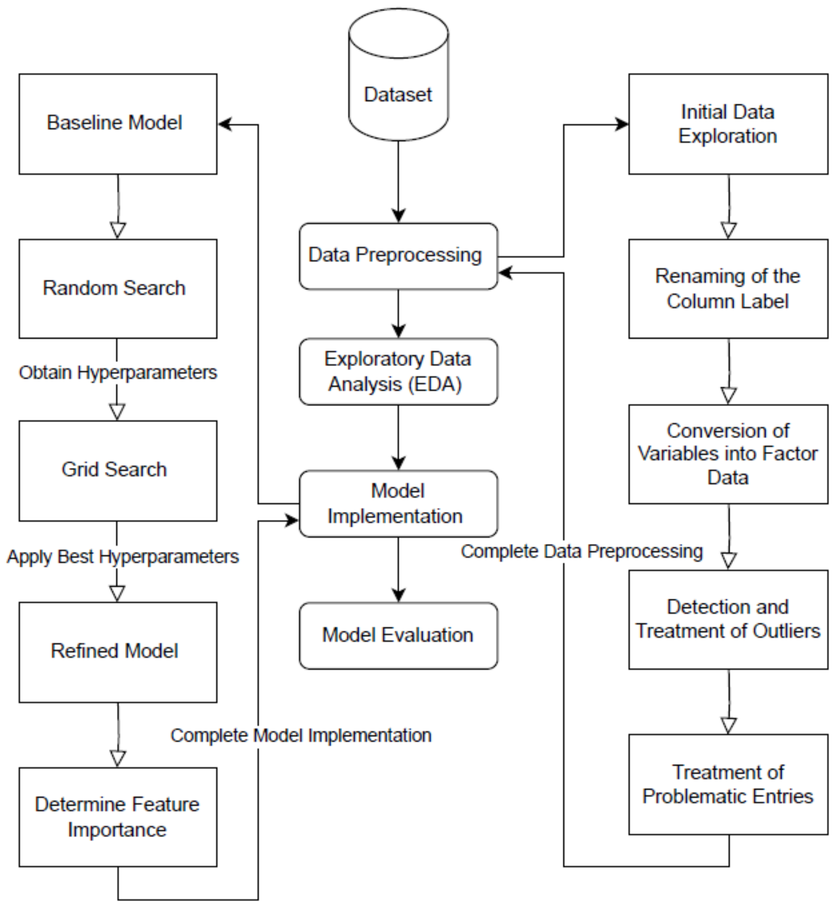
3.2.2 Packages Utilized

To extend the IDE's capabilities beyond its fundamental set of functionalities that comes with its installation, several R packages were installed and subsequently utilized during the analysis process of this study. The table below outlines the packages which were installed and the description on the respective roles that they play in this study. While certain packages mentioned in the table below may be utilized more extensively while others may not, all packages are however essential in making sure that the entire data analysis process remained feasible and productive.

Packages	Description
reshape2	A package used to facilitate for the restructuring and aggregating of data, enabling for the transformation of data between wide and long formats.
rpart	A package often used for the implementation of iterative partitioning for tasks involving classification, regression, and survival trees.
MLmetrics	The package contains a set of evaluation metrics which are useful for the evaluation of the performance of machine learning models.
pROC	A package known for its visualization, smoothing, and comparisons of the Receiver Operating Characteristics (ROC) curves.
dplyr	A core package for performing data manipulation.
missForest	A package for imputation tasks where missing values are imputed using a non-parametric random forest method.
DataExplorer	A package that facilitates for Exploratory Data Analysis, offering functions for exploration and summarization of data.
caret	A wide-ranging package made for the purpose of developing predictive models, where the process of model training and model tuning are streamlined across a diverse array of algorithms.
ROCR	A package that enables for the visualization of the effectiveness of scoring classifiers. Emphasis is place on the ROC curves.
ggplot2	A package which is extensively used to produce high-quality and customizable graphical plots.
caTools	A package that contains functions including reading and writing specific binary data formats, smoothing and manipulating of data, as well as for the moving of window statistics.
xgboost	A library that provides for the effective execution of gradient boosting.
randomForest	A package that supports the implementation of the random forest algorithm for tasks involving classification and regression. The package is acknowledged for its performance and ability to estimate feature significance.

3.2.3 Flowchart of the Data Analysis Process

The flowchart below shows the entire data analysis process of this study. It outlines a structured approach to machine learning classification, beginning with the dataset, followed by the preprocessing data, EDA, model implementation, and lastly, the evaluation of the models examined. Discussions on each of the stages which could be found in the following sections beginning from that of Section 5.0, will all be in accordance to the logical sequencing of the flowchart.


Dataset
Baseline Model
Random Search
Obtain Hyperparameters
Grid Search
Apply Best Hyperparameters
Refined Model
Determine Feature Importance
Complete Model Implementation
Data Preprocessing
Exploratory Data Analysis (EDA)
Model Implementation
Model Evaluation
Complete Data Preprocessing
Initial Data Exploration
Renaming of the Column Label
Conversion of Variables into Factor Data
Detection and Treatment of Outliers
Treatment of Problematic Entries

3.2.4 Rationale for the Choice of Machine Learning Models

Discussion contained within this section will address the justifications for the choice of the machine learning models of this study. Three machine learning models were selected, namely, XGBoost, Random Forest, and Decision Tree model. Putting aside the rationale of the said models having superior performance, which is one apparent reason for including them in this study, the three models' suitability with regards to the given classification task and domain, was also reasons supporting the inclusion of them in this study. While majority of the reviewed literatures in the previous sections did not explicitly justify their choice of the said models, the chosen models (XGBoost, Random Forest, and Decision Tree) are however, not without sound reasonings. Given that, this section would serve to provide the relevant justifications where separate reasonings would be made for each of the three models.

Machine Learning Model	**Justifications**
XGBoost	▪ **Ability to Handle Missing Data**: Given its ability to determine the optimal direction for missing values when making splits in the data, the model is thereby best suited for the given dataset of this study, bearing in mind that there are a considerable number of missing values flagged in the dataset. The mentioning of the discovery and assigning of missing values within the dataset could be found in the later section. ▪ **Model Complexity Control:** As the model is integrated with regularization parameters to avoid the occurrence of excessive training data memorization, in other words, overfitting, the model is thereby befitting for the dataset of this study, where missing values and a large number of attributes are present.
Random Forest	▪ **Ability to Handle Missing Data:** Similar to that of the XGBoost model, Random Forest is also very competent in treating missing values. These values are managed internally during the training process where

	the Random Forest model would implement alternative splits to determine the best division among the features. ▪ **Resiliency against Overfitting**: The model's robustness to overfitting lies in its ensemble learning manner. That is, even though the model is complex, the model's ensemble approach ensures that the issue of overfitting could be mitigated as predictions from multiple decision trees are being aggregated. ▪ **Diversity in Overall Model**: Owing to its ensemble learning approach, that is, because the model would aggregate many decision trees together, a varied perspective on the data is thereby possible. Diversity in the context of incomplete datasets is especially favourable because when different trees handled the missing values differently, it essentially does ensure that the overall model is robust. ▪ **Significance of Attributes**: Given that the dataset employed in this study contained a significant number of features, the Random Forest's built-in mechanism for identifying the importance of each attribute is then especially valuable. This is more so in the educational settings where interactions are often times complex and multi-faceted. Recall that one of the objectives of this study includes the identifying of features that are informative of the academic outcomes of students.
Decision Tree	▪ **Increased Interpretability**: Decision Tree models are known for being simple and easy to interpret. This is a value-added in the context of this study as delivering model's insights to stakeholders for decision-making purposes is especially important. ▪ **Versatility in Handling Missing Values**: The model is able to treat missing values in various ways, either through imputation, surrogate splits, or simply by

	disregarding the missing values altogether when performing the split. ▪ Significance of Attributes: Similar to that of the Random Forest model, the Decision Tree model is able to identify the most telling features while performing the training process.

3.2.5 Rationale for the Choice of Evaluation Metrics

While model implementation is undoubtedly instrumental in the entirety of this study, evaluation of the performances of all models implemented is equally important nonetheless. Evaluation is necessary to ascertain the models' true capabilities in accurately addressing the task at hand, and to at the same time, ensure that they are able to learn and adapt to unseen and new data. Evaluation metrics selected for this study are namely, the Accuracy metric, the Precision metric, the Recall metric, and lastly, the AUC-ROC metric. These metrics serve to provide distinct perspectives into the performance of the models, ultimately offering for a complete and comprehensive understanding of how well the models were at predicting the academic outcomes of students. Each of the said metrics would be briefly discussed in the table down below.

Evaluation Metric	Description
Accuracy	The Accuracy metric is a measurement of the proportion of predictions which were correctly made out of the total number of predictions made during the modelling. Generally, a higher accuracy is thereby desirable.
Precision	The Precision metric is a measurement of the accuracy of the positive predictions which is made by the model. Precision in the context of this study meant that when the machine learning model predicted a student as "Dropout", the metric would reflect how likely it will be that the student truly belongs to the "Dropout" category. Hence, aligning with the overall aim of this study, that is to better identify students at risk of dropping out, a high precision value for all target classes, more so for that of the "Dropout" class, is thus desired.
Recall	In the context of a classification problem, the Recall metric, sometimes also known as the Sensitivity metric, is a measurement that indicates the ability of the model to accurately identify all instances of a certain class. In the case of this study, a high Recall in the "Dropout" class is desired because it simply meant that the model was able to effectively capture most of the students who are at risk of dropping out. Similar logic applies to that of the "Graduate" and "Enrolled" classes.
AUC-ROC	The AUC-ROC metric is a measurement used to assess the ability of classification models to make distinctions between the target classes. The ROC curve is a probability curve which plots the True Positive Rate against the False Positive Rate, at varying threshold settings. The AUC however, is a value that represents the entire area under the ROC curve. A higher AUC value is therefore desired as it is indicative of a better performing classification model.

It is important to note that, ideally, a balance has to be achieved between the Precision and Recall metrics. Balance such that both metrics such remain high, that is, the model should accurately identify majority of the students who are at risk of dropping out (represented by a high Recall value) but at the same time should also ensure that those who are identified as at risk are indeed susceptible to dropping out (represented by a high Precision value). Also, it is noteworthy that all four of the metrics mentioned above would be measured for both the training and test datasets. This is to ensure that the models are not only working well with the provided dataset, but also on any unseen and new data, essentially ensuring for generalizability of the model. At the same time, these metrics are crucial in making sure there is no issue of overfitting, where the model performed well only on the training data but performed otherwise on the test data. Fitness of model is hence checked for.

4.0 Preparation of the Dataset

The following subsections would address the necessary processes to be carried out prior to the implementation of the three models previously mentioned.

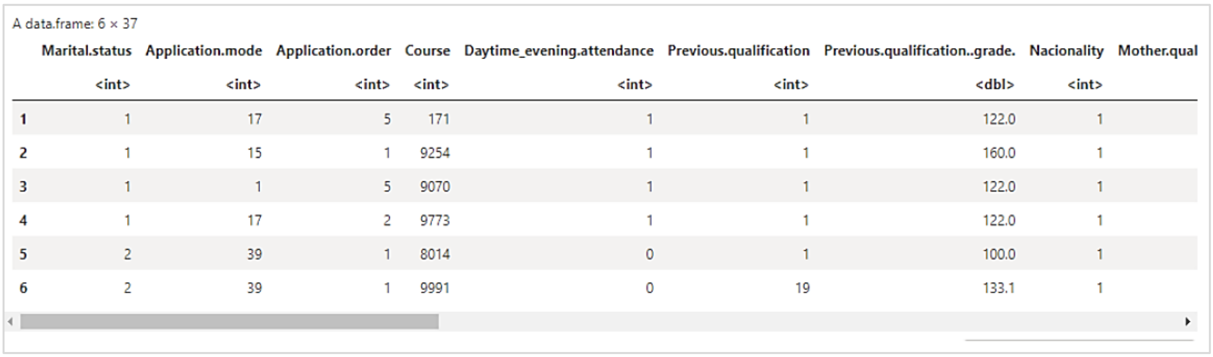
4.1 Data Preprocessing

4.1.1 Initial Data Exploration

The purpose of performing initial data exploration is to ensure that fundamental understanding of the dataset's characteristics and structure are acquired before any further data cleaning, data transformation or model selection is to be carried out. As a result of this preliminary exploration, possibly present patterns, anomalies, data quality issues, and the relationships between attributes could then be effectively captured and identified. A thorough understanding of the dataset then lays the groundwork for an informed data analysis subsequently.

4.1.1.1 Quick Preview of Dataset

```
# Section 2.0: Data Preprocessing

# Subsection 2.1 Initial exploration of the dataset

# Show initial entries of the dataset for a quick preview
head(ds)
```



```
A data.frame: 6 × 37
  Marital.status Application.mode Application.order Course Daytime_evening.attendance Previous.qualification Previous.qualification..grade. Nacionality Mother.qual
           <int>            <int>             <int>  <int>                      <int>                  <int>                          <dbl>       <int>
1              1               17                 5    171                          1                      1                          122.0           1
2              1               15                 1   9254                          1                      1                          160.0           1
3              1                1                 5   9070                          1                      1                          122.0           1
4              1               17                 2   9773                          1                      1                          122.0           1
5              2               39                 1   8014                          0                      1                          100.0           1
6              2               39                 1   9991                          0                     19                          133.1           1
```

Once the dataset has been successfully loaded into the IDE (RStudio), the head() function was called on to display the first few entries of the dataset. From the code output shown above, it is evident that the dataset contained a total of 37 attributes, where one of them is the Target variable. Upon observation, and consistent with that which was discussed in previous sections, it is revealed that the dataset indeed contained a blend of data types, precisely, numerical, categorical, binary and discrete data types.

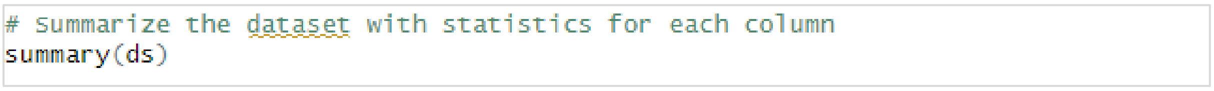
4.1.1.2 Structure of the Dataset

```
# Examine the dataset's structure, including data types and column information
str(ds)
```

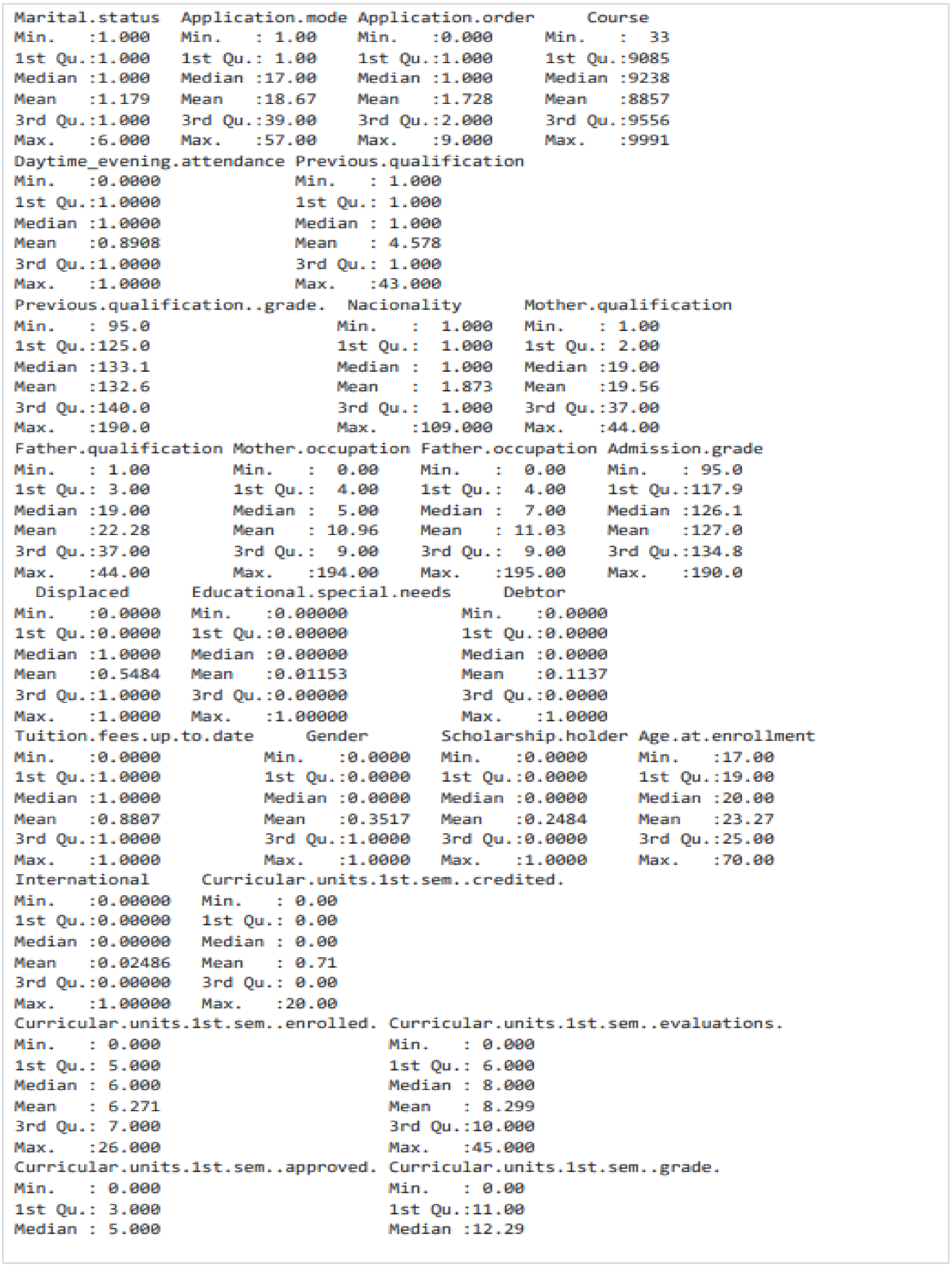
```
'data.frame':	4424 obs. of  37 variables:
 $ Marital.status                              : int  1 1 1 1 2 2 1 1 1 1 ...
 $ Application.mode                            : int  17 15 1 17 39 39 1 18 1 1
...
 $ Application.order                           : int  5 1 5 2 1 1 1 4 3 1 ...
 $ Course                                      : int  171 9254 9070 9773 8014 9
991 9500 9254 9238 9238 ...
 $ Daytime_evening.attendance                  : int  1 1 1 1 0 0 1 1 1 1 ...
 $ Previous.qualification                      : int  1 1 1 1 1 19 1 1 1 1 ...
 $ Previous.qualification..grade.              : num  122 160 122 122 100 ...
 $ Nacionality                                 : int  1 1 1 1 1 1 1 1 62 1 ...
 $ Mother.qualification                        : int  19 1 37 38 37 37 19 37 1
1 ...
 $ Father.qualification                        : int  12 3 37 37 38 37 38 37 1
19 ...
 $ Mother.occupation                           : int  5 3 9 5 9 9 7 9 9 4 ...
 $ Father.occupation                           : int  9 3 9 3 9 7 10 9 9 7 ...
 $ Admission.grade                             : num  127 142 125 120 142 ...
 $ Displaced                                   : int  1 1 1 1 0 0 1 1 0 1 ...
 $ Educational.special.needs                   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ Debtor                                      : int  0 0 0 0 0 1 0 0 0 1 ...
 $ Tuition.fees.up.to.date                     : int  1 0 0 1 1 1 1 0 1 0 ...
 $ Gender                                      : int  1 1 1 0 0 1 0 1 0 0 ...
 $ Scholarship.holder                          : int  0 0 0 0 0 0 1 0 1 0 ...
 $ Age.at.enrollment                           : int  20 19 19 20 45 50 18 22 2
1 18 ...
 $ International                               : int  0 0 0 0 0 0 0 0 1 0 ...
 $ Curricular.units.1st.sem..credited.         : int  0 0 0 0 0 0 0 0 0 0 ...
 $ Curricular.units.1st.sem..enrolled.         : int  0 6 6 6 6 5 7 5 6 6 ...
 $ Curricular.units.1st.sem..evaluations.      : int  0 6 0 8 9 10 9 5 8 9 ...
 $ Curricular.units.1st.sem..approved.         : int  0 6 0 6 5 5 7 0 6 5 ...
 $ Curricular.units.1st.sem..grade.            : num  0 14 0 13.4 12.3 ...
 $ Curricular.units.1st.sem..without.evaluations.: int  0 0 0 0 0 0 0 0 0 0 ...
 $ Curricular.units.2nd.sem..credited.         : int  0 0 0 0 0 0 0 0 0 0 ...
 $ Curricular.units.2nd.sem..enrolled.         : int  0 6 6 6 6 5 8 5 6 6 ...
 $ Curricular.units.2nd.sem..evaluations.      : int  0 6 0 10 6 17 8 5 7 14
...
 $ Curricular.units.2nd.sem..approved.         : int  0 6 0 5 6 5 8 0 6 2 ...
 $ Curricular.units.2nd.sem..grade.            : num  0 13.7 0 12.4 13 ...
 $ Curricular.units.2nd.sem..without.evaluations.: int  0 0 0 0 0 5 0 0 0 0 ...
 $ Unemployment.rate                           : num  10.8 13.9 10.8 9.4 13.9 1
6.2 15.5 15.5 16.2 8.9 ...
 $ Inflation.rate                              : num  1.4 -0.3 1.4 -0.8 -0.3 0.
3 2.8 2.8 0.3 1.4 ...
 $ GDP                                         : num  1.74 0.79 1.74 -3.12 0.79
-0.92 -4.06 -4.06 -0.92 3.51 ...
 $ Target                                      : chr  "Dropout" "Graduate" "Dro
pout" "Graduate" ...
```

To view the structure of the dataset in greater details, the str() function was called on. The code output shows that there are a total of 4424 observations and a total of 37 attributes. The data types for each attribute were indicated as well, where most of them are of integer data types, followed by numerical data types, and lastly those of character types. Information about the data types of each attribute is important because it then determines the necessary data handling techniques that are required. Continuous numerical variables for instance, hints that techniques like normalization or scaling might be required in the subsequent stages.

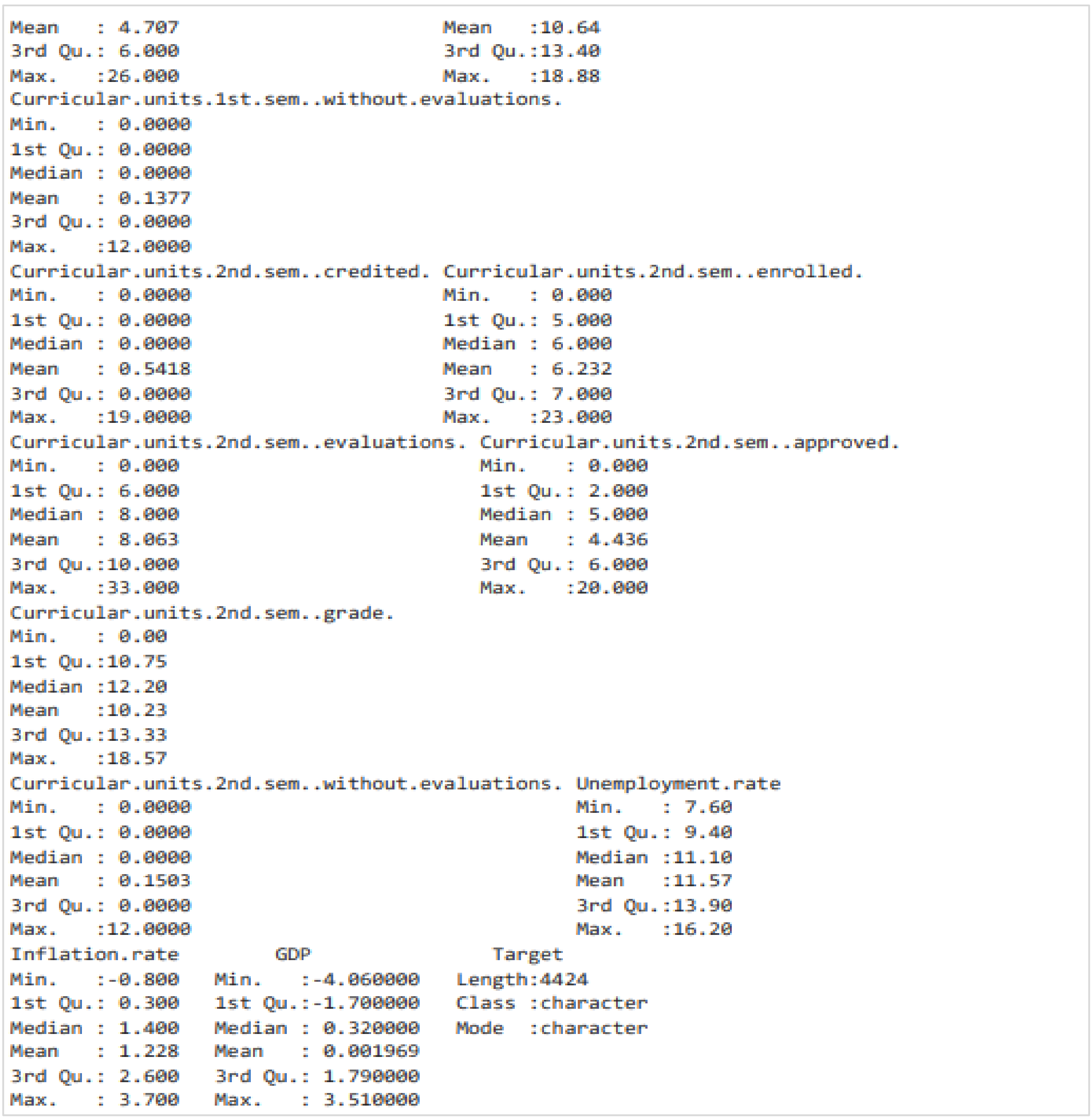
4.1.1.3 Summary of the Dataset



```
# summarize the dataset with statistics for each column
summary(ds)
```



```
 Marital.status  Application.mode Application.order     Course
 Min.   :1.000   Min.   : 1.00    Min.   :0.000     Min.   :  33
 1st Qu.:1.000   1st Qu.: 1.00    1st Qu.:1.000     1st Qu.:9085
 Median :1.000   Median :17.00    Median :1.000     Median :9238
 Mean   :1.179   Mean   :18.67    Mean   :1.728     Mean   :8857
 3rd Qu.:1.000   3rd Qu.:39.00    3rd Qu.:2.000     3rd Qu.:9556
 Max.   :6.000   Max.   :57.00    Max.   :9.000     Max.   :9991
 Daytime_evening.attendance Previous.qualification
 Min.   :0.0000             Min.   : 1.000
 1st Qu.:1.0000             1st Qu.: 1.000
 Median :1.0000             Median : 1.000
 Mean   :0.8908             Mean   : 4.578
 3rd Qu.:1.0000             3rd Qu.: 1.000
 Max.   :1.0000             Max.   :43.000
 Previous.qualification..grade.  Nacionality      Mother.qualification
 Min.   : 95.0                  Min.   :  1.000   Min.   : 1.00
 1st Qu.:125.0                  1st Qu.:  1.000   1st Qu.: 2.00
 Median :133.1                  Median :  1.000   Median :19.00
 Mean   :132.6                  Mean   :  1.873   Mean   :19.56
 3rd Qu.:140.0                  3rd Qu.:  1.000   3rd Qu.:37.00
 Max.   :190.0                  Max.   :109.000   Max.   :44.00
 Father.qualification Mother.occupation Father.occupation Admission.grade
 Min.   : 1.00        Min.   :  0.00    Min.   :  0.00    Min.   : 95.0
 1st Qu.: 3.00        1st Qu.:  4.00    1st Qu.:  4.00    1st Qu.:117.9
 Median :19.00        Median :  5.00    Median :  7.00    Median :126.1
 Mean   :22.28        Mean   : 10.96    Mean   : 11.03    Mean   :127.0
 3rd Qu.:37.00        3rd Qu.:  9.00    3rd Qu.:  9.00    3rd Qu.:134.8
 Max.   :44.00        Max.   :194.00    Max.   :195.00    Max.   :190.0
   Displaced      Educational.special.needs     Debtor
 Min.   :0.0000   Min.   :0.00000          Min.   :0.0000
 1st Qu.:0.0000   1st Qu.:0.00000          1st Qu.:0.0000
 Median :1.0000   Median :0.00000          Median :0.0000
 Mean   :0.5484   Mean   :0.01153          Mean   :0.1137
 3rd Qu.:1.0000   3rd Qu.:0.00000          3rd Qu.:0.0000
 Max.   :1.0000   Max.   :1.00000          Max.   :1.0000
 Tuition.fees.up.to.date     Gender       Scholarship.holder Age.at.enrollment
 Min.   :0.0000          Min.   :0.0000   Min.   :0.0000     Min.   :17.00
 1st Qu.:1.0000          1st Qu.:0.0000   1st Qu.:0.0000     1st Qu.:19.00
 Median :1.0000          Median :0.0000   Median :0.0000     Median :20.00
 Mean   :0.8807          Mean   :0.3517   Mean   :0.2484     Mean   :23.27
 3rd Qu.:1.0000          3rd Qu.:1.0000   3rd Qu.:0.0000     3rd Qu.:25.00
 Max.   :1.0000          Max.   :1.0000   Max.   :1.0000     Max.   :70.00
 International     Curricular.units.1st.sem..credited.
 Min.   :0.00000   Min.   : 0.00
 1st Qu.:0.00000   1st Qu.: 0.00
 Median :0.00000   Median : 0.00
 Mean   :0.02486   Mean   : 0.71
 3rd Qu.:0.00000   3rd Qu.: 0.00
 Max.   :1.00000   Max.   :20.00
 Curricular.units.1st.sem..enrolled. Curricular.units.1st.sem..evaluations.
 Min.   : 0.000                      Min.   : 0.000
 1st Qu.: 5.000                      1st Qu.: 6.000
 Median : 6.000                      Median : 8.000
 Mean   : 6.271                      Mean   : 8.299
 3rd Qu.: 7.000                      3rd Qu.:10.000
 Max.   :26.000                      Max.   :45.000
 Curricular.units.1st.sem..approved. Curricular.units.1st.sem..grade.
 Min.   : 0.000                      Min.   : 0.00
 1st Qu.: 3.000                      1st Qu.:11.00
 Median : 5.000                      Median :12.29
```



```
Mean   : 4.707                      Mean   :10.64
3rd Qu.: 6.000                      3rd Qu.:13.40
Max.   :26.000                      Max.   :18.88
Curricular.units.1st.sem..without.evaluations.
Min.   : 0.0000
1st Qu.: 0.0000
Median : 0.0000
Mean   : 0.1377
3rd Qu.: 0.0000
Max.   :12.0000
Curricular.units.2nd.sem..credited. Curricular.units.2nd.sem..enrolled.
Min.   : 0.0000                     Min.   : 0.000
1st Qu.: 0.0000                     1st Qu.: 5.000
Median : 0.0000                     Median : 6.000
Mean   : 0.5418                     Mean   : 6.232
3rd Qu.: 0.0000                     3rd Qu.: 7.000
Max.   :19.0000                     Max.   :23.000
Curricular.units.2nd.sem..evaluations. Curricular.units.2nd.sem..approved.
Min.   : 0.000                         Min.   : 0.000
1st Qu.: 6.000                         1st Qu.: 2.000
Median : 8.000                         Median : 5.000
Mean   : 8.063                         Mean   : 4.436
3rd Qu.:10.000                         3rd Qu.: 6.000
Max.   :33.000                         Max.   :20.000
Curricular.units.2nd.sem..grade.
Min.   : 0.00
1st Qu.:10.75
Median :12.20
Mean   :10.23
3rd Qu.:13.33
Max.   :18.57
Curricular.units.2nd.sem..without.evaluations. Unemployment.rate
Min.   : 0.0000                                Min.   : 7.60
1st Qu.: 0.0000                                1st Qu.: 9.40
Median : 0.0000                                Median :11.10
Mean   : 0.1503                                Mean   :11.57
3rd Qu.: 0.0000                                3rd Qu.:13.90
Max.   :12.0000                                Max.   :16.20
Inflation.rate        GDP               Target
Min.   :-0.800   Min.   :-4.060000   Length:4424
1st Qu.: 0.300   1st Qu.:-1.700000   Class :character
Median : 1.400   Median : 0.320000   Mode  :character
Mean   : 1.228   Mean   : 0.001969
3rd Qu.: 2.600   3rd Qu.: 1.790000
Max.   : 3.700   Max.   : 3.510000
```

The have a deeper understanding of the dataset, the summary() function was subsequently called on. From the summary code output above, several important observations could be inferred:

1. **Mismatch in Data Types of Attributes:** When making cross-references between the description of each attribute which was provided by the UCI Machine Learning Repository (2021) with the summary output shown above, it was noticed that even though many of the attributes were supposed to treated as a factor, they were however, addressed as being numerical data types instead in the dataset. For instance, the attribute of Gender carried the properties of a categorical data where "0" would indicate that the

student was a female while “1” would indicate that the student was a male instead. Given that, such mismatches in the data type of the attributes needs to be addressed promptly.

2. **Mismatch in Data Type for the Target Variable:** Similar to that of the previous point, notice that the Target variable was treated as a character data type. While it is logically correct, such is however, contextually wrong because in order to facilitate for the effective modelling of the Target variable, the Target variable has to be in a factor data type.
3. **Correct Data Representation:** Macroeconomics attributes, namely Unemployment.rate, Inflation.rate, and GDP were treated as numerical data. Given that, these attributes could be used directly in the subsequent data analysis stages. However, the distributions and potential issue of outliers within these variables would still need to be checked for.
4. **Wide Ranges:** Attributes namely GDP and Admission.grade were found to have had a wide range, essentially provide hints that technique such as normalization or scaling might be needed to address it.

4.1.1.4 Dimension of the Dataset

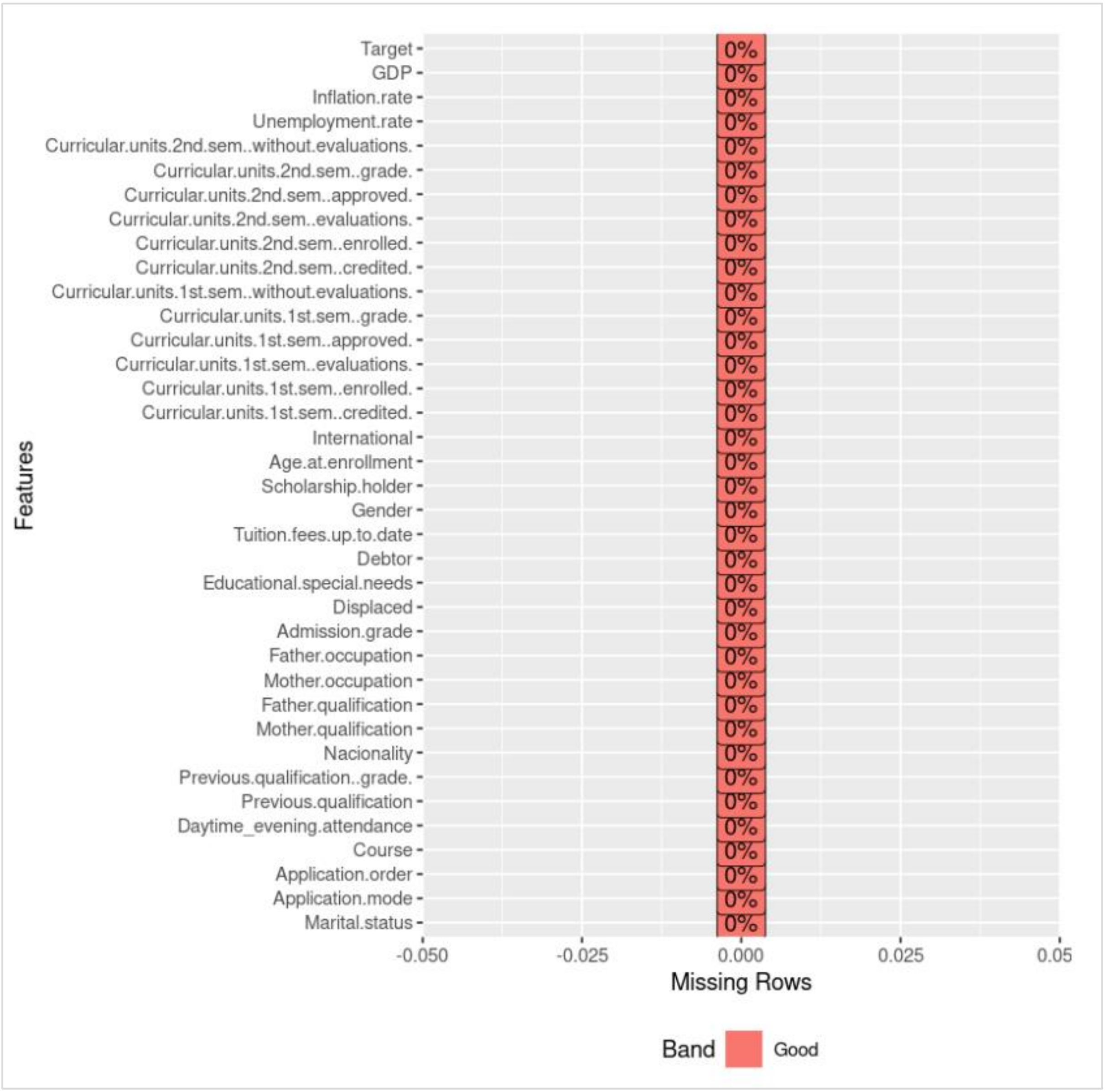
```
# Check for the dimensions of the dataset, namely the count of rows and columns
dim(ds)
```

```
4424 · 37
```

After calling on the dim() function on the dataset, the code output showed that once again, there are a total of 4424 rows and 37 columns. The rows are indicative of the number of observations while the columns are indicative of the number of attributes within the dataset. This information is important because it sets the expectation for the computational resources to be employed as well as the time required for the analysis to be done. Overall, the data is considered to be moderate in size, hence proposing that computational power and resources would not be an issue when analyzing the data.

4.1.1.5 Detection of Missing Values

```
# Graphically represent the presence of missing entries across all columns
plot_missing(ds)
```



The plot above uncovered that there are no missing values present within the dataset. Such scenario is indeed a favored scenario as no treatment of missing values needs to be carried out. Consequently, the data preprocessing steps become more straightforward.

4.1.1.6 Detection of Distinct Entries in Each Attribute

```
# Determine the count of distinct entries in each column
sapply(ds, function(x) length(unique(x)))
```

```
Marital.status:        6 Application.mode:        18 Application.order:        8 Course:        17 Daytime_evening.attendance:        2 Previous.qualification:        17
 Previous.qualification..grade.:        101 Nacionality:        21 Mother.qualification:        29 Father.qualification:        34 Mother.occupation:        32
 Father.occupation:        46 Admission.grade:        620 Displaced:        2 Educational.special.needs:        2 Debtor:        2 Tuition.fees.up.to.date:        2
 Gender:        2 Scholarship.holder:        2 Age.at.enrollment:        46 International:        2 Curricular.units.1st.sem..credited.:        21
 Curricular.units.1st.sem..enrolled.:        23 Curricular.units.1st.sem..evaluations.:        35 Curricular.units.1st.sem..approved.:        23
 Curricular.units.1st.sem..grade.:        797 Curricular.units.1st.sem..without.evaluations.:        11 Curricular.units.2nd.sem..credited.:        19
 Curricular.units.2nd.sem..enrolled.:        22 Curricular.units.2nd.sem..evaluations.:        30 Curricular.units.2nd.sem..approved.:        20
 Curricular.units.2nd.sem..grade.:        782 Curricular.units.2nd.sem..without.evaluations.:        10 Unemployment.rate:        10 Inflation.rate:        9 GDP:        10
 Target:        3
```

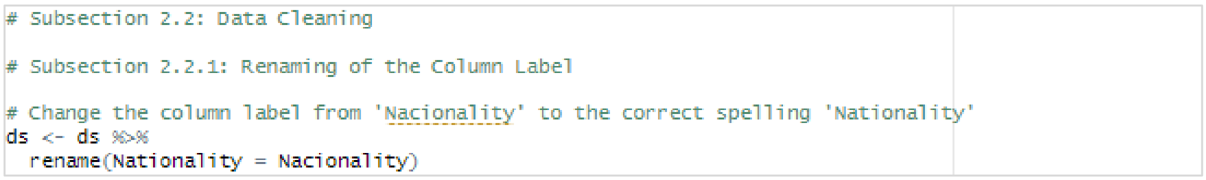
The code output above revealed that all of the attributes within the dataset do indeed contain distinct entries, but with varying degree. The observations could be summarized as follows, where the number of distinct entries is further broken down into three groups namely, Minimal, Moderate, and Substantial.

Group	Attributes Belonging to the Group	Remark
Minimal	▪ Daytime_evening.attendance ▪ Displaced ▪ Educational.special.needs ▪ Debtor ▪ Tuition.fees.up.to.date ▪ Gender ▪ Scholarship.holder ▪ International	All of the attributes listed under this group contained only 2 unique values. This strongly suggests that they are binary variables.
Moderate	▪ Marital.status ▪ Application.mode ▪ Application.order ▪ Course ▪ Previous.qualification ▪ Nacionality ▪ Mother.qualification ▪ Father.qualification ▪ Mother.occupation ▪ Father.occupation ▪ Age.at.enrolment ▪ Unemployment.rate ▪ Inflation.rate ▪ GDP ▪ Target	All attributes listed under this group contained a moderate number of distinct entries, hinting that they might be of categorical data types or that of discrete numerical data types.
Substantial	▪ Previous.qualification..grade. ▪ Admission.grade ▪ Curricular.units.1st.sem..grade ▪ Curricular.units.2nd.sem.grade.	All attributes listed under this group contained a substantial number of unique entries , suggesting that they could be of a continuous numerical data type.

Given the above findings, several decisions could then be made moving forward, based on the varying degree of distinct entries. The possibly decisions to be made are as outlined below:

1. **Potential for Feature Engineering:** Considerations to bin the observations of attributes with substantial number of distinct entries could ensure that modelling and interpretation are simplified and made straightforward.
2. **Choice of Encoding Techniques:** For when deciding on which encoding techniques to employ, the rule of thumb is to perform one-hot encoding for attributes with minimal distinct entries, while target encoding or other more sophisticated approaches are used for when attributes are heavily plagued by distinct entries.
3. **Detection of Irregularities in Data:** The number of distinct entries could serve as an intuition to the presence of outliers or any anomalies, bearing in mind that it might not always be the case, hence warranting for further investigations.

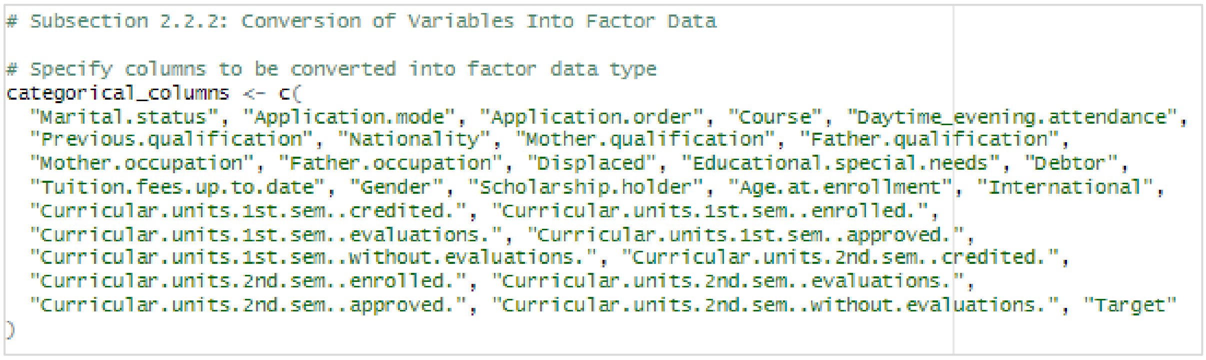
4.1.2 Renaming of the Column Label



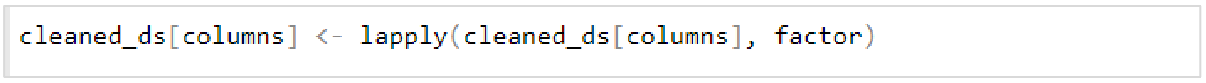
```r
# Subsection 2.2: Data Cleaning

# Subsection 2.2.1: Renaming of the Column Label

# Change the column label from 'Nacionality' to the correct spelling 'Nationality'
ds <- ds %>%
  rename(Nationality = Nacionality)
```

When performing data exploration at the initial stage, it was noticed that the label name specifically for the Nationality attribute was misspelled as “Nacionality”. Following such observation, corrections were made to the label name of the said attribute.

4.1.3 Conversion of Variables into Factor Data



```r
# Subsection 2.2.2: Conversion of Variables Into Factor Data

# Specify columns to be converted into factor data type
categorical_columns <- c(
  "Marital.status", "Application.mode", "Application.order", "Course", "Daytime_evening.attendance",
  "Previous.qualification", "Nationality", "Mother.qualification", "Father.qualification",
  "Mother.occupation", "Father.occupation", "Displaced", "Educational.special.needs", "Debtor",
  "Tuition.fees.up.to.date", "Gender", "Scholarship.holder", "Age.at.enrollment", "International",
  "Curricular.units.1st.sem..credited.", "Curricular.units.1st.sem..enrolled.",
  "Curricular.units.1st.sem..evaluations.", "Curricular.units.1st.sem..approved.",
  "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..without.evaluations.", "Target"
)
```

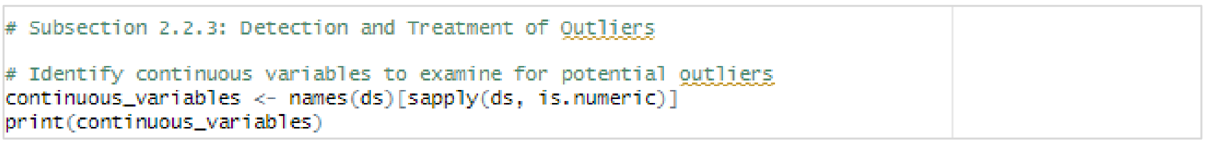


```r
cleaned_ds[columns] <- lapply(cleaned_ds[columns], factor)
```

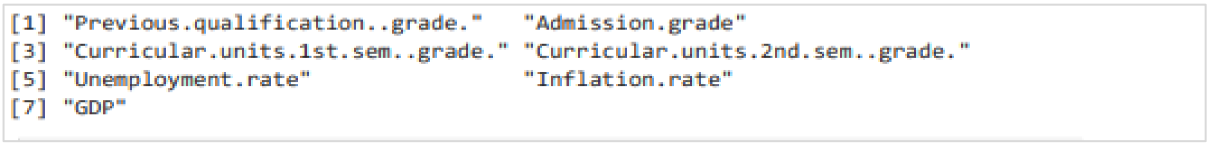
As previously mentioned in the earlier section, they were a considerable number of attributes whose data types were mismatch with their supposedly data types. Hence, to ensure for effective statistical modelling in the RStudio’s IDE, all problematic attributes were then converted into that of factor data types.

4.1.4 Detection and Treatment of Outliers

Given that outliers would susbtantially distort the findings of data analysis and statistical modellings, it is thus imperative to check for their presences.

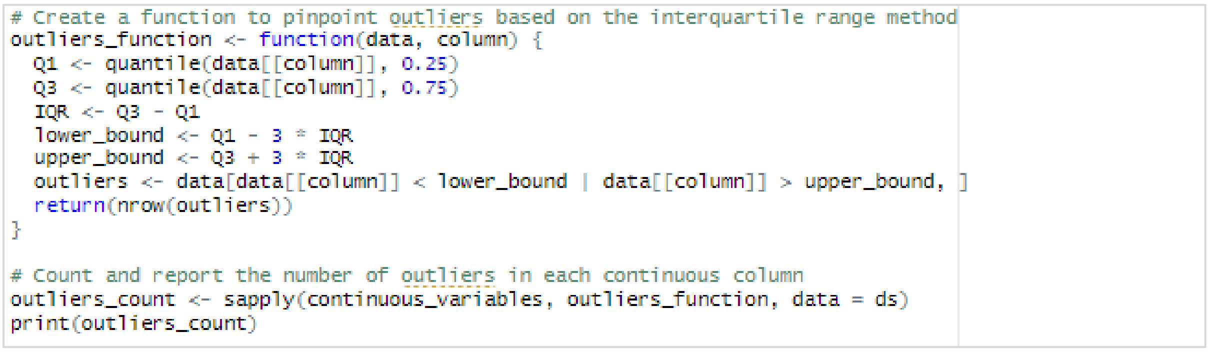


```
# Subsection 2.2.3: Detection and Treatment of Outliers

# Identify continuous variables to examine for potential outliers
continuous_variables <- names(ds)[sapply(ds, is.numeric)]
print(continuous_variables)
```

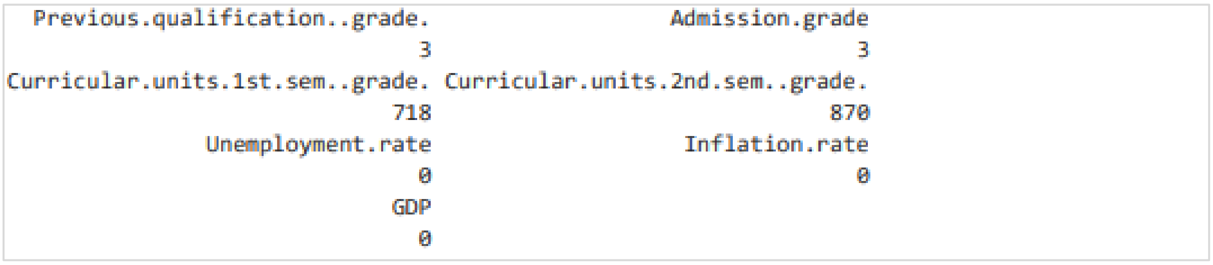


```
[1] "Previous.qualification..grade."   "Admission.grade"
[3] "Curricular.units.1st.sem..grade." "Curricular.units.2nd.sem..grade."
[5] "Unemployment.rate"                "Inflation.rate"
[7] "GDP"
```

In order to perform the detection of outliers within the dataset, the first step is to identify all of the continous variables of the dataset. The code output above shows that there are a total of 7 attributes which are continuous in nature.

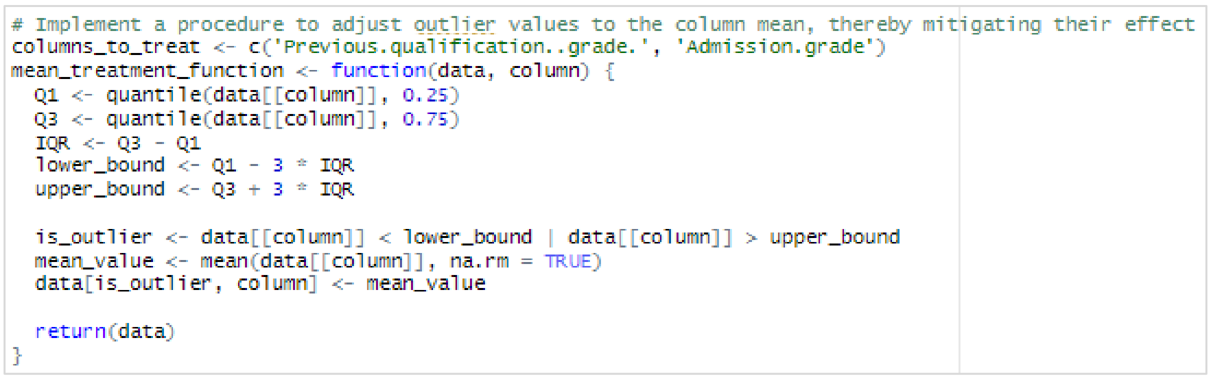


```
# Create a function to pinpoint outliers based on the interquartile range method
outliers_function <- function(data, column) {
  Q1 <- quantile(data[[column]], 0.25)
  Q3 <- quantile(data[[column]], 0.75)
  IQR <- Q3 - Q1
  lower_bound <- Q1 - 3 * IQR
  upper_bound <- Q3 + 3 * IQR
  outliers <- data[data[[column]] < lower_bound | data[[column]] > upper_bound, ]
  return(nrow(outliers))
}

# Count and report the number of outliers in each continuous column
outliers_count <- sapply(continuous_variables, outliers_function, data = ds)
print(outliers_count)
```

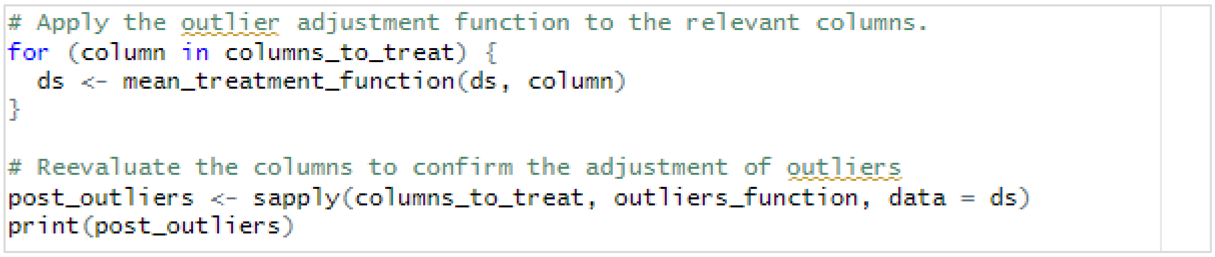


```
  Previous.qualification..grade.                  Admission.grade
                               3                                3
Curricular.units.1st.sem..grade. Curricular.units.2nd.sem..grade.
                             718                              870
               Unemployment.rate                   Inflation.rate
                               0                                0
                             GDP
                               0
```

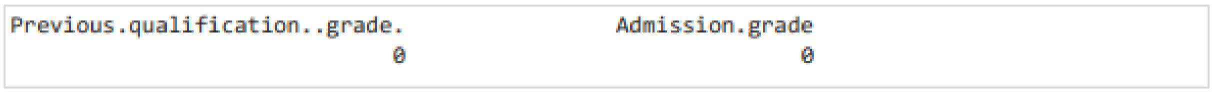
Following the identification of all continuous variables, a function called "outliers_function" was then created to pinpoint the outliers should there be any, based on the Interquartile Range (IQR) approach. That is, according to the IQR approach, any values that falls below the lower bound or that falls above the upper bound would then be considered as an outlier. The IQR multiplier was set at 3 instead of the conventional 1.5 due to a more conservative approach that this study takes. Once the function has been set up, the function was called on using the sapply() function where the output of the code showed that there is indeed a significant number of outliers contained within the dataset.



```
# Implement a procedure to adjust outlier values to the column mean, thereby mitigating their effect
columns_to_treat <- c('Previous.qualification..grade.', 'Admission.grade')
mean_treatment_function <- function(data, column) {
  Q1 <- quantile(data[[column]], 0.25)
  Q3 <- quantile(data[[column]], 0.75)
  IQR <- Q3 - Q1
  lower_bound <- Q1 - 3 * IQR
  upper_bound <- Q3 + 3 * IQR

  is_outlier <- data[[column]] < lower_bound | data[[column]] > upper_bound
  mean_value <- mean(data[[column]], na.rm = TRUE)
  data[is_outlier, column] <- mean_value

  return(data)
}
```



```
# Apply the outlier adjustment function to the relevant columns.
for (column in columns_to_treat) {
  ds <- mean_treatment_function(ds, column)
}

# Reevaluate the columns to confirm the adjustment of outliers
post_outliers <- sapply(columns_to_treat, outliers_function, data = ds)
print(post_outliers)
```

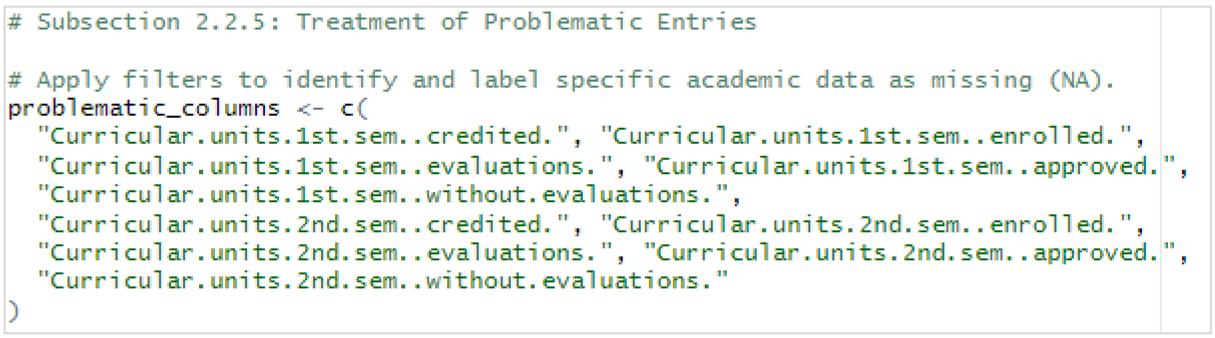


```
Previous.qualification..grade.                 Admission.grade
                             0                               0
```

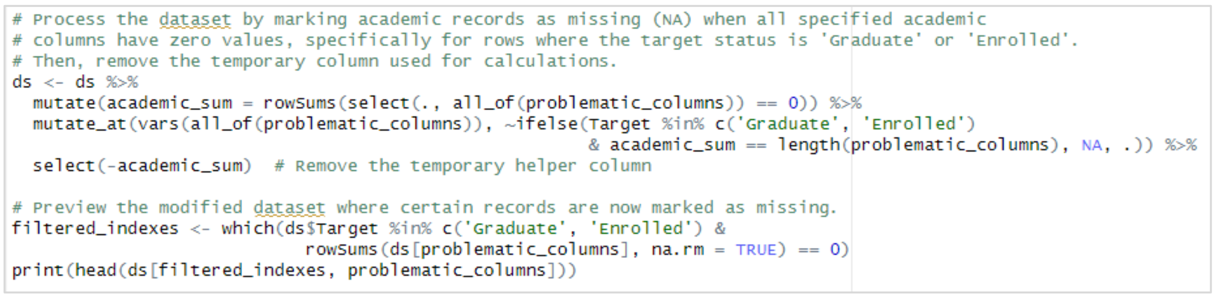
After having identified the presence of the outliers, another function called the "mean_treatment _function" was created to implement the adjustment to the outlier values using the mean values of the respective attributes. Even though outliers were detected in 4 attributes namely Previous.qualification..grade., Admission.grade, Curricular.units.1st.sem..grade., and Curricular.units.2nd.sem..grade, it is important to note that only the first two columns mentioned will be treated for outliers. This is so because, upon further investigation, it was noticed that the outliers detected for both attributes were predominantly 0 values. Considering that the number of these 0 values are not minimal but are considerably large instead, it is only then intuitive to retain them and consider them as acceptable for the time being. Further discussion on this matter would be addressed in the immediate next sections to come. Once the function has been successfully called on to be applied to the relevant columns, the columns affected were then reevaluated once again to confirm if the adjustments were successful. The code output shows that there are no only any outliers contain within the attributes of Previous.qualification..grade. and Admission.grade.

4.1.5 Treatment of Problematic Entries

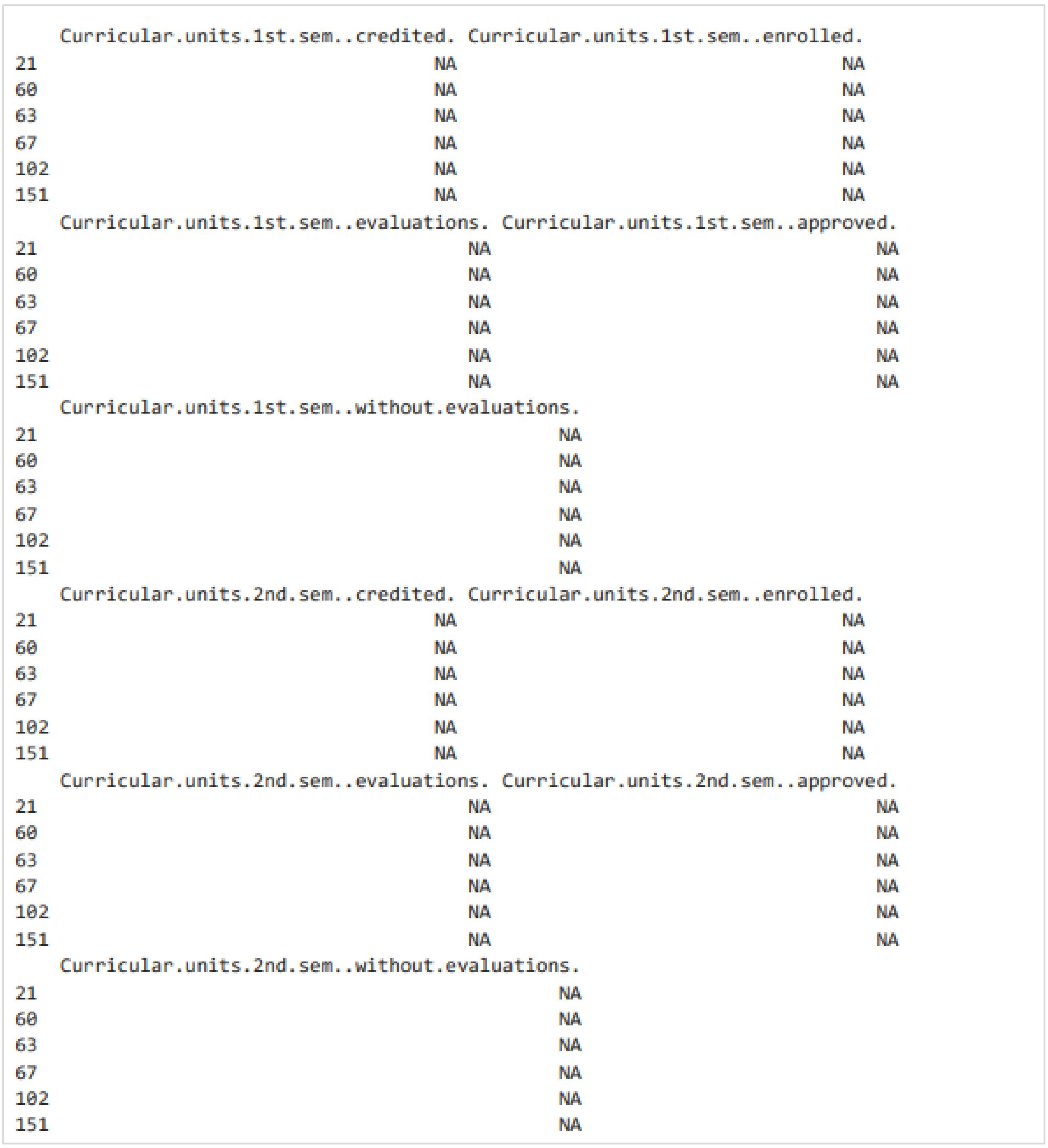
Resuming the discussion on the exclusion of outliers detection for the attributes of Curricular.units.1st.sem..grade., and Curricular.units.2nd.sem..grade., further investigation reported that for observations with 0 values for both of the aforementioned attributes, they too had 0 values in other semester-academic-related attributes. These attributes were stored under the problematic_columns variable, as could be observed in the source code down below. Such occurrences were deemed as illogical because it does not make sense for students to have 0 units enrolled for the entirety of the two semesters, and yet was still able to be graduate or have the enrolled status. Put simply, if students were to have no units enrolled for the entire 2 semesters, these students would reasonably be speaking, be behind their peers, essentially causing them to not be able to finish their course in the set course duration. Ultimately, these students would then be categorized as "Dropouts" instead of what was recorded as "Enrolled" or "Graduate". Following this direction of the argument, the zero values entered is then very likely that there are acting as a representation of the phenomenon of missing values across these student records. Therefore, treatments of these problematic entries will be carried out and they are detailed as follows.



```
# Subsection 2.2.5: Treatment of Problematic Entries

# Apply filters to identify and label specific academic data as missing (NA).
problematic_columns <- c(
  "Curricular.units.1st.sem..credited.", "Curricular.units.1st.sem..enrolled.",
  "Curricular.units.1st.sem..evaluations.", "Curricular.units.1st.sem..approved.",
  "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..without.evaluations."
)
```



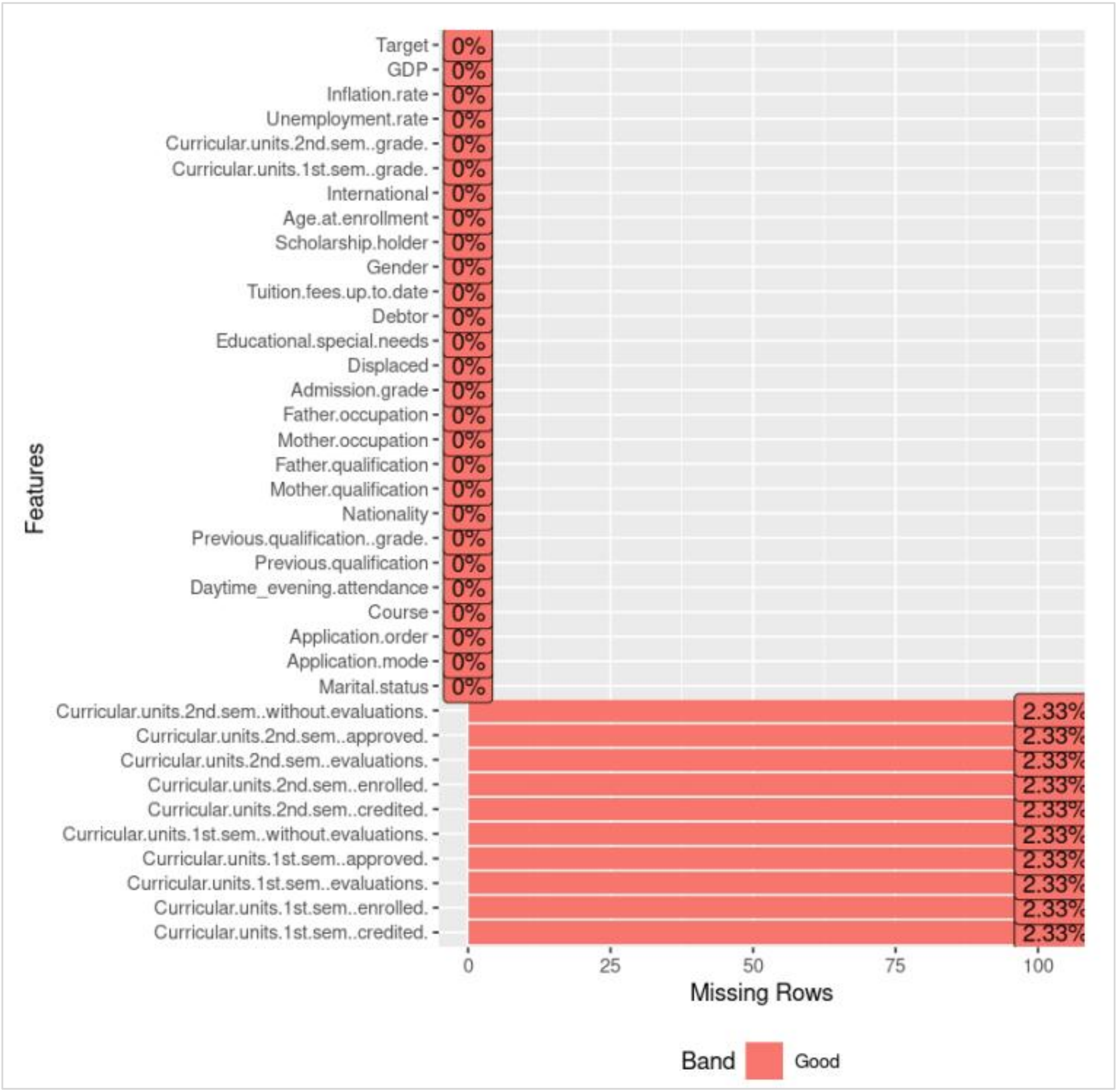
```
# Process the dataset by marking academic records as missing (NA) when all specified academic
# columns have zero values, specifically for rows where the target status is 'Graduate' or 'Enrolled'.
# Then, remove the temporary column used for calculations.
ds <- ds %>%
  mutate(academic_sum = rowSums(select(., all_of(problematic_columns)) == 0)) %>%
  mutate_at(vars(all_of(problematic_columns)), ~ifelse(Target %in% c('Graduate', 'Enrolled')
                                                        & academic_sum == length(problematic_columns), NA, .)) %>%
  select(-academic_sum)  # Remove the temporary helper column

# Preview the modified dataset where certain records are now marked as missing.
filtered_indexes <- which(ds$Target %in% c('Graduate', 'Enrolled') &
                            rowSums(ds[problematic_columns], na.rm = TRUE) == 0)
print(head(ds[filtered_indexes, problematic_columns]))
```



```
    Curricular.units.1st.sem..credited. Curricular.units.1st.sem..enrolled.
21                                   NA                                  NA
60                                   NA                                  NA
63                                   NA                                  NA
67                                   NA                                  NA
102                                  NA                                  NA
151                                  NA                                  NA
    Curricular.units.1st.sem..evaluations. Curricular.units.1st.sem..approved.
21                                      NA                                  NA
60                                      NA                                  NA
63                                      NA                                  NA
67                                      NA                                  NA
102                                     NA                                  NA
151                                     NA                                  NA
    Curricular.units.1st.sem..without.evaluations.
21                                              NA
60                                              NA
63                                              NA
67                                              NA
102                                             NA
151                                             NA
    Curricular.units.2nd.sem..credited. Curricular.units.2nd.sem..enrolled.
21                                   NA                                  NA
60                                   NA                                  NA
63                                   NA                                  NA
67                                   NA                                  NA
102                                  NA                                  NA
151                                  NA                                  NA
    Curricular.units.2nd.sem..evaluations. Curricular.units.2nd.sem..approved.
21                                      NA                                  NA
60                                      NA                                  NA
63                                      NA                                  NA
67                                      NA                                  NA
102                                     NA                                  NA
151                                     NA                                  NA
    Curricular.units.2nd.sem..without.evaluations.
21                                              NA
60                                              NA
63                                              NA
67                                              NA
102                                             NA
151                                             NA
```

Treatment of these problematic attributes began by marking those academic records as missing (NA) when all specified academic columns have had 0 value, and more so for entries where the Target class is either “Graduate” or “Enrolled”. Once the conversion of “0” values to NA values is completed, the modified dataset is then printed out to check if the conversion was successfully. The code output above shows that the conversion was indeed successful.

```
# Visualize missing data across all columns of the dataset.
plot_missing(ds)
```



A plot showing the missing values was also created to reflect that the conversion was indeed successful.

```
# Implement missForest for imputing the missing values identified
seed <- 123
set.seed(seed)
ds_cleaned <- missForest(ds, maxiter = 10, ntree = 100, variablewise = TRUE, decreasing = TRUE)
ds_cleaned <- missForest(ds)$ximp
```

Once the conversion is completed, the last set in the treatment of these problematic attributes is to call on the missForest package for the purpose of performing imputation on these now missing values.

```
# Save the dataset with imputed values as a new file
write.csv(ds_cleaned, file = "cleaned_data.csv", row.names = FALSE)

# Reload the dataset with the imputed values
cleaned_ds <- read.csv("cleaned_data.csv")
```

Once the imputation is successful, the dataset is then saved as a new file titled "cleaned_data" and the dataset is then reloaded into the IDE of RStudio for the subsequent processes within this study.

4.2 Exploratory Data Analysis (EDA)

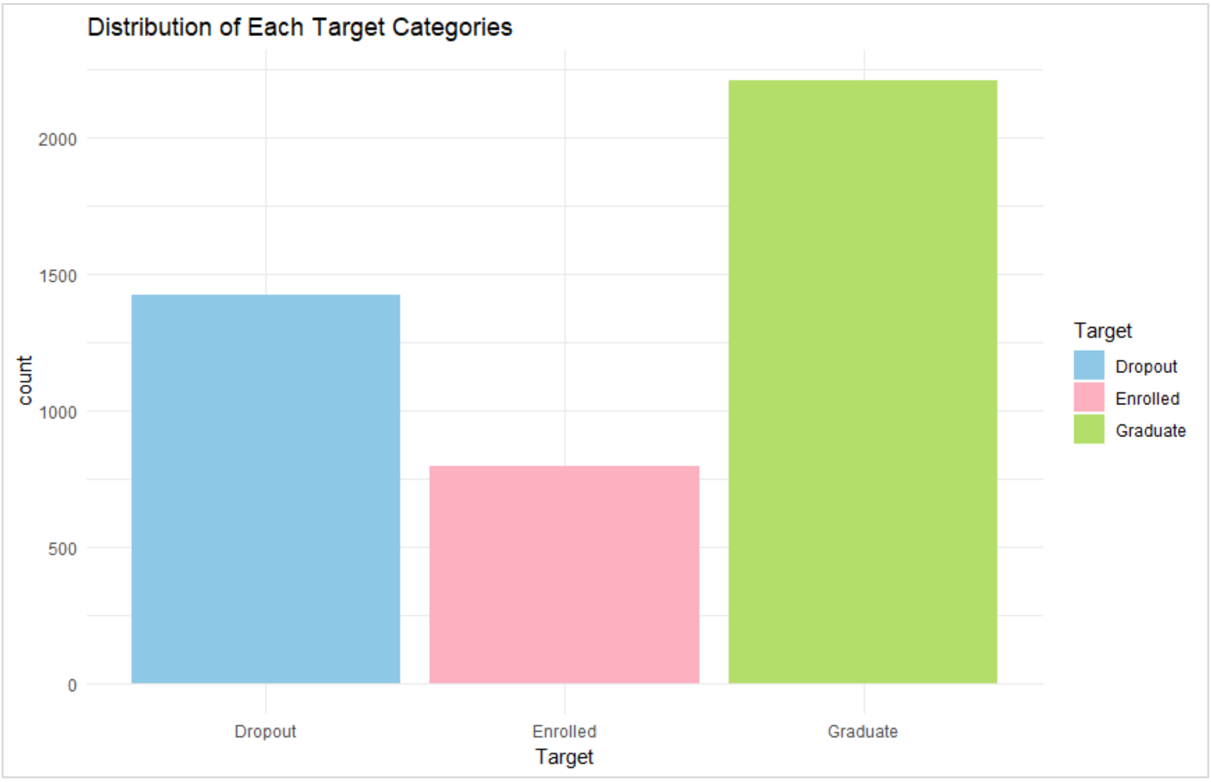
4.2.1 Analysis on the Target Column

The purpose of conducting an analysis on the Target attribute is to ensure that the respective distribution of the three Target classes is being thoroughly explored. This is especially important because should there exist any imbalances within the classes of the Target attribute, balancing techniques such as SMOTE, might then need to be employed to balance the classes.

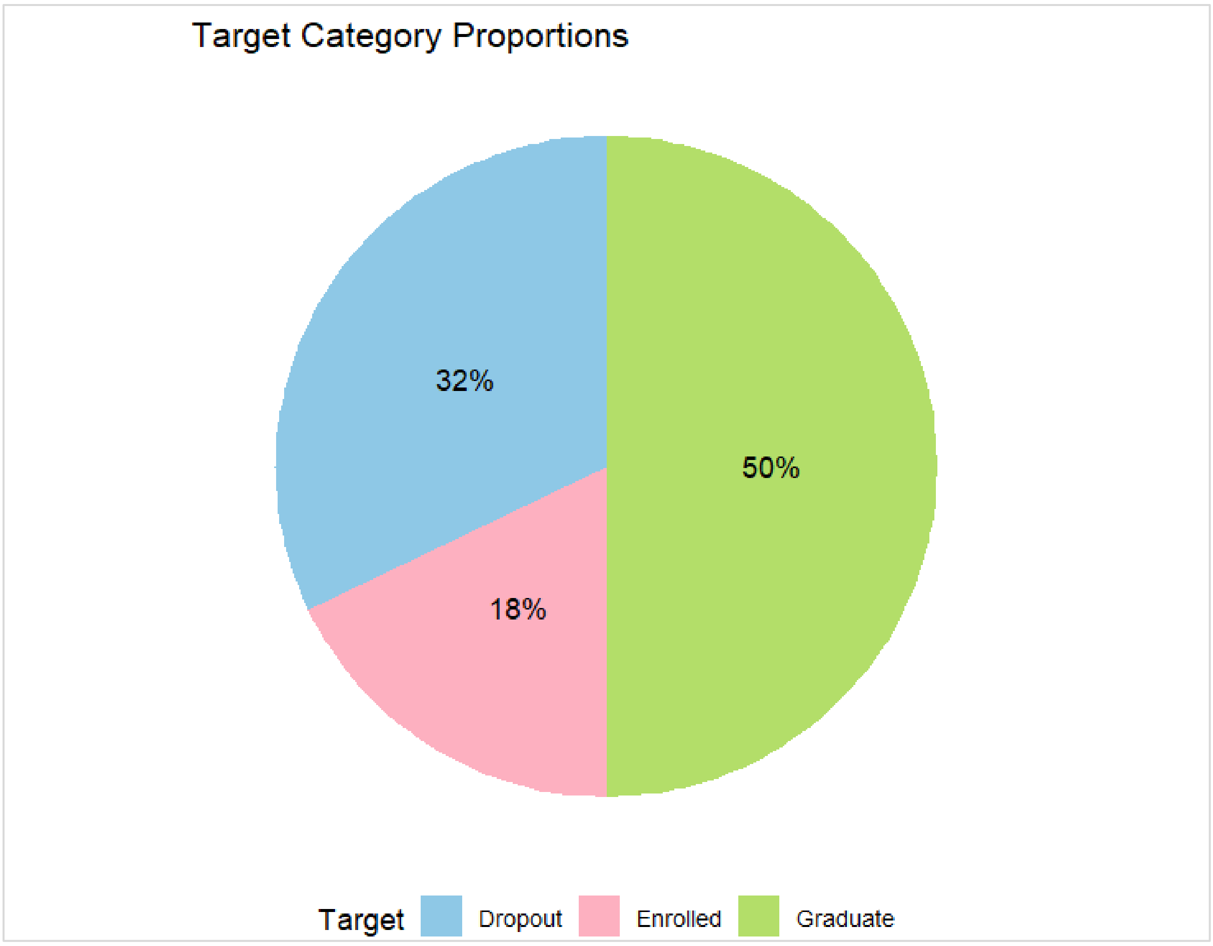
```
# Section 3.0: Exploratory Data Analysis (EDA)

# Subsection 3.1: Examining the Target Column

# Count and display the frequency of each target category
table(cleaned_ds$Target)
```

```
Dropout Enrolled Graduate
   1421      794     2209
```

```
# Visual representation of target category distribution using a bar chart with eye-friendly colors
ggplot(cleaned_ds, aes(x = Target, fill = Target)) +
  geom_bar(stat = "count") +
  labs(title = "Distribution of Each Target Categories") +
  scale_fill_manual(values = c("#8EC8E6", "#FDB0C0", "#B3DE69")) + # Soft blue, soft pink, soft green
  theme_minimal()
```



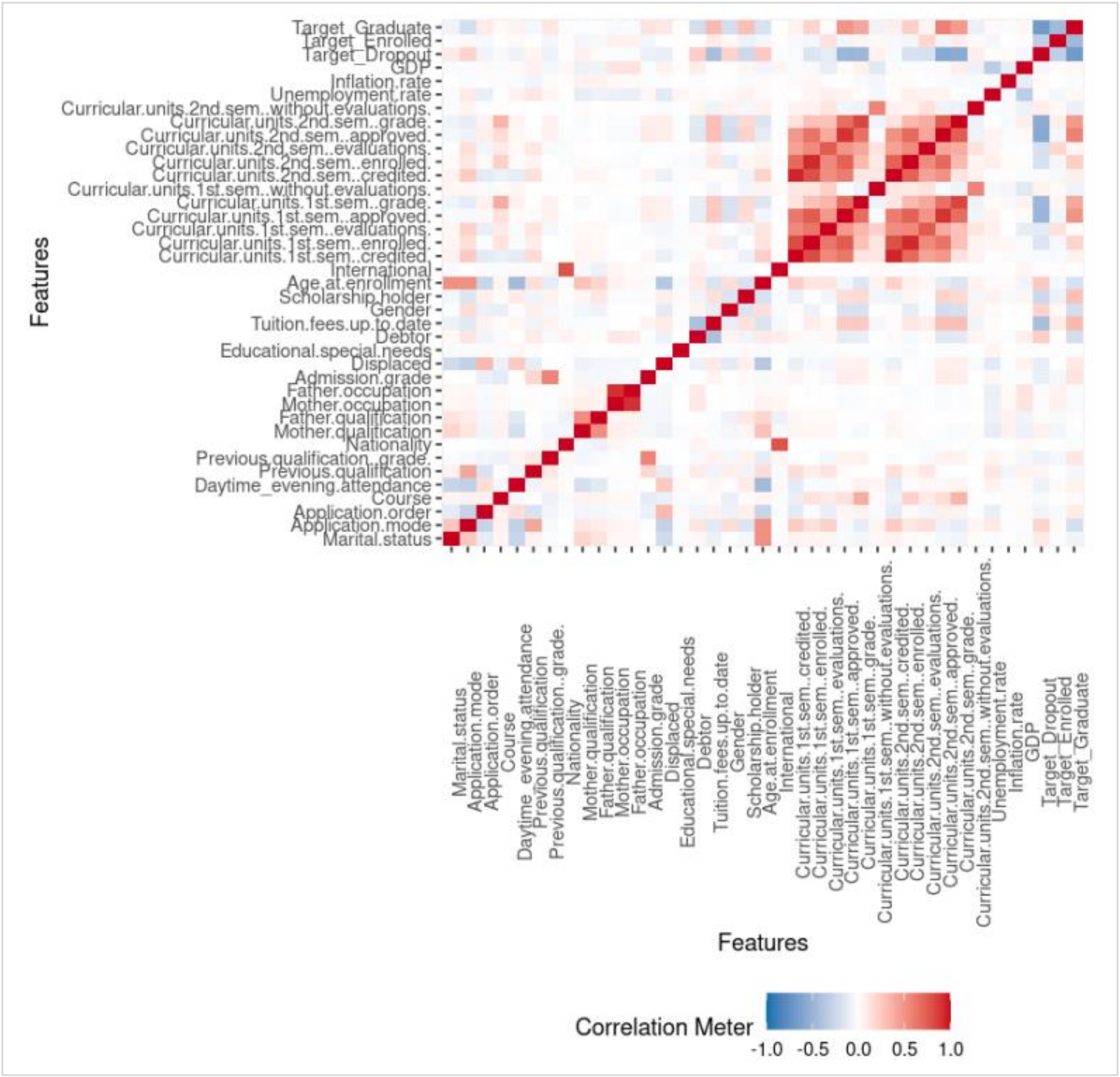
```r
# Represent target category proportions using a pie chart.
ggplot(cleaned_ds, aes(x = "", fill = Target)) +
  geom_bar(width = 1, stat = "count") +
  coord_polar("y", start = 0) +
  labs(title = "Target Category Proportions") +
  scale_fill_manual(values = c("#8EC8E6", "#FDB0C0", "#B3DE69")) +
  geom_text(aes(label = scales::percent(..count.. / sum(..count..))),
            stat = "count", position = position_stack(vjust = 0.5)) +
  theme_void() +
  theme(legend.position = "bottom")
```



From the code outputs above, it is observed that even though there are differences between the classes of the Target attribute, these differences are however, not large enough to warrant for any balancing techniques. The rule of thumb is the as long as the differences is not equal or above an 80/20 proportion, where 80 refers to the majority class and 20 refers to the minority class, the dataset will still be considered as a balanced dataset. From the pie chart shown above, the majority class was only 50% of the entire dataset, essentially making it not fulfill the rule of thumb. Put simply, data from the dataset of this study is balanced.

4.2.2 Analysis using the Correlation Matrix

```
# Subsection 3.2: Exploring Correlations in Data

# Generate a matrix to visualize the correlations between different variables in the dataset
plot_correlation(cleaned_ds)
```



The correlation heatmap shown below was employed to visually illustrate the correlation values between pairs of attributes within the dataset, where the intensities of the colors represent the strength while the shades of the colors indicate the direction of the relationship. The heatmap is important in providing a quick review of the relationships between attributes, essentially aiding in the identification of possible patterns within the dataset. That said, several observations could be derived from the heatmap above, and these observations are outlined as follows:

1. **Target and Other Attributes:** The Target variable (Target_Dropout, Target_Enrolled, and Target_Graduate) exhibited varied extents of correlation with the rest of the attributes. It is worth noting however that, for attributes that are strongly correlated with the three classes of the target variable, suggestion is that these attributes might be indicative of the academic outcomes of students.
2. **Parental Attributes:** It is worth noting that attributes precisely the occupation of both parents together with the qualifications of both mother and father displayed very minimal correlations with the other attributes, and also with the classes of the Target variable.
3. **Factors Relating to the Application Component:** Attributes namely the Application.order and Application.mode both showed a rather varied degree of correlation with the other attributes too. This suggests that there is potentially some role in which the application process plays in the prediction of student's academic outcomes.
4. **Interrelatedness between the Curricular Units:** Notice that among the attributes which pertained to the curricular aspect of the data, correlation values are high, more so when the curricular components are of the same semester. In terms of the correlation of the curricular attributes with that of the classes of the target variables, certain attributes namely the number of curricular units with evaluations and the number of curricular units approved for both semester each showed a relatively strong correlation. This warrants for further investigation into the predictive power that these variables have on the classes of the target variable.

While strong correlations between the independent and dependent variables are highly sought after, it is important to note that such is not the case for when examining the correlations between independent variables. This is because if correlations between the independent variables are high, the issue of multicollinearity is thereby said to be present within the dataset. While some models are robust to the issue of multicollinearity, it is important to acknowledge that there are certain models who are however, especially sensitive to multicollinearity, for instance, Logistic Regression models and Linear Regression models. Given that, the need to understand the relationship between the attributes of the dataset stems not only for the purpose of model interpretation, but at the same time, to inform the data handling approaches so that data is made aligned with the assumptions and requirements of the chosen models. Approaches to handle multicollinearity include feature reduction using techniques such as variable selection or Principal Component Analysis (PCA), or by applying regularization techniques (Lasso or

Ridge Regression) to penalize and make adjustments to the independent variables which are highly correlated with one another.

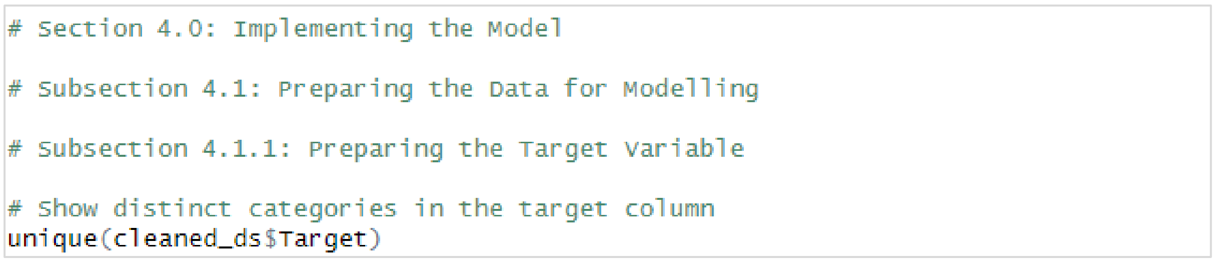
5.0 Implementing the Model

In this section, discussion on the preparation of the data for modelling and the implementation of the models would be outlined.

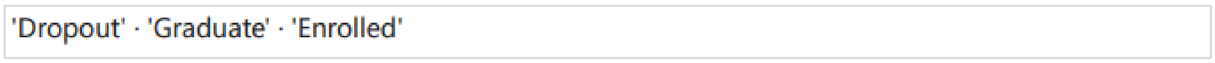
5.1 Preparing the Data for Modelling

5.1.1 Preparing the Target Variable

Prior to carrying out the implementation of the three chosen models, the dataset has to be further prepared for that implementation. That is to say, even though the dataset has already completed the cleaning process, further preparation is necessary because the cleaned dataset still needs to be split into separate training and test sets. The rationale behind the split is so that the models is able to be trained on one subset of the data and subsequently be validated using another subset of the data. In essence, this provides for the evaluation of the models generalizability when dealing with new and unseen data. Overfitting could also be mitigated through this approach. Apart for the said rationales, preparing the data for model implementation could also include processes such as feature scaling or normalization, all with the purpose of ensuring that the dataset does meet the specified input requirements of the chosen models.

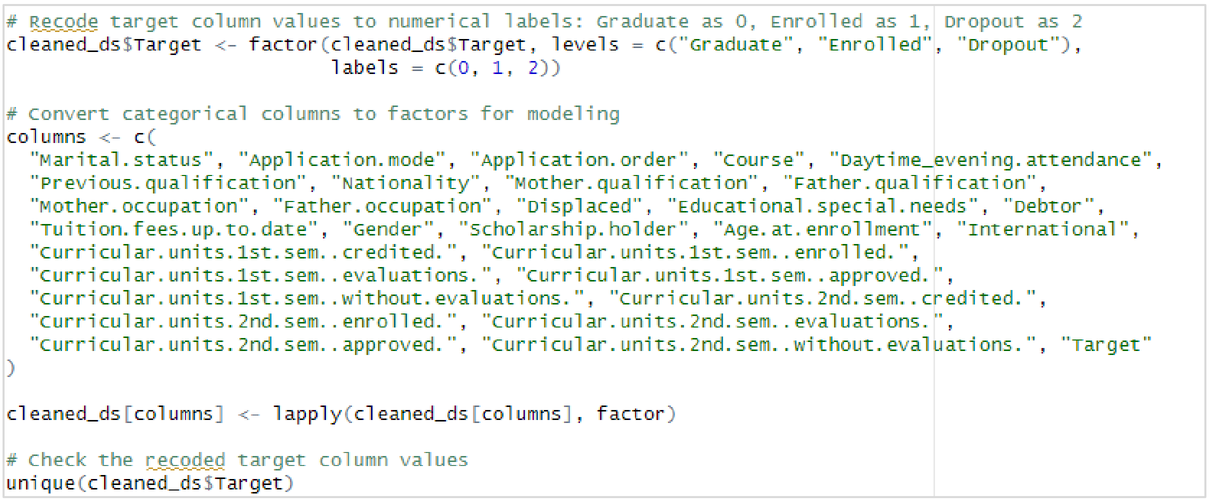


```
# Section 4.0: Implementing the Model

# Subsection 4.1: Preparing the Data for Modelling

# Subsection 4.1.1: Preparing the Target Variable

# Show distinct categories in the target column
unique(cleaned_ds$Target)
```

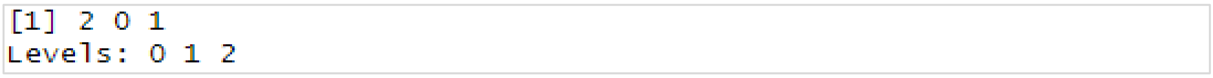


```
'Dropout' · 'Graduate' · 'Enrolled'
```

Before further endeavoring further into the data preparation process, the distinct categories in the target column are being examined. The code output showed that there are three classes namely “Dropout”, “Graduate” and “Enrolled”.



```
# Recode target column values to numerical labels: Graduate as 0, Enrolled as 1, Dropout as 2
cleaned_ds$Target <- factor(cleaned_ds$Target, levels = c("Graduate", "Enrolled", "Dropout"),
                            labels = c(0, 1, 2))

# Convert categorical columns to factors for modeling
columns <- c(
  "Marital.status", "Application.mode", "Application.order", "Course", "Daytime_evening.attendance",
  "Previous.qualification", "Nationality", "Mother.qualification", "Father.qualification",
  "Mother.occupation", "Father.occupation", "Displaced", "Educational.special.needs", "Debtor",
  "Tuition.fees.up.to.date", "Gender", "Scholarship.holder", "Age.at.enrollment", "International",
  "Curricular.units.1st.sem..credited.", "Curricular.units.1st.sem..enrolled.",
  "Curricular.units.1st.sem..evaluations.", "Curricular.units.1st.sem..approved.",
  "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..without.evaluations.", "Target"
)

cleaned_ds[columns] <- lapply(cleaned_ds[columns], factor)

# Check the recoded target column values
unique(cleaned_ds$Target)
```



```
[1] 2 0 1
Levels: 0 1 2
```

Once the distinct categories of the target variable have been identified, the categories are then recoded to take on numerical labels where the “Graduate” class is denoted by “0”, the “Enrolled” class is denoted by “1”, and the “Dropout” class is denoted by 2. The purpose of making such recoding of the target classes is because and recall that XGBoost is actually one out of the three models chosen for this study. The recoding was thereby necessary because it is common practice to make sure that the classes of the target variable have an index starting from 0. In other words, a zero-indexing format is convention and expectation required by the XGBoost model. Following this rationale, and since the other two models namely Random Forest and Decision Tree model are flexible in terms of handling numerical labels, whether they start from 0 or 1, the initially recoded version implemented here would then be made to persist.

After performing the recoding, and recall that the dataset does contain several categorical attributes and that these variables are not treated as their rightful categories, these attributes are thereby converted into factor data type, as could be seen in the source code above. The attributes to be converted are stored within the *columns* variable. As previously mentioned in the earlier section, the changing of the data types of attributes into their rightful type is necessary to ensure that these attributes would be accounted for accurately when modelling is done. Once the conversion has been completed, the target column values are called once again to check if the conversion has been reflected.

5.1.2 Normalizing Numerical Attributes

```
# Subsection 4.1.2: Standardizing Numerical Data

# Identify and list columns with numeric data
numeric_columns <- names(cleaned_ds)[sapply(cleaned_ds, is.numeric)]

# Standardize these columns for uniform scaling
cleaned_ds[numeric_columns] <- lapply(cleaned_ds[numeric_columns], scale)

# Overview of standardized data
summary(cleaned_ds[numeric_columns])
```

```
Previous.qualification..grade..V1 Admission.grade.V1
Min.   :-2.867157                 Min.   :-2.219488
1st Qu.:-0.578001                 1st Qu.:-0.627953
Median : 0.040072                 Median :-0.058059
Mean   : 0.000000                 Mean   : 0.000000
3rd Qu.: 0.566578                 3rd Qu.: 0.546586
Max.   : 3.954530                 Max.   : 3.993753
Curricular.units.1st.sem..grade..V1 Curricular.units.2nd.sem..grade..V1
Min.   :-2.1968541                  Min.   :-1.9632667
1st Qu.: 0.0741543                  1st Qu.: 0.0997531
Median : 0.3395968                  Median : 0.3780209
Mean   : 0.0000000                  Mean   : 0.0000000
3rd Qu.: 0.5696470                  3rd Qu.: 0.5955176
Max.   : 1.6999898                  Max.   : 1.6007542
Unemployment.rate.V1  Inflation.rate.V1          GDP.V1
Min.   :-1.4888746   Min.   :-1.4667052   Min.   :-1.7894645
1st Qu.:-0.8131610   1st Qu.:-0.6711664   1st Qu.:-0.7497873
Median :-0.1749870   Median : 0.1243724   Median : 0.1401058
Mean   : 0.0000000   Mean   : 0.0000000   Mean   : 0.0000000
3rd Qu.: 0.8761230   3rd Qu.: 0.9922329   3rd Qu.: 0.7877013
Max.   : 1.7395349   Max.   : 1.7877717   Max.   : 1.5454321
```

While all three of the chosen models do not require for the input attributes to be standardized, it is however good practice to have the attributes standardized to the same scale. This is because while Random Forest and Decision Tree models are typically not affected by the scale of their input attributes, and while such is the case for XGBoost too, it is important to note that unlike its other two counterparts, XGBoost could however benefit from the standardization process, in cases where feature scales are vastly varied, as is the case of the dataset employed in this study. Performing standardization meant that the training process could not only be accelerated, but at the same time stabilized as well. A complete summary statistic is then displayed once standardization of all 7 continuous attributes mentioned above is completed. The mean value of all these said attributes has now changed to 0, essentially indicating that the standardization was a success.

5.1.3 Splitting Data into Training and Test Sets

```
# Subsection 4.1.3: Splitting Data into Training and Testing Sets

# Divide the dataset into training and testing sets using a 70:30 ratio
set.seed(321)
split = sample.split(cleaned_ds$Target, SplitRatio = 0.7)
training_data = subset(cleaned_ds, split == TRUE)
testing_data = subset(cleaned_ds, split == FALSE)
```

Recall previously on the discussion about the importance of having subset of training and test sets. To perform the data split, the seed number is first set to "321" for reproducibility purposes. Following that, the sample.split function which is from the "caTools" package was called upon to perform the splitting with a 70 to 30 ratio. That is, 70% of the total observations will be assigned to the training set (represented by the *training_data* variable), while the remaining 30% of the observations will be allocated to the testing set (represented by the *testing_data* variables). Once all of the above is completed, the process is data preparation for modelling purposes is then said to have reached the end. Implementation of the models will then follow.

5.2 Implementation of XGBoost Model

5.2.1 XGBoost Baseline Model

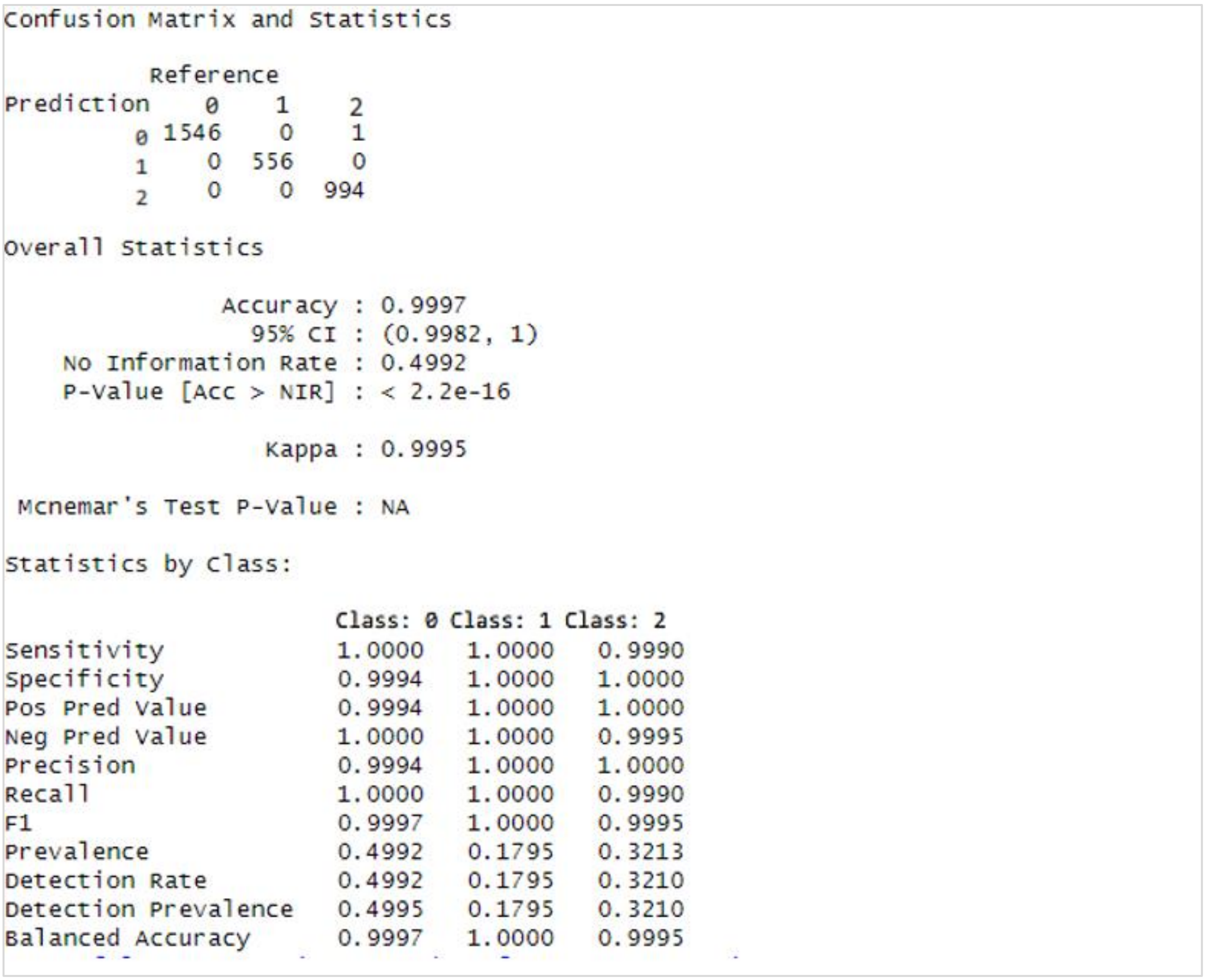
This subsection addresses the process of implementing the baseline model for the first model of XGBoost and the evaluation of the performance of both the training and testing datasets.

```r
# Subsection 4.2: Building an XGBoost Model

# Subsection 4.2.1: Initial Model Training

# Setting default parameters for XGBoost
params <- list(
  booster = "gbtree",
  num_class = 3,
  objective = "multi:softprob",
  verbosity = 1
)

# Converting categorical variables in training data to numeric
training_data[] <- lapply(training_data, function(x) if (is.factor(x)) as.numeric(x) else x)

# Identifying the position of the target variable in the dataset
target_column_index <- which(colnames(training_data) == "Target")

# Preparing feature matrix and target vector for model input
features_matrix <- as.matrix(training_data[, -target_column_index, drop = FALSE])
labels_vector <- as.vector(training_data[, target_column_index]) - 1

# Creating DMatrix for XGBoost
xgb_data <- xgb.DMatrix(data = features_matrix, label = labels_vector)

# Executing the XGBoost training
set.seed(321)
xgb_base_model <- xgb.train(params = params, data = xgb_data, nrounds = 100)
```

The implementation of the baseline XGBoost model began by first initializing the default parameters for the model. The parameters initialized are namely, the booster, num_class, objective and verbosity settings. Following that, all factor attributes in the training data are then converted into numeric data types in order to be in line with the requirements of the XGBoost model. The feature matrix and target vector for the model input are subsequently created as well, followed by the creating of the DMatrix. The seed number is set to "321" once again to ensure that all outputs of the model could be reproduced easily. Training is the caried out where there are a total number of 100 boosting which have occurred during the training.

5.2.1.1 Evaluating the Baseline Model's Performance (Training Data)

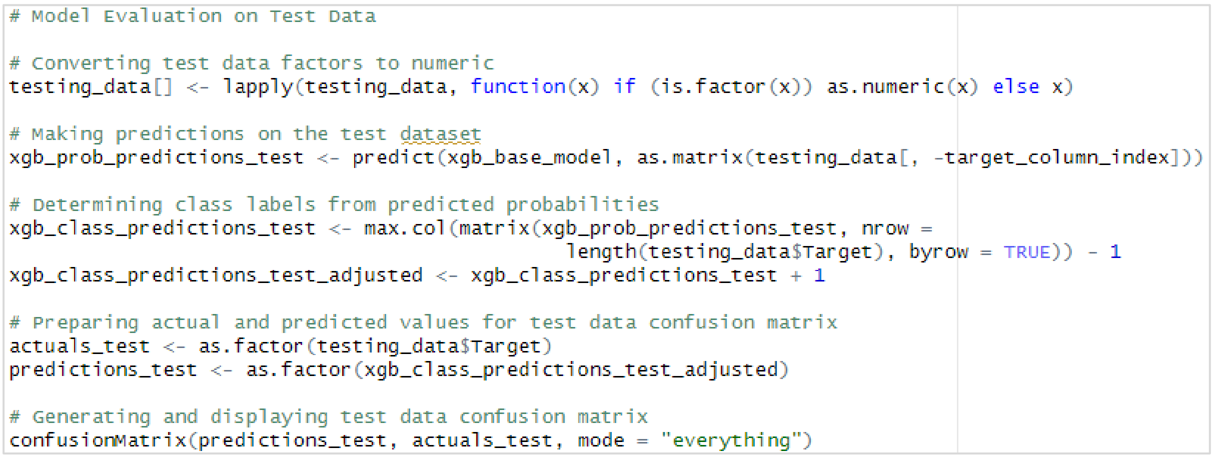
```r
# Assessing the Base Model on Training Data

# Predicting probabilities on training data
xgb_prob_predictions <- predict(xgb_base_model, as.matrix(training_data[, -target_column_index]))

# Transforming probabilities to predicted classes
xgb_class_predictions <- max.col(matrix(xgb_prob_predictions, nrow =
                                         length(training_data$Target), byrow = TRUE))

# Preparing actual and predicted values for confusion matrix
actuals <- as.factor(training_data$Target)
predictions <- as.factor(xgb_class_predictions)

# Generating and displaying training data confusion matrix
confusionMatrix(predictions, actuals, mode = "everything")
```



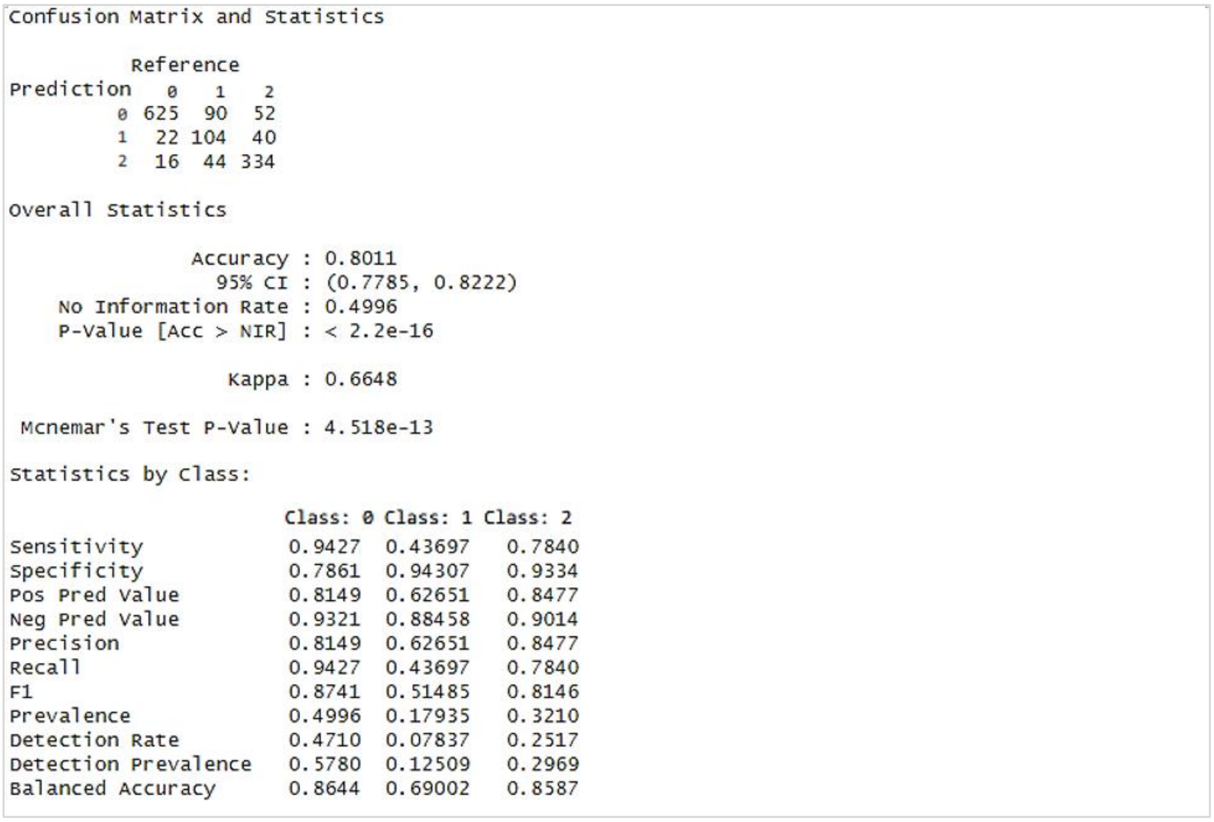
```
Confusion Matrix and Statistics

          Reference
Prediction    0    1    2
         0 1546    0    1
         1    0  556    0
         2    0    0  994

Overall Statistics

               Accuracy : 0.9997
                 95% CI : (0.9982, 1)
    No Information Rate : 0.4992
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9995

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            1.0000   1.0000   0.9990
Specificity            0.9994   1.0000   1.0000
Pos Pred Value         0.9994   1.0000   1.0000
Neg Pred Value         1.0000   1.0000   0.9995
Precision              0.9994   1.0000   1.0000
Recall                 1.0000   1.0000   0.9990
F1                     0.9997   1.0000   0.9995
Prevalence             0.4992   0.1795   0.3213
Detection Rate         0.4992   0.1795   0.3210
Detection Prevalence   0.4995   0.1795   0.3210
Balanced Accuracy      0.9997   1.0000   0.9995
```

Upon completion of the training process, the performance of the model on the training data is then evaluated. Making reference to the confusion matrix above, the baseline model has attained a close to perfect score of 99.97% accuracy with the training dataset. The high values of the other metrics namely Precision and Recall all proved that the model has had an outstanding performance. Note that both Recall and Sensitivity are referring to the same concept. Hence, for the purpose of consistency throughout the report, only the Recall metric would be mentioned.

5.2.1.2 Evaluating the Baseline Model’s Performance (Testing Data)



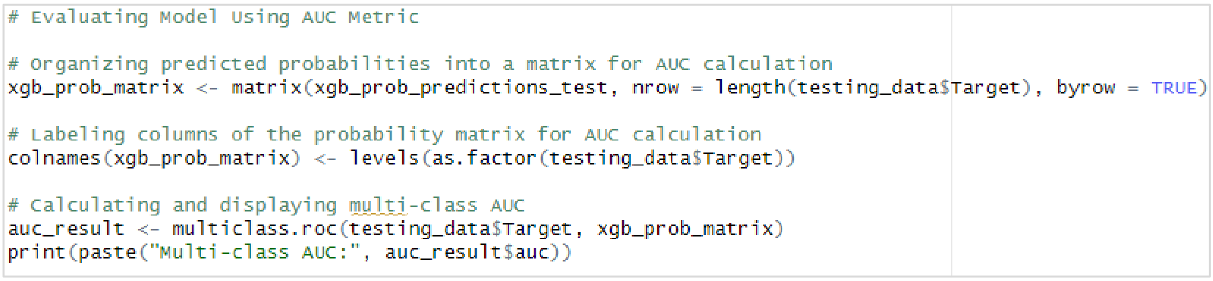
```r
# Model Evaluation on Test Data

# Converting test data factors to numeric
testing_data[] <- lapply(testing_data, function(x) if (is.factor(x)) as.numeric(x) else x)

# Making predictions on the test dataset
xgb_prob_predictions_test <- predict(xgb_base_model, as.matrix(testing_data[, -target_column_index]))

# Determining class labels from predicted probabilities
xgb_class_predictions_test <- max.col(matrix(xgb_prob_predictions_test, nrow =
                                               length(testing_data$Target), byrow = TRUE)) - 1
xgb_class_predictions_test_adjusted <- xgb_class_predictions_test + 1

# Preparing actual and predicted values for test data confusion matrix
actuals_test <- as.factor(testing_data$Target)
predictions_test <- as.factor(xgb_class_predictions_test_adjusted)

# Generating and displaying test data confusion matrix
confusionMatrix(predictions_test, actuals_test, mode = "everything")
```



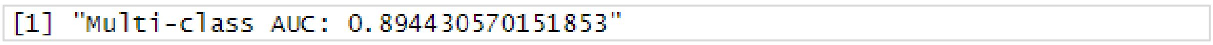
```
Confusion Matrix and Statistics

          Reference
Prediction   0   1   2
         0 625  90  52
         1  22 104  40
         2  16  44 334

Overall Statistics

               Accuracy : 0.8011
                 95% CI : (0.7785, 0.8222)
    No Information Rate : 0.4996
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6648

 Mcnemar's Test P-Value : 4.518e-13

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            0.9427  0.43697   0.7840
Specificity            0.7861  0.94307   0.9334
Pos Pred Value         0.8149  0.62651   0.8477
Neg Pred Value         0.9321  0.88458   0.9014
Precision              0.8149  0.62651   0.8477
Recall                 0.9427  0.43697   0.7840
F1                     0.8741  0.51485   0.8146
Prevalence             0.4996  0.17935   0.3210
Detection Rate         0.4710  0.07837   0.2517
Detection Prevalence   0.5780  0.12509   0.2969
Balanced Accuracy      0.8644  0.69002   0.8587
```

Resembling to the training process of the baseline model using the training dataset, before carrying out the testing process, all factor attributes within the testing dataset were converted to numeric data types first. Following that, the initially trained baseline model is then employed to make predictions with the testing dataset as the input. Performance of the model on the testing dataset is outlined in the confusion matrix shown above. Accuracy value stood at 80.11%, a reasonably high and acceptable accuracy. Precision and Recall values remained considerably high as well for the Class 0 (Graduate) and Class 2 (Dropout) . Even though

Precision and Recall was slightly lower for the Class 1 (Enrolled), it is still acceptable, more so since the primary objective of this analysis is focused more heavily on the Dropout class.



```r
# Evaluating Model Using AUC Metric

# Organizing predicted probabilities into a matrix for AUC calculation
xgb_prob_matrix <- matrix(xgb_prob_predictions_test, nrow = length(testing_data$Target), byrow = TRUE)

# Labeling columns of the probability matrix for AUC calculation
colnames(xgb_prob_matrix) <- levels(as.factor(testing_data$Target))

# Calculating and displaying multi-class AUC
auc_result <- multiclass.roc(testing_data$Target, xgb_prob_matrix)
print(paste("Multi-class AUC:", auc_result$auc))
```



```
[1] "Multi-class AUC: 0.894430570151853"
```

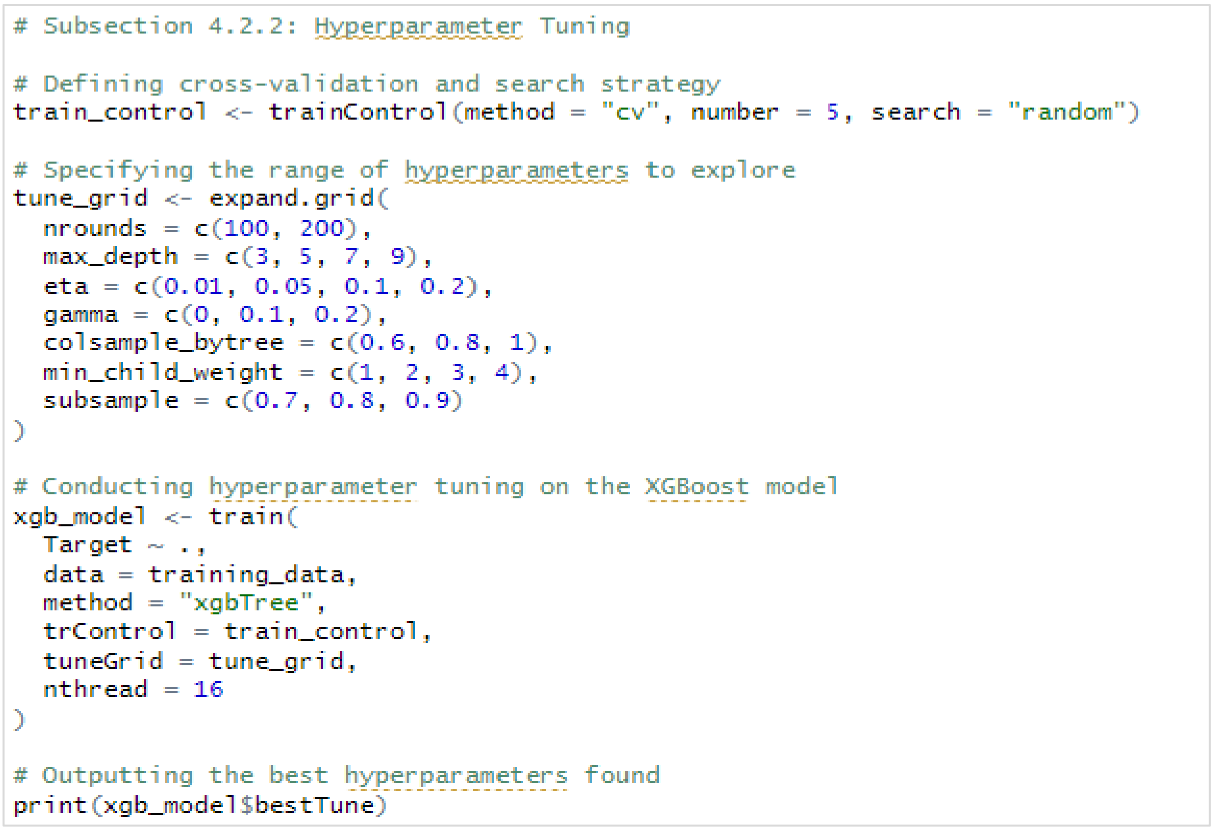
Lastly, the AUC value of 0.8944 indicates that the model is able to differentiate between the classes relatively well, when tested with the testing data.

In summary, the baseline XGBoost model exhibited commendable performance when dealing with both training and testing dataset. Notice that there are relatively sizable differences between the metrics of the training dataset and that of the testing dataset. This phenomenon suggests that there might be the potential of overfitting in the model. To address this issue, hyperparameter tuning will be carried in the subsequent subsections.

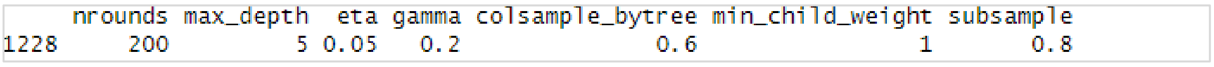
5.2.2 Refined XGBoost Model

Having implemented the baseline model, hyperparameter tuning will be applied to the model to further optimize the model’s performance, ultimately resulting in a refined model.

5.2.2.1 Model Refinement Through Hyperparameter Tuning



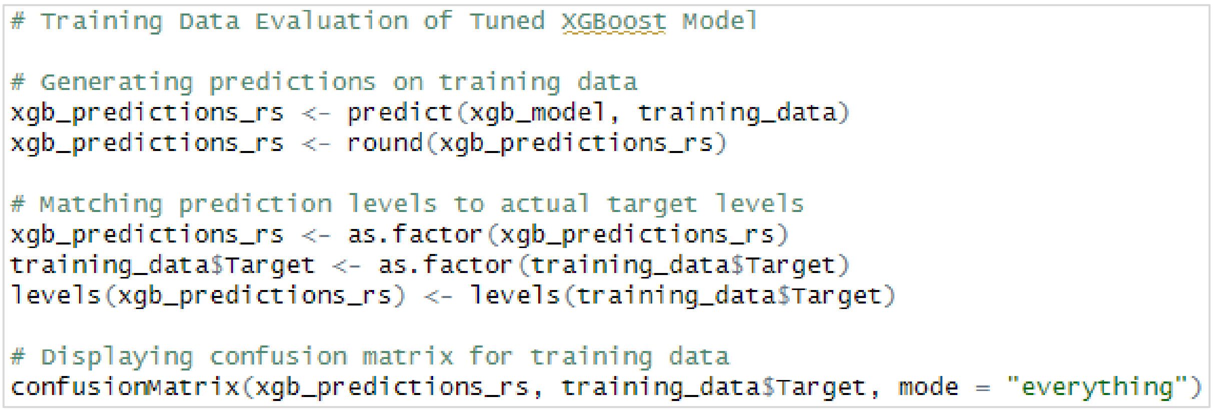
```
# Subsection 4.2.2: Hyperparameter Tuning

# Defining cross-validation and search strategy
train_control <- trainControl(method = "cv", number = 5, search = "random")

# Specifying the range of hyperparameters to explore
tune_grid <- expand.grid(
  nrounds = c(100, 200),
  max_depth = c(3, 5, 7, 9),
  eta = c(0.01, 0.05, 0.1, 0.2),
  gamma = c(0, 0.1, 0.2),
  colsample_bytree = c(0.6, 0.8, 1),
  min_child_weight = c(1, 2, 3, 4),
  subsample = c(0.7, 0.8, 0.9)
)

# Conducting hyperparameter tuning on the XGBoost model
xgb_model <- train(
  Target ~ .,
  data = training_data,
  method = "xgbTree",
  trControl = train_control,
  tuneGrid = tune_grid,
  nthread = 16
)

# Outputting the best hyperparameters found
print(xgb_model$bestTune)
```



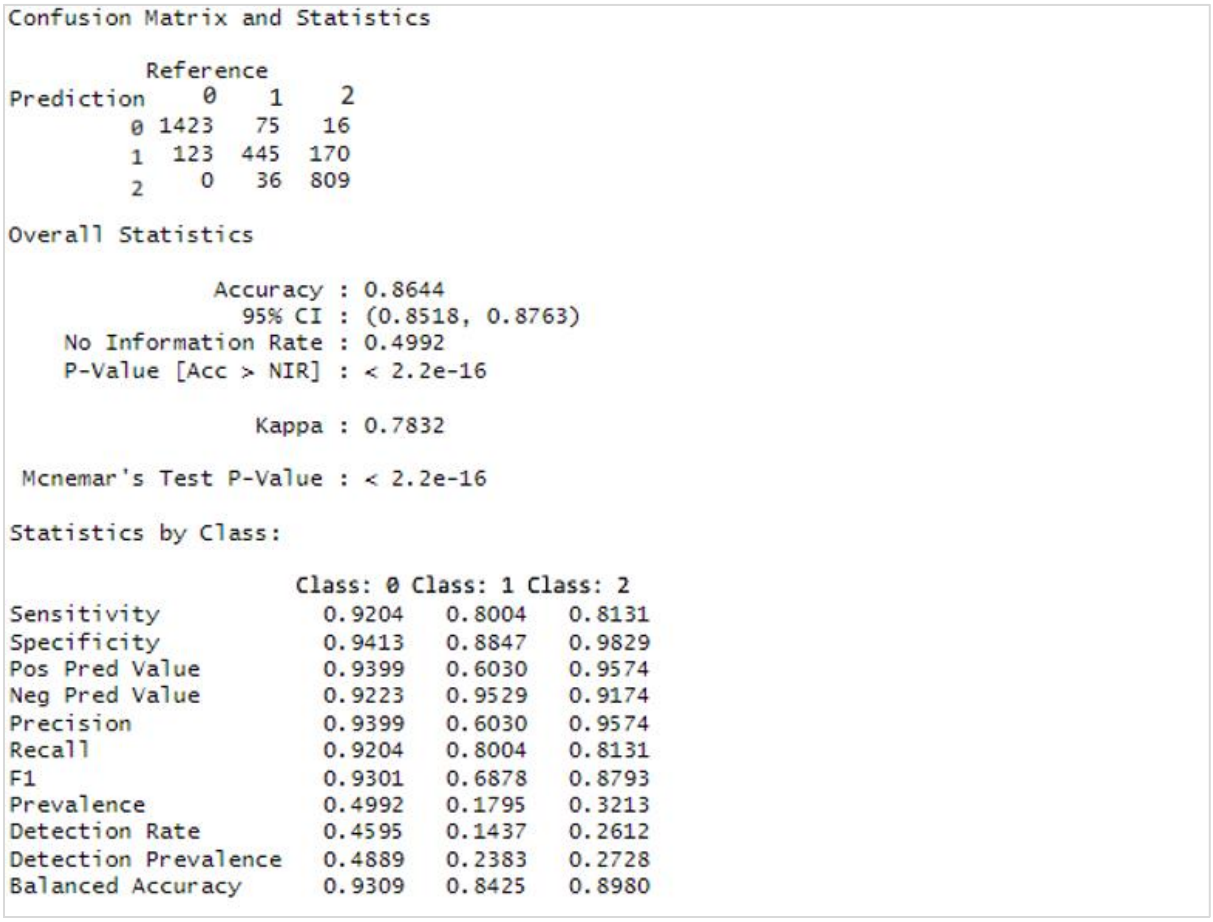
```
     nrounds max_depth  eta gamma colsample_bytree min_child_weight subsample
1228     200         5 0.05   0.2              0.6                1       0.8
```

Before proceeding with defining the range of hyperparameters to explore, a cross-validation and search strategy is first defined and assigned as *train_control*. The hyperparameter landscape is then specified as could be seen in the source code. A random search approach is then employed to implement the hyperparameter search using the initially specified range of hyperparameters. Once the random search has been concluded, the best hyperparameters found were then printed out as observed in the code output above.

5.2.2.2 Evaluating the Refined Model's Performance (Training Data)



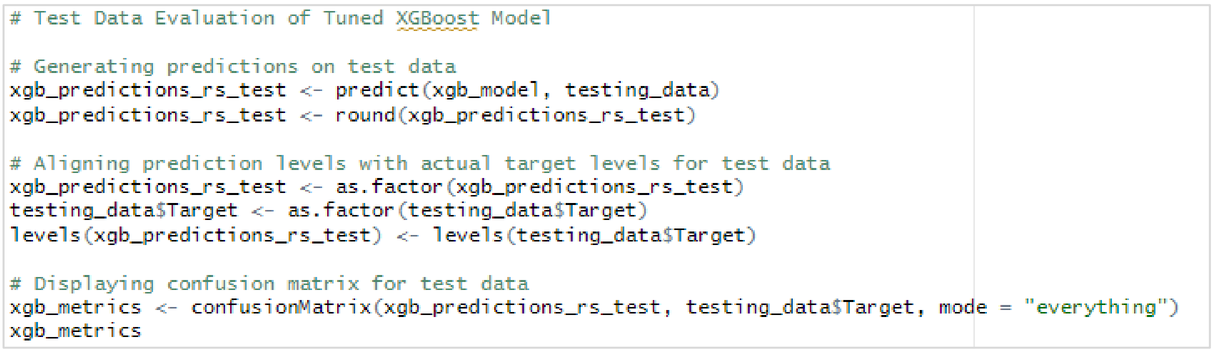
```r
# Training Data Evaluation of Tuned XGBoost Model

# Generating predictions on training data
xgb_predictions_rs <- predict(xgb_model, training_data)
xgb_predictions_rs <- round(xgb_predictions_rs)

# Matching prediction levels to actual target levels
xgb_predictions_rs <- as.factor(xgb_predictions_rs)
training_data$Target <- as.factor(training_data$Target)
levels(xgb_predictions_rs) <- levels(training_data$Target)

# Displaying confusion matrix for training data
confusionMatrix(xgb_predictions_rs, training_data$Target, mode = "everything")
```



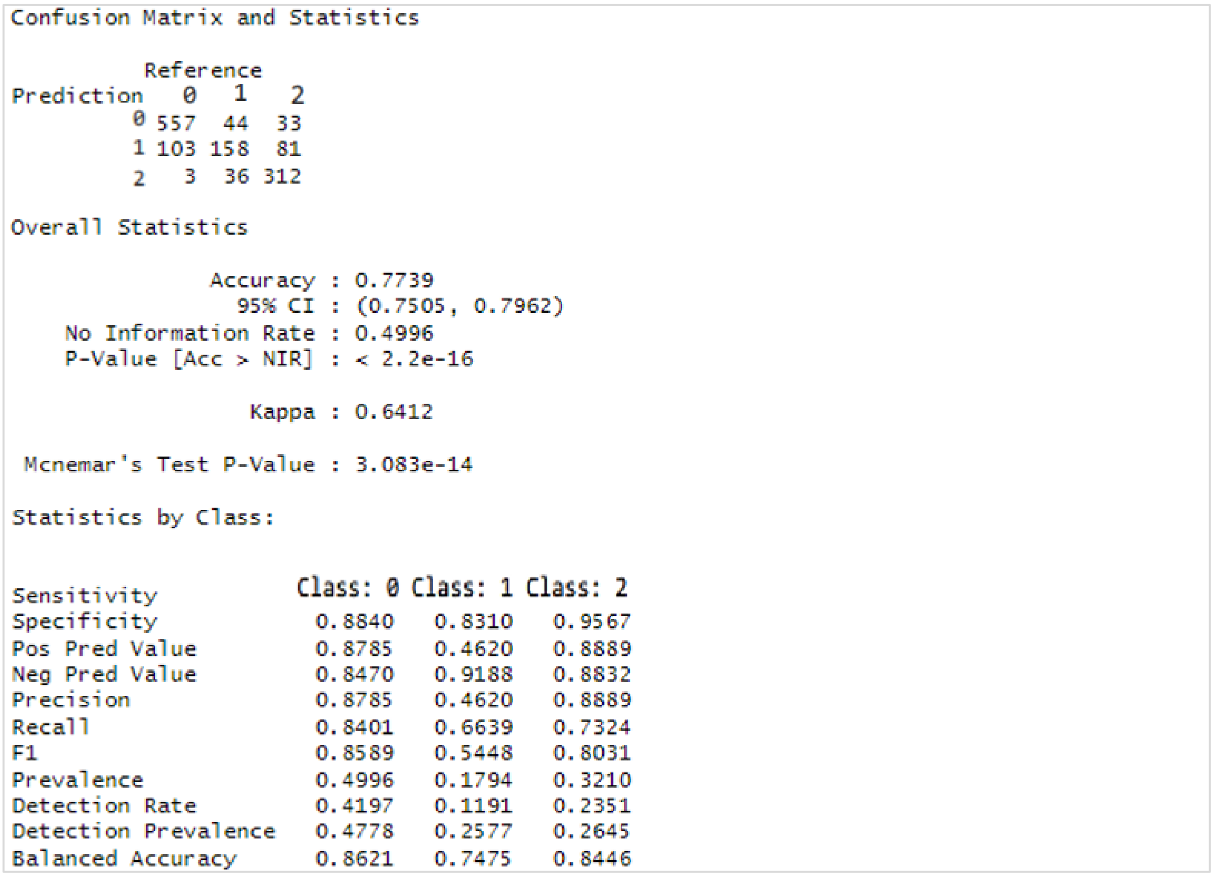
```
Confusion Matrix and Statistics

          Reference
Prediction    0    1    2
         0 1423   75   16
         1  123  445  170
         2    0   36  809

Overall Statistics

               Accuracy : 0.8644
                 95% CI : (0.8518, 0.8763)
    No Information Rate : 0.4992
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7832

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            0.9204   0.8004   0.8131
Specificity            0.9413   0.8847   0.9829
Pos Pred Value         0.9399   0.6030   0.9574
Neg Pred Value         0.9223   0.9529   0.9174
Precision              0.9399   0.6030   0.9574
Recall                 0.9204   0.8004   0.8131
F1                     0.9301   0.6878   0.8793
Prevalence             0.4992   0.1795   0.3213
Detection Rate         0.4595   0.1437   0.2612
Detection Prevalence   0.4889   0.2383   0.2728
Balanced Accuracy      0.9309   0.8425   0.8980
```

Having identified the best hyperparameter, the model is now trained on the training dataset using the hyperparameter previously identified. The confusion matrix outlines the refined model's performance where Accuracy of the model stood at 86.44% while Precision and Recall for all classes stood considerably high, with the exception to the Class 1 (Enrolled). Even though Precision and Recall was slightly lower for the Class 1 (Enrolled), the results for that class still remained reasonably acceptable.

5.2.2.3 Evaluating the Refined Model's Performance (Testing Data)

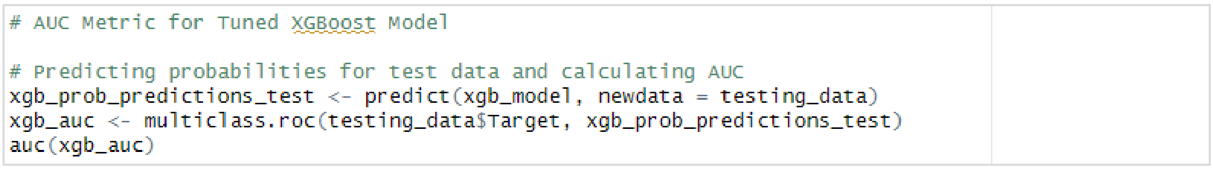


```
# Test Data Evaluation of Tuned XGBoost Model

# Generating predictions on test data
xgb_predictions_rs_test <- predict(xgb_model, testing_data)
xgb_predictions_rs_test <- round(xgb_predictions_rs_test)

# Aligning prediction levels with actual target levels for test data
xgb_predictions_rs_test <- as.factor(xgb_predictions_rs_test)
testing_data$Target <- as.factor(testing_data$Target)
levels(xgb_predictions_rs_test) <- levels(testing_data$Target)

# Displaying confusion matrix for test data
xgb_metrics <- confusionMatrix(xgb_predictions_rs_test, testing_data$Target, mode = "everything")
xgb_metrics
```

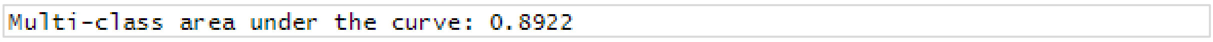


```
Confusion Matrix and Statistics

          Reference
Prediction   0   1   2
         0 557  44  33
         1 103 158  81
         2   3  36 312

Overall Statistics

               Accuracy : 0.7739
                 95% CI : (0.7505, 0.7962)
    No Information Rate : 0.4996
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6412

 Mcnemar's Test P-Value : 3.083e-14

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity
Specificity            0.8840   0.8310   0.9567
Pos Pred Value         0.8785   0.4620   0.8889
Neg Pred Value         0.8470   0.9188   0.8832
Precision              0.8785   0.4620   0.8889
Recall                 0.8401   0.6639   0.7324
F1                     0.8589   0.5448   0.8031
Prevalence             0.4996   0.1794   0.3210
Detection Rate         0.4197   0.1191   0.2351
Detection Prevalence   0.4778   0.2577   0.2645
Balanced Accuracy      0.8621   0.7475   0.8446
```

Similar to the process of the training of the refined model discussed above, except that now predictions are made using the testing dataset, overall accuracy of the model output was slightly lower, standing at 77.39%. Similar patterns of the Precision and Recall metrics among the three classes were observed. That is, the Class 1 (Enrolled) once again has had slightly lower values for both the said metrics.



```
# AUC Metric for Tuned XGBoost Model

# Predicting probabilities for test data and calculating AUC
xgb_prob_predictions_test <- predict(xgb_model, newdata = testing_data)
xgb_auc <- multiclass.roc(testing_data$Target, xgb_prob_predictions_test)
auc(xgb_auc)
```



```
Multi-class area under the curve: 0.8922
```

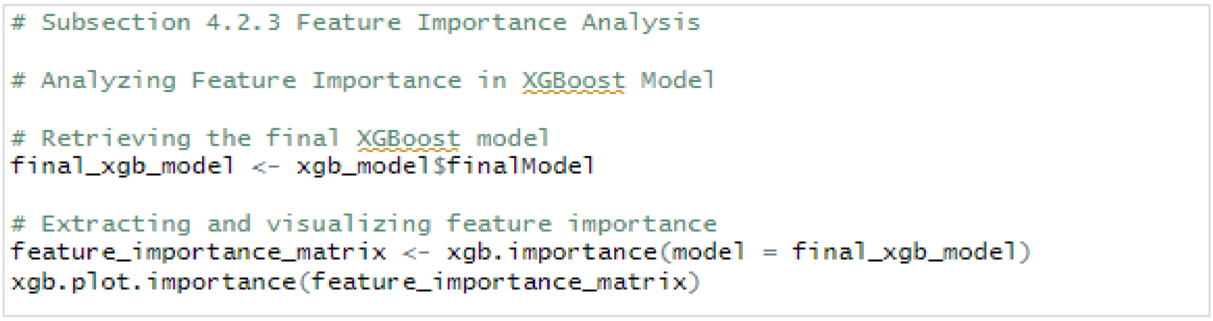
The AUC value of the refined XGBoost model after hyperparameter tuning has been applied stood at a considerably high value of 0.8922. Robustness of the refined model's ability to effectively distinguish between the classes of the target variable is implied.

Metrics	**Baseline Model (XGBoost)**	**Refined Model (XGBoost)**
Accuracy	80.11%	79.62%
Precision (Class 0)	0.8149	0.8785
Precision (Class 1)	0.6265	0.4620
Precision (Class 2)	0.8477	0.8889
Recall (Class 0)	0.9427	0.8401
Recall (Class 1)	0.4369	0.6639
Recall (Class 2)	0.7840	0.7324
AUC	0.8944	0.8922

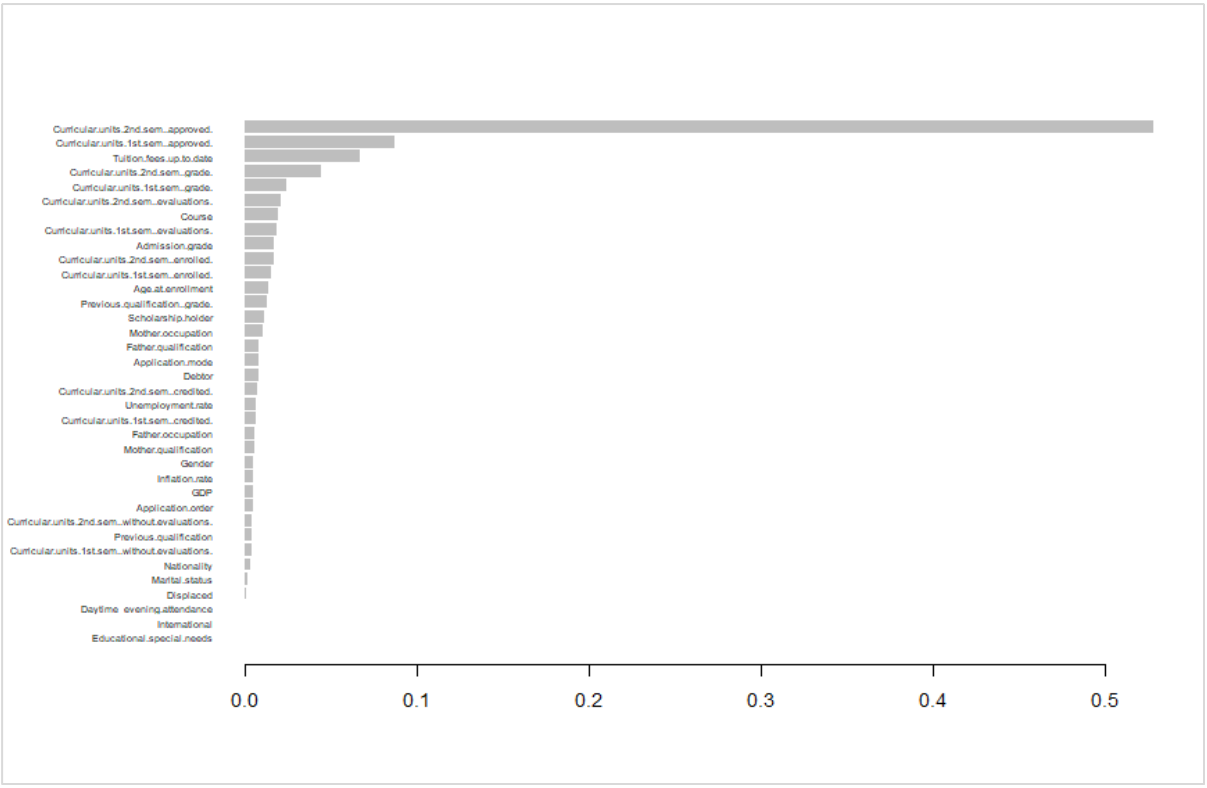
The above table summarizes the performance metrics of the models when tested used the testing dataset. While there do exists differences between the baseline model and the refined model in terms of the AUC value, the difference was merely marginal. Similar conclusion could be made with regards to the overall model's accuracy between the two models.

5.2.2.4 Significance of Features

The analysis of feature importance is aimed at identifying the attributes which have the most influence on the predictions of the model, essentially enabling stakeholders to pinpoint which attributes are indicative of the academic outcomes of students.



```
# Subsection 4.2.3 Feature Importance Analysis

# Analyzing Feature Importance in XGBoost Model

# Retrieving the final XGBoost model
final_xgb_model <- xgb_model$finalModel

# Extracting and visualizing feature importance
feature_importance_matrix <- xgb.importance(model = final_xgb_model)
xgb.plot.importance(feature_importance_matrix)
```



The above source code serves to extract the most influential features among all others in the context of the XGBoost refined model. A feature importance graph has been plotted above where several key insights could be derived from. These insights are as follows:

1. **Most Influential Attribute:** The attribute of Curricular.units.2nd.sem..approved. was found to be most influential among all the other attributes. This observation hints that the number of curricular units approved in the 2nd Semester does have considerable predictive power in identifying students at the risk of dropping out. This finding of the feature importance graph is indeed consistent with that of the correlation heatmap previously discussed.
2. **Highly Influential Attributes:** The attribute of Curricular.units.1st.sem..approved. together with the attribute of Tuition.fees.up.to.date were found to have a considerable influence on the predictions of the academic outcomes of students. In terms of the number of units approved in the first semester, intuition suggests that the approval process of curricular units could be seen as an indicator of the potential academic success of students. Although details of the approval is not made known, approval could potentially be based on the good behaviour, past results or other measures that could inform the institutions about the student's ability to proceed with the units. The attribute that tells of whether or the tuition fees of students are kept up-to-date or not suggests

that financial circumstance of the students could be the pivoting point as to whether or not they managed to graduate or dropout instead.

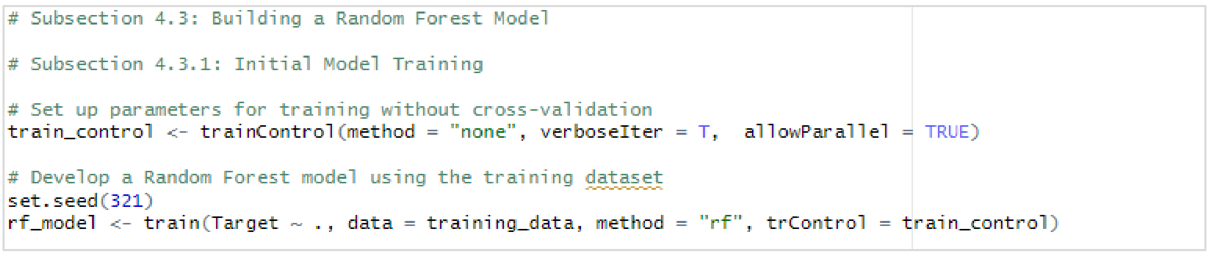
3. **Moderately Influential Attribute:** Attributes such as the Admission.grade, Course, Age.at.enrollment to name a few are identified as features with moderate influence on the prediction of student's academic outcomes. Put simply, while there is no denying that they are influential, their influence however will not be as critical as that of other attributes previously mentioned in the two points above.
4. **Minimal Influential Attribute:** Attributes such as GDP, Gender, Debtor, Parent's qualifications and occupations to name a few, were found to exhibit minimal influence. In essence, it is possible that these attributes do not directly impact the academic outcomes of students.

The understanding that the feature importance analysis provided proved to be valuable insights when stakeholders are strategizing for early interventions as well as mitigating efforts to better tackle the issue of student dropouts.

5.3 Implementation of Random Forest Model

5.3.1 Random Forest Baseline Model

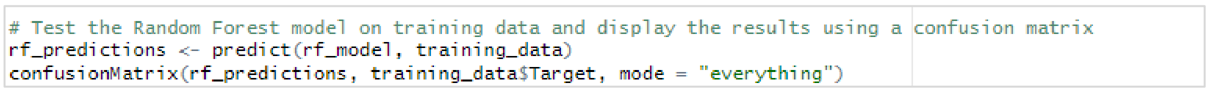
This subsection addresses the process of implementing the baseline model for the first model of Random Forest and the evaluation of the performance of both the training and testing datasets.



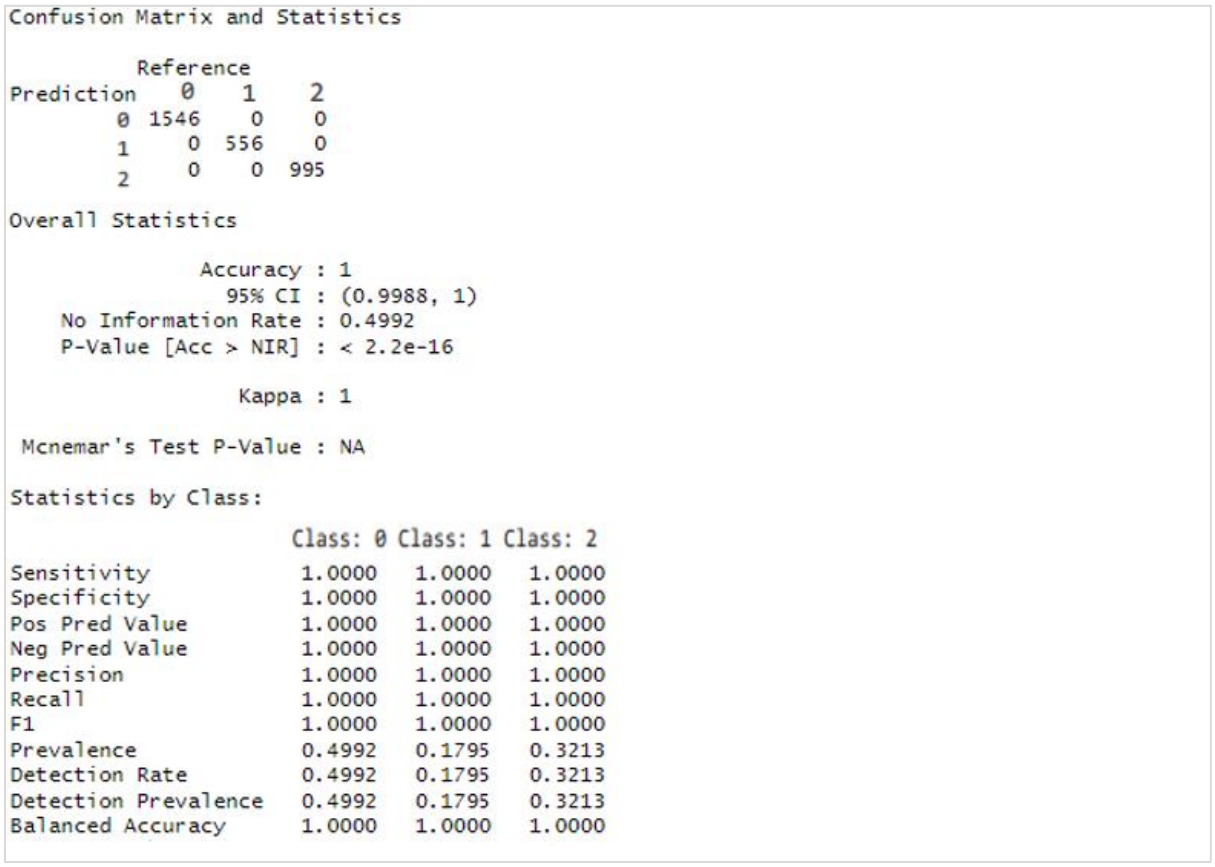
```
# Subsection 4.3: Building a Random Forest Model

# Subsection 4.3.1: Initial Model Training

# Set up parameters for training without cross-validation
train_control <- trainControl(method = "none", verboseIter = T,  allowParallel = TRUE)

# Develop a Random Forest model using the training dataset
set.seed(321)
rf_model <- train(Target ~ ., data = training_data, method = "rf", trControl = train_control)
```

The implementation of the baseline Random Forest model began by first setting up the parameters for training, denoted by *train_control.* Following that, the seed number is then defined and the baseline Random Forest model is then developed using the training dataset. Once the model has been successfully trained, the confusion matrix will then be computed in order for evaluation of the model's performance to take place.

5.3.1.1 Evaluating the Baseline Model’s Performance (Training Data)



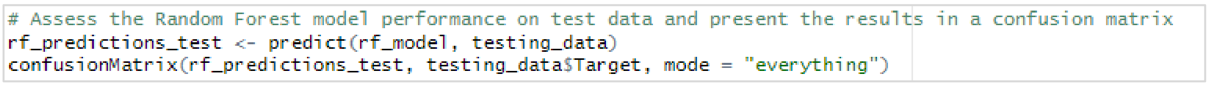
```
# Test the Random Forest model on training data and display the results using a confusion matrix
rf_predictions <- predict(rf_model, training_data)
confusionMatrix(rf_predictions, training_data$Target, mode = "everything")
```



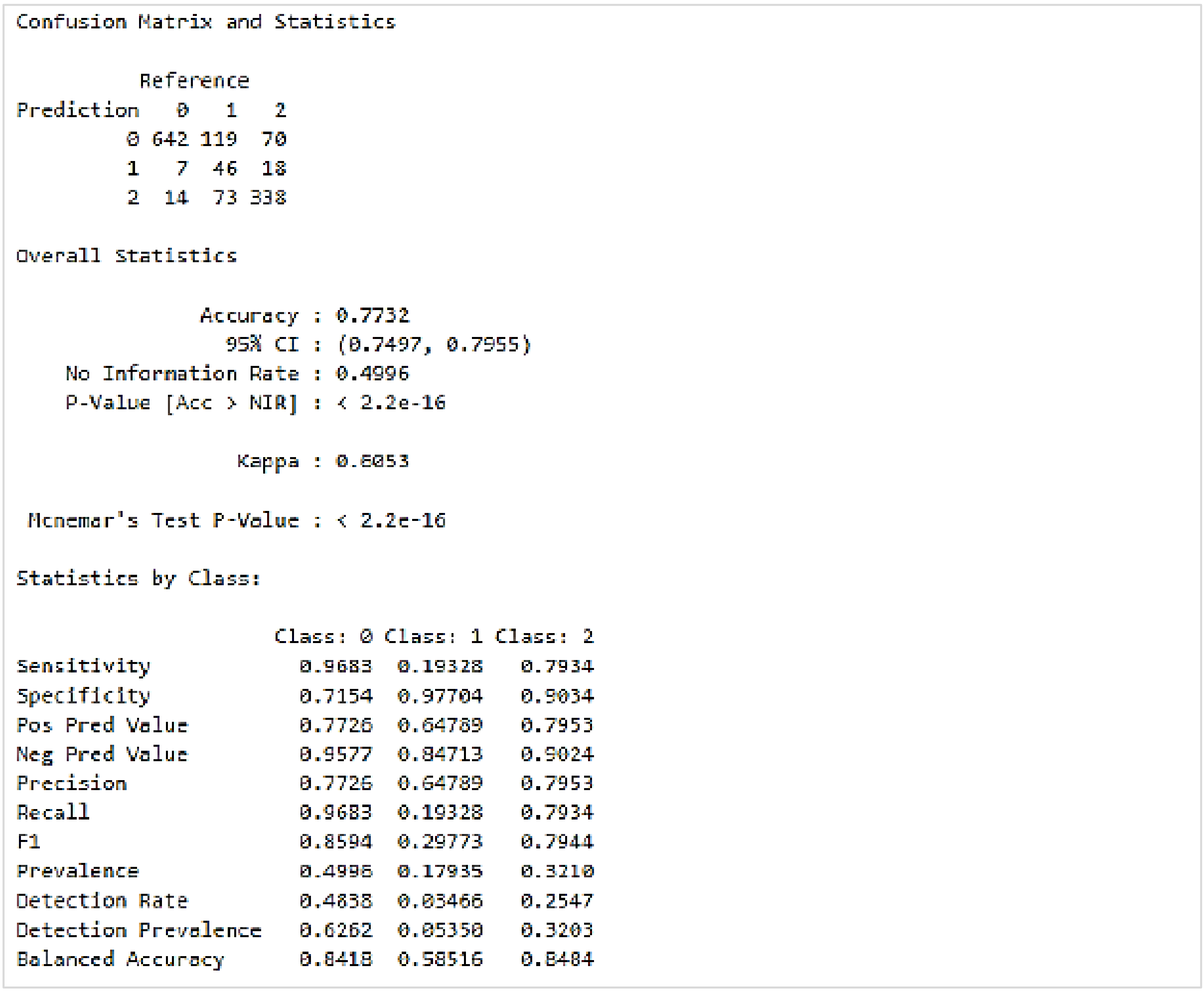
```
Confusion Matrix and Statistics

          Reference
Prediction    0    1    2
         0 1546    0    0
         1    0  556    0
         2    0    0  995

Overall Statistics

               Accuracy : 1
                 95% CI : (0.9988, 1)
    No Information Rate : 0.4992
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            1.0000   1.0000   1.0000
Specificity            1.0000   1.0000   1.0000
Pos Pred Value         1.0000   1.0000   1.0000
Neg Pred Value         1.0000   1.0000   1.0000
Precision              1.0000   1.0000   1.0000
Recall                 1.0000   1.0000   1.0000
F1                     1.0000   1.0000   1.0000
Prevalence             0.4992   0.1795   0.3213
Detection Rate         0.4992   0.1795   0.3213
Detection Prevalence   0.4992   0.1795   0.3213
Balanced Accuracy      1.0000   1.0000   1.0000
```

Upon completion of the training process, the performance of the model on the training data is then evaluated. Making reference to the confusion matrix above, the baseline model has attained a perfect score of 100% accuracy with the training dataset. The perfect scores of the other metrics namely Precision and Recall all proved that the model has had an exceptionally outstanding performance.

5.3.1.2 Evaluating the Baseline Model’s Performance (Testing Data)



```r
# Assess the Random Forest model performance on test data and present the results in a confusion matrix
rf_predictions_test <- predict(rf_model, testing_data)
confusionMatrix(rf_predictions_test, testing_data$Target, mode = "everything")
```



```
Confusion Matrix and Statistics

          Reference
Prediction   0   1   2
         0 642 119  70
         1   7  46  18
         2  14  73 338

Overall Statistics

               Accuracy : 0.7732
                 95% CI : (0.7497, 0.7955)
    No Information Rate : 0.4996
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6053

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            0.9683  0.19328   0.7934
Specificity            0.7154  0.97704   0.9034
Pos Pred Value         0.7726  0.64789   0.7953
Neg Pred Value         0.9577  0.84713   0.9024
Precision              0.7726  0.64789   0.7953
Recall                 0.9683  0.19328   0.7934
F1                     0.8594  0.29773   0.7944
Prevalence             0.4996  0.17935   0.3210
Detection Rate         0.4838  0.03466   0.2547
Detection Prevalence   0.6262  0.05350   0.3203
Balanced Accuracy      0.8418  0.58516   0.8484
```

Following the successful training of the model using the training dataset, the trained baseline model is then employed to make predictions with the testing dataset as the input. The performance of the model on the testing dataset could be found in the confusion matrix shown above. Accuracy value stood at 77.32%, a reasonably high and acceptable accuracy. Precision and Recall values remained reasonably high as well for both the Class 0 (Graduate) and Class 2 (Dropout) . Even though Recall was significantly lower for the Class 1 (Enrolled), the model is still deemed as appropriate on grounds that the priority of this study is on the groups of students at risk of dropping out rather than to predict groups who students who are likely to get enrolled.

```
# Evaluate model accuracy using Area Under Curve (AUC) metric
auc <- multiclass.roc(testing_data$Target, as.numeric(rf_predictions_test))
auc(auc)
```

```
Multi-class area under the curve: 0.8268
```

The AUC value of 0.8268 indicates that the model is able to differentiate between the classes relatively well, when tested with the testing data.

To conclude, the baseline Random Forest model demonstrated commendable performance when dealt with both training and testing dataset. The decrease in the values of the performance metrics from a higher value in the initial training data to a lower value in the testing data suggests that there might exist the potential of overfitting in the model. To address this issue, hyperparameter tuning will be carried in the subsequent subsections.

5.3.2 Refined Random Forest Model

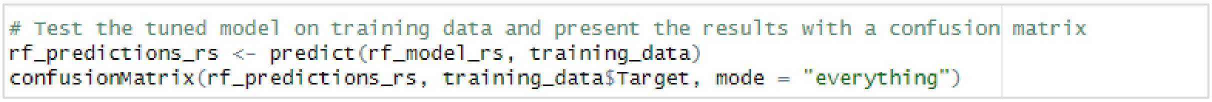
5.3.2.1 First Model Refinement Through Hyperparameter Tuning (Random Search)

```
# Subsection 4.3.2: Hyperparameter Tuning

# Set up random search for optimizing model hyperparameters with cross-validation
train_control <- trainControl(method = "cv", number = 2, search="random", allowParallel = TRUE)

# Execute random search to fine-tune the Random Forest model
set.seed(321)
rf_model_rs <- train(Target ~ ., data = training_data, method = "rf", trControl = train_control,
                     metric = "Accuracy", tuneLength = 15)

# Print the details of the model
print(rf_model_rs)
```

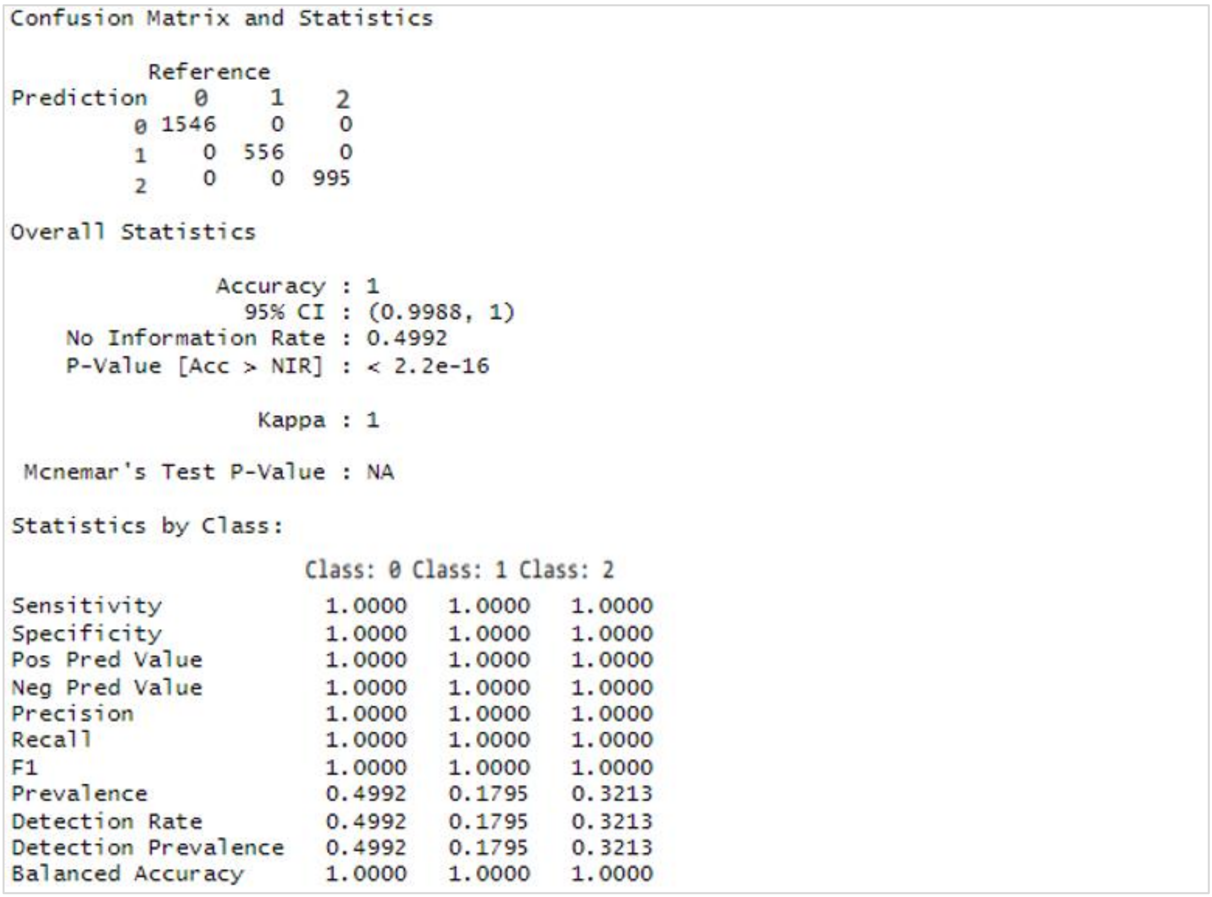
```
Random Forest

3097 samples
  36 predictor
   3 classes: '0', '1', '2'

No pre-processing
Resampling: Cross-Validated (2 fold)
Summary of sample sizes: 1549, 1548
Resampling results across tuning parameters:

  mtry  Accuracy   Kappa
  11    0.7781748  0.6262675
  13    0.7791425  0.6289887
  15    0.7739777  0.6207898
  16    0.7759140  0.6240351
  17    0.7784980  0.6285591
  18    0.7762381  0.6254905
  24    0.7746229  0.6227164
  25    0.7723623  0.6186776
  31    0.7707490  0.6161037
  32    0.7710724  0.6169267
  34    0.7720408  0.6189307
  36    0.7688123  0.6139098

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 13.
```

Similar to the hyperparameter tuning process previously implemented for the XGBoost model, the purpose of carrying out the tuning of the hyperparameters is to ensure that the models' performance could be optimized through the exploration of the best hyperparameters. For the hyperparameter tuning process, the random search technique will first be employed, followed by the evaluation of the models based on the hyperparameters identified by the random search technique. Before that, the random search function with 2-folds cross validation was created and assigned to as *train_control*. tuneLength equals to 15 meant that a total of 15 random sets of hyperparameters will be tested for. The hyperparameters of the best performing model will then be saved to *rf_model_rs*. Upon completion of the hyperparameter tuning process, the final output showed that the optimal number to use at each split ("mtry") would be 13 with the highest Accuracy of 77.91% obtained. Hence, mtry equals to 13 is set to be the final parameter for the model.

5.3.2.1.1 Evaluating the Refined Model's Performance (Training Data)



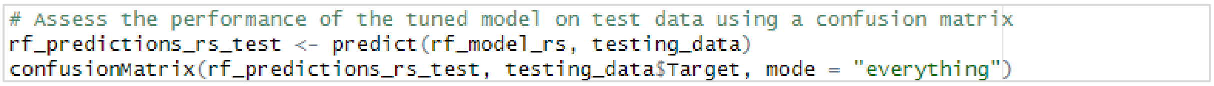
```
# Test the tuned model on training data and present the results with a confusion matrix
rf_predictions_rs <- predict(rf_model_rs, training_data)
confusionMatrix(rf_predictions_rs, training_data$Target, mode = "everything")
```



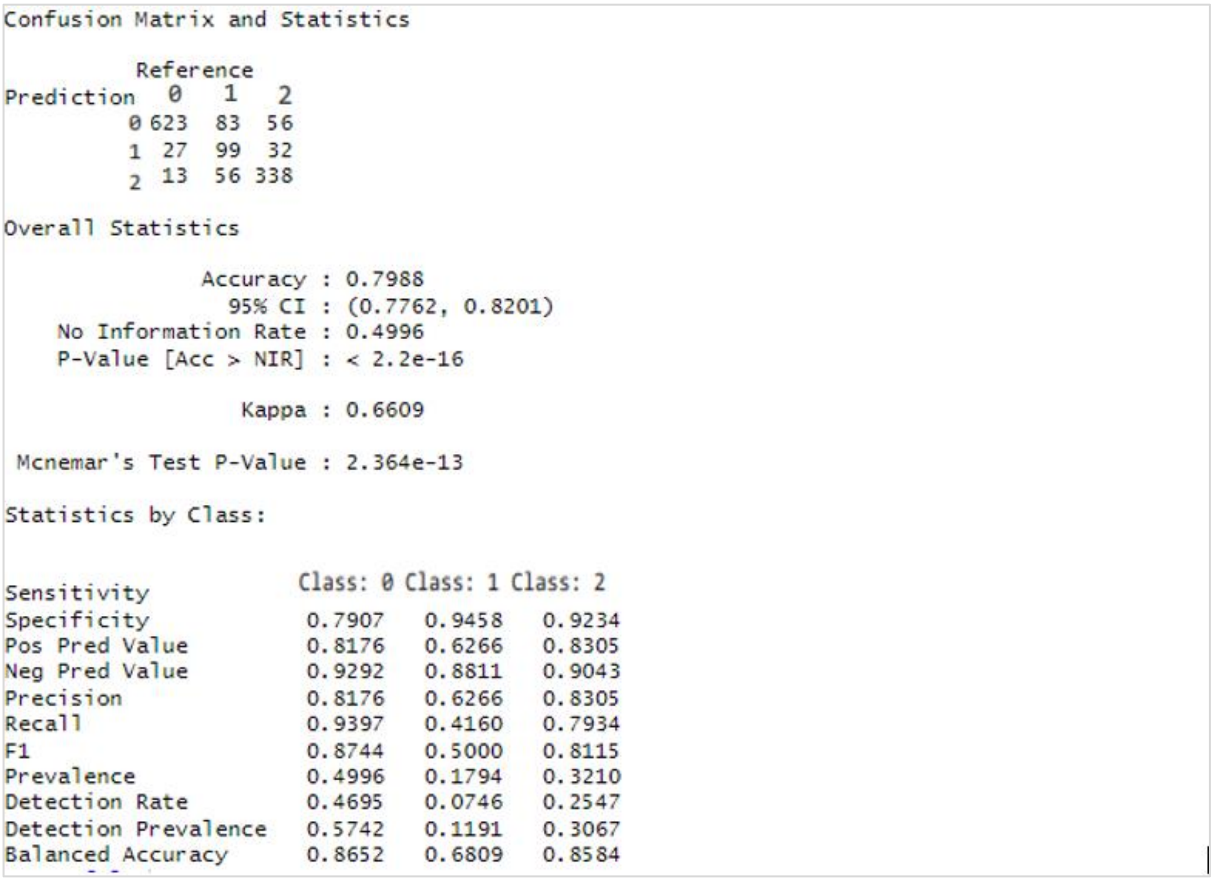
```
Confusion Matrix and Statistics

          Reference
Prediction    0    1    2
         0 1546    0    0
         1    0  556    0
         2    0    0  995

Overall Statistics

               Accuracy : 1
                 95% CI : (0.9988, 1)
    No Information Rate : 0.4992
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            1.0000   1.0000   1.0000
Specificity            1.0000   1.0000   1.0000
Pos Pred Value         1.0000   1.0000   1.0000
Neg Pred Value         1.0000   1.0000   1.0000
Precision              1.0000   1.0000   1.0000
Recall                 1.0000   1.0000   1.0000
F1                     1.0000   1.0000   1.0000
Prevalence             0.4992   0.1795   0.3213
Detection Rate         0.4992   0.1795   0.3213
Detection Prevalence   0.4992   0.1795   0.3213
Balanced Accuracy      1.0000   1.0000   1.0000
```

Having now identified the best hyperparameters according to the Random Search function, the model is now tested on the training dataset using the hyperparameter previously identified. The confusion matrix outlines the refined model's performance where Accuracy of the model stood at 100% while Precision and Recall for all classes stood at 1.

5.3.2.1.2 Evaluating the Refined Model's Performance (Testing Data)



```
# Assess the performance of the tuned model on test data using a confusion matrix
rf_predictions_rs_test <- predict(rf_model_rs, testing_data)
confusionMatrix(rf_predictions_rs_test, testing_data$Target, mode = "everything")
```



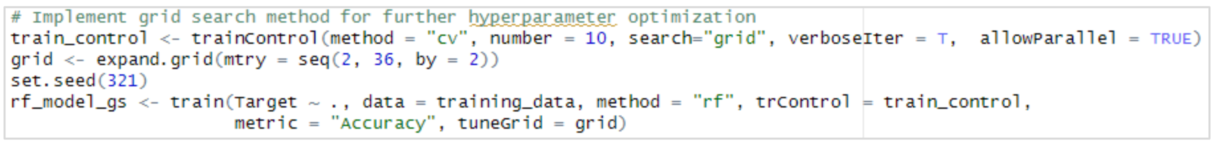
```
Confusion Matrix and Statistics

          Reference
Prediction   0   1   2
         0 623  83  56
         1  27  99  32
         2  13  56 338

Overall Statistics

               Accuracy : 0.7988
                 95% CI : (0.7762, 0.8201)
    No Information Rate : 0.4996
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6609

 Mcnemar's Test P-Value : 2.364e-13

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity
Specificity            0.7907   0.9458   0.9234
Pos Pred Value         0.8176   0.6266   0.8305
Neg Pred Value         0.9292   0.8811   0.9043
Precision              0.8176   0.6266   0.8305
Recall                 0.9397   0.4160   0.7934
F1                     0.8744   0.5000   0.8115
Prevalence             0.4996   0.1794   0.3210
Detection Rate         0.4695   0.0746   0.2547
Detection Prevalence   0.5742   0.1191   0.3067
Balanced Accuracy      0.8652   0.6809   0.8584
```

Similar to the process of the training of the refined model discussed above, except that now predictions are made using the testing dataset, overall accuracy of the model output was slightly lower, standing at 79.88%. Similar patterns of the Precision and Recall metrics among the three classes were observed. That is, the Class 1 (Enrolled) once again has had slightly lower values for both the said metrics.

Metrics	Baseline Model (Random Forest)	Refined Model (Random Forest)
Accuracy	77.32%	79.88%
Precision (Class 0)	0.7726	0.8176
Precision (Class 1)	0.6478	0.6266
Precision (Class 2)	0.7953	0.8305
Recall (Class 0)	0.9683	0.9397
Recall (Class 1)	0.1932	0.4160
Recall (Class 2)	0.7934	0.7934

The above table summarizes the performance metrics of the models when tested used the testing dataset. Overall, the accuracy of the model has increased after the implementation of hyperparameter tuning. Precision values for both Class 0 and Class 2 have seen an increase after the refinement. Recall for Class 1 has too experienced an increase. Even though there were certain reduction in the class-specific metrics, it could be concluded that overall, the model's performance did increase and the model did benefit from the tuning which was done. Also, it is important note that since the primary focus of this study is to predict students at risk of dropping out, the increased Precision and Recall values once again confirmed the enhanced performance of the refined model.

5.3.2.2 Second Model Refinement Through Hyperparameter Tuning (Grid Search)

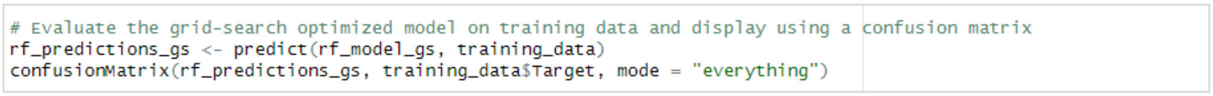


```
# Implement grid search method for further hyperparameter optimization
train_control <- trainControl(method = "cv", number = 10, search="grid", verboseIter = T,  allowParallel = TRUE)
grid <- expand.grid(mtry = seq(2, 36, by = 2))
set.seed(321)
rf_model_gs <- train(Target ~ ., data = training_data, method = "rf", trControl = train_control,
                     metric = "Accuracy", tuneGrid = grid)
```

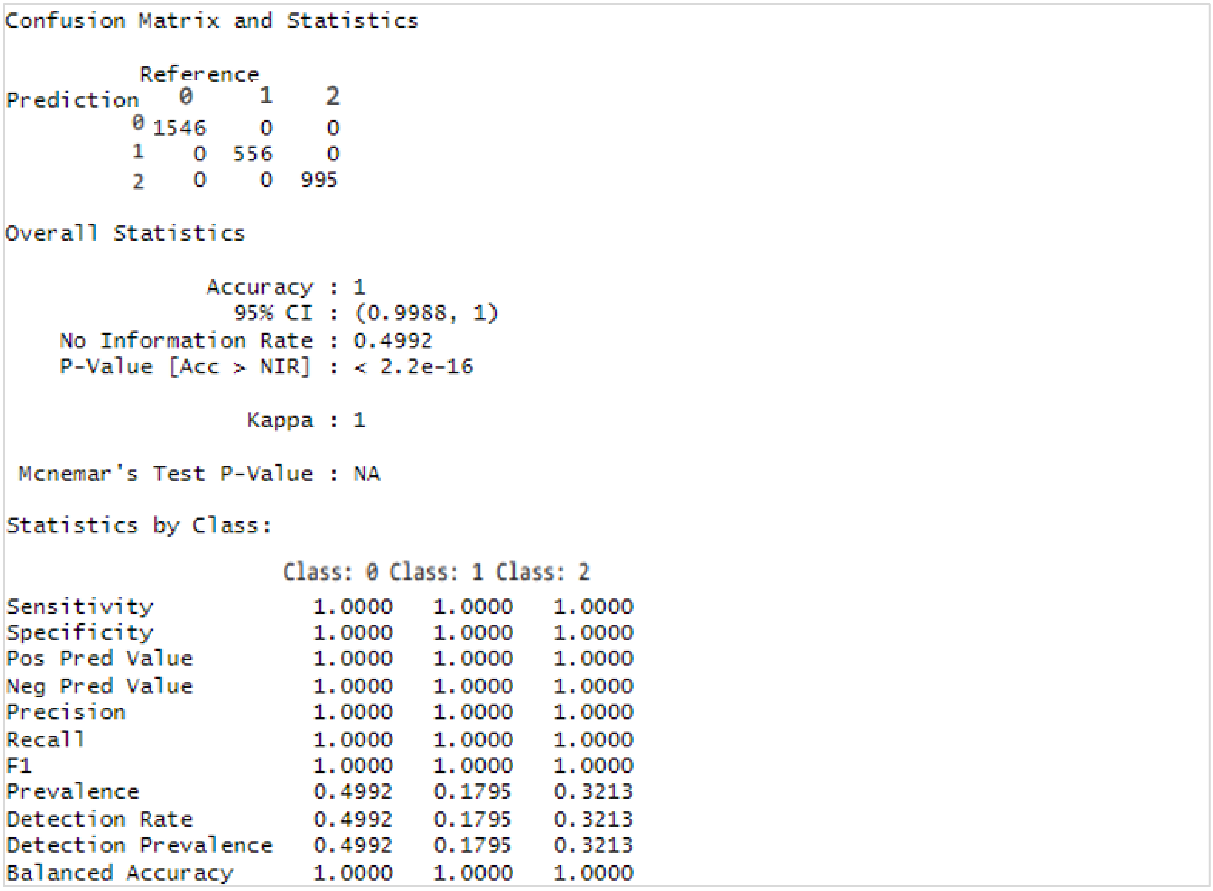
In order to further optimize the predictive power of the model, the next step is to implement a Grid Search by taking the output of hyperparameters obtained from the previously performed Random Search. Recall that the best mtry suggested by the Random Search function was 13. Given that, the range of the hyperparameter values to be carried out by the Grid Search function must thereby contain this value. Also, making reference back to the output of the finalized model of the hyperparameter tuning done in the previous subsection, the mtry values started from 11 and stopped at 36. That said, the grid range therefore must also contain the values from 11 to 36. Following that, the grid would start from mtry equals to 2 and end at mtry equals to 36, with an increment of 2 mtry values every round. Before defining the grid range however, the training control function must be specified first, where a 10-folds cross validation is to be implemented during the training process.

Once the train control and grid functions are defined, the seed number is then set to “321” for reproducibility purposes, and the model is subsequently trained using the specified grid on the training dataset.

5.3.2.2.1 Evaluating the Refined Model’s Performance (Training Data)



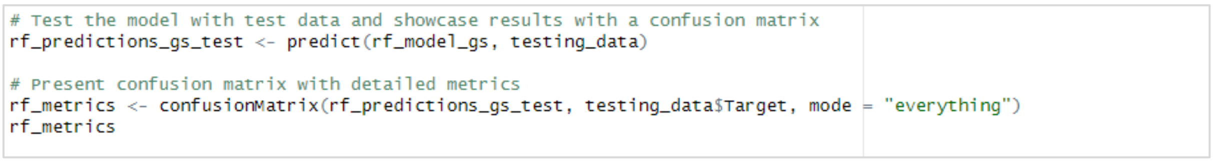
```
# Evaluate the grid-search optimized model on training data and display using a confusion matrix
rf_predictions_gs <- predict(rf_model_gs, training_data)
confusionMatrix(rf_predictions_gs, training_data$Target, mode = "everything")
```



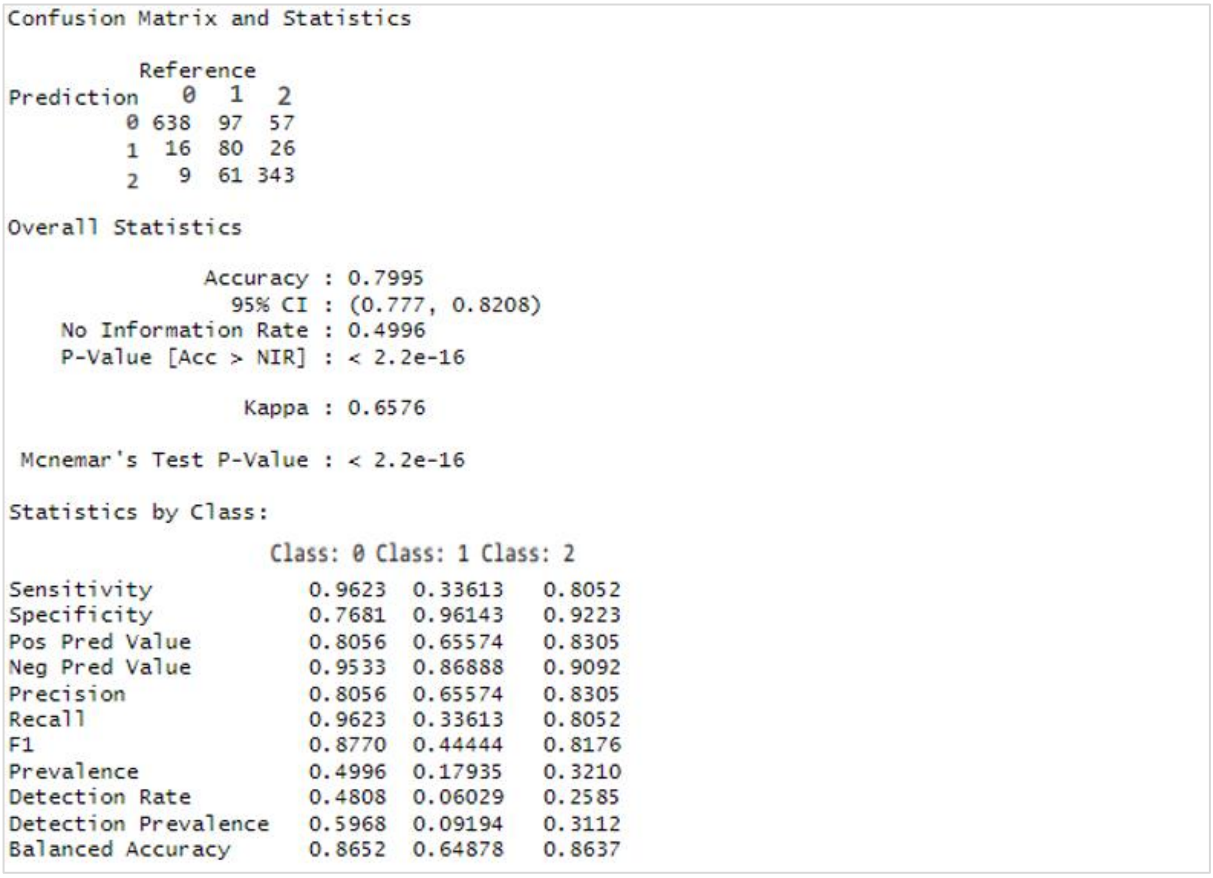
```
Confusion Matrix and Statistics

          Reference
Prediction    0    1    2
         0 1546    0    0
         1    0  556    0
         2    0    0  995

Overall Statistics

               Accuracy : 1
                 95% CI : (0.9988, 1)
    No Information Rate : 0.4992
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            1.0000   1.0000   1.0000
Specificity            1.0000   1.0000   1.0000
Pos Pred Value         1.0000   1.0000   1.0000
Neg Pred Value         1.0000   1.0000   1.0000
Precision              1.0000   1.0000   1.0000
Recall                 1.0000   1.0000   1.0000
F1                     1.0000   1.0000   1.0000
Prevalence             0.4992   0.1795   0.3213
Detection Rate         0.4992   0.1795   0.3213
Detection Prevalence   0.4992   0.1795   0.3213
Balanced Accuracy      1.0000   1.0000   1.0000
```

The model’s performance recorded a 100% accuracy in its predictions. Precision and Recall both stood at a value of 1 as well, essentially indicating that the model is very proficient in predicting the academic outcomes of students.

5.3.2.2.2 Evaluating the Refined Model’s Performance (Testing Data)



```
# Test the model with test data and showcase results with a confusion matrix
rf_predictions_gs_test <- predict(rf_model_gs, testing_data)

# Present confusion matrix with detailed metrics
rf_metrics <- confusionMatrix(rf_predictions_gs_test, testing_data$Target, mode = "everything")
rf_metrics
```



```
Confusion Matrix and Statistics

          Reference
Prediction   0   1   2
         0 638  97  57
         1  16  80  26
         2   9  61 343

Overall Statistics

               Accuracy : 0.7995
                 95% CI : (0.777, 0.8208)
    No Information Rate : 0.4996
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6576

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            0.9623  0.33613   0.8052
Specificity            0.7681  0.96143   0.9223
Pos Pred Value         0.8056  0.65574   0.8305
Neg Pred Value         0.9533  0.86888   0.9092
Precision              0.8056  0.65574   0.8305
Recall                 0.9623  0.33613   0.8052
F1                     0.8770  0.44444   0.8176
Prevalence             0.4996  0.17935   0.3210
Detection Rate         0.4808  0.06029   0.2585
Detection Prevalence   0.5968  0.09194   0.3112
Balanced Accuracy      0.8652  0.64878   0.8637
```

When the model is being tested using the testing dataset, its accuracy decreased to 79.95%, a reasonably high accuracy still. While the Precision and Recall values among the classes are varying, the priority dropout class (Class 2) recorded considerably high values. Such is the case for the Class 0 (Graduate) as well. However, it is worth noting once again that the discrepancies between the model’s performance when tested on the training and testing dataset do suggest that the phenomenon of overfitting might be present. Further and more in-depth investigation is necessary to confirm such claims.

```
# Measure model performance using Area Under the Curve
# Employ the multiclass.roc() function for AUC calculation
rf_auc <- multiclass.roc(testing_data$Target, as.numeric(rf_predictions_gs_test))
auc(rf_auc)
```

```
Multi-class area under the curve: 0.822
```

The model's AUC value is recorded at 0.822, implying that the model is proficient in discriminating between the classes. Higher AUC values are desired where the closer the value is to 1, the greater is the discriminatory power of the model.

Metrics	Refined Model using Random Search (Random Forest)	Refined Model using Grid Search (Random Forest)
Accuracy	79.88%	79.95%
Precision (Class 0)	0.8176	0.8056
Precision (Class 1)	0.6266	0.6557
Precision (Class 2)	0.8305	0.8305
Recall (Class 0)	0.9397	0.9623
Recall (Class 1)	0.4160	0.3361
Recall (Class 2)	0.7934	0.8052

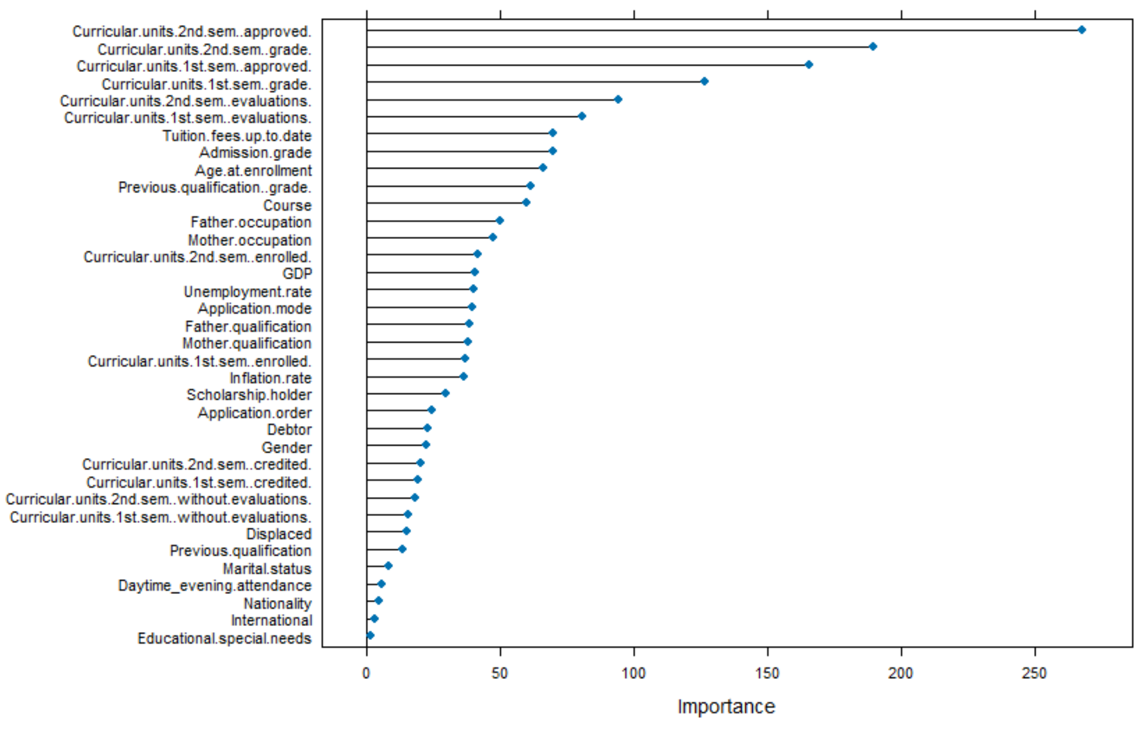
The above table summarizes the performance metrics of the models between the refined models for when Random Search and Grid Search were implemented. Overall, the accuracy of the model has increased after the implementation of hyperparameter tuning. Recall value for the Class 2 (Dropout) has increased, while Precision has remained unchanged. In essence, the implementation of Grid Search to tune the hyperparameters of the model has evidently improved the performance of the refined model.

5.3.2.2.3 Significance of Features

```
# Subsection 4.3.3: Feature Importance Analysis

# Assess and select key features based on their significance
# Determine the importance of each feature
feature_importance <- varImp(rf_model_gs, scale = FALSE)
print(feature_importance)
```

```
rf variable importance

  only 20 most important variables shown (out of 36)

                                       Overall
Curricular.units.2nd.sem..approved.     267.42
Curricular.units.2nd.sem..grade.        189.43
Curricular.units.1st.sem..approved.     165.58
Curricular.units.1st.sem..grade.        126.22
Curricular.units.2nd.sem..evaluations.   94.15
Curricular.units.1st.sem..evaluations.   80.71
Tuition.fees.up.to.date                  69.58
Admission.grade                          69.52
Age.at.enrollment                        65.89
Previous.qualification..grade.           61.05
Course                                   59.67
Father.occupation                        49.80
Mother.occupation                        47.43
Curricular.units.2nd.sem..enrolled.      41.38
GDP                                      40.58
Unemployment.rate                        39.98
Application.mode                         39.46
Father.qualification                     38.42
Mother.qualification                     37.63
Curricular.units.1st.sem..enrolled.      36.97
```

```
# Visualize the significance of each feature
plot(feature_importance)
```



To identify which attributes are most indicative of the academic outcomes of students, the varImp() function which is a function made available by the caret package is being called on. Note that the function is applied to the finalized Random Forest model (rf_model_gs) - the one whereby the hyperparameters identified by Grid Search were tuned accordingly. The resulting significance of each attribute were then visualized using the plot above. From the plot and the quantitative scoring provided, several key observations could be made, and they are as follows:

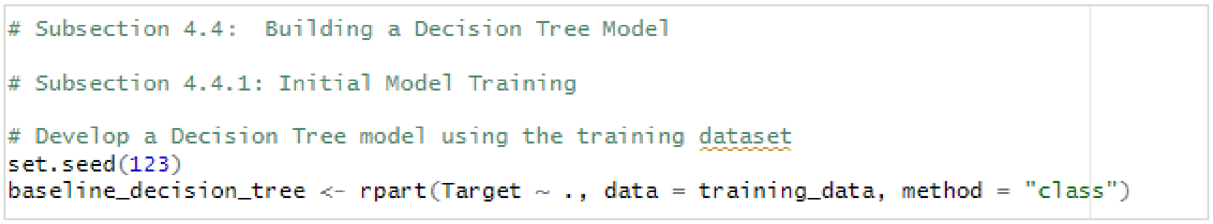
1. Substantially Influential Attributes: Consistent with findings from the feature importance analysis carried out using the refined XGBoost model, the Curricular.unit.2nd.sem..approved was determined to be the most influential attribute in the task of predicting students' academic outcomes.
2. Highly Influential Attributes: Attributes namely Curricular.units.2nd.sem..grade, Curricular.units.1st.sem..grade, and Curricular.units.1st.sem..approved were shown to have had considerable predictive power in the task of identifying students at risk of dropping out.
3. Moderately Influential Attributes: Attributes such as those pertaining to the status of the tuition fees, the course, and the admission grade of students to name a few, could be seen as contributing factors to the dropout rates of students, but their impact may not be too sizable.
4. Minimally Influential Attributes: Attributes that had a lower scoring for instance, nationality, marital status and previous qualification of students have negligible impact when trying to make any predictions pertaining to academic outcomes.

It is important to note that the findings from the above feature importance analysis are largely similar to that which was previously carried out with the refined Random Forest model, thereby reinforces the reliability of the findings.

5.4 Implementation of Decision Tree Model

5.4.1 Decision Tree Baseline Model

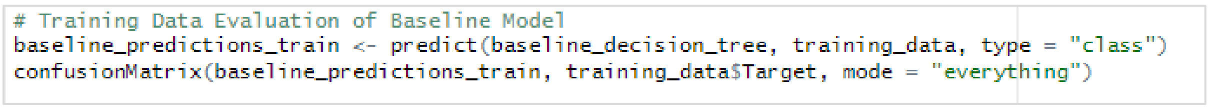
This subsection addresses the process of implementing the baseline model for the first model of Decision Tree and the evaluation of the performance of both the training and testing datasets.



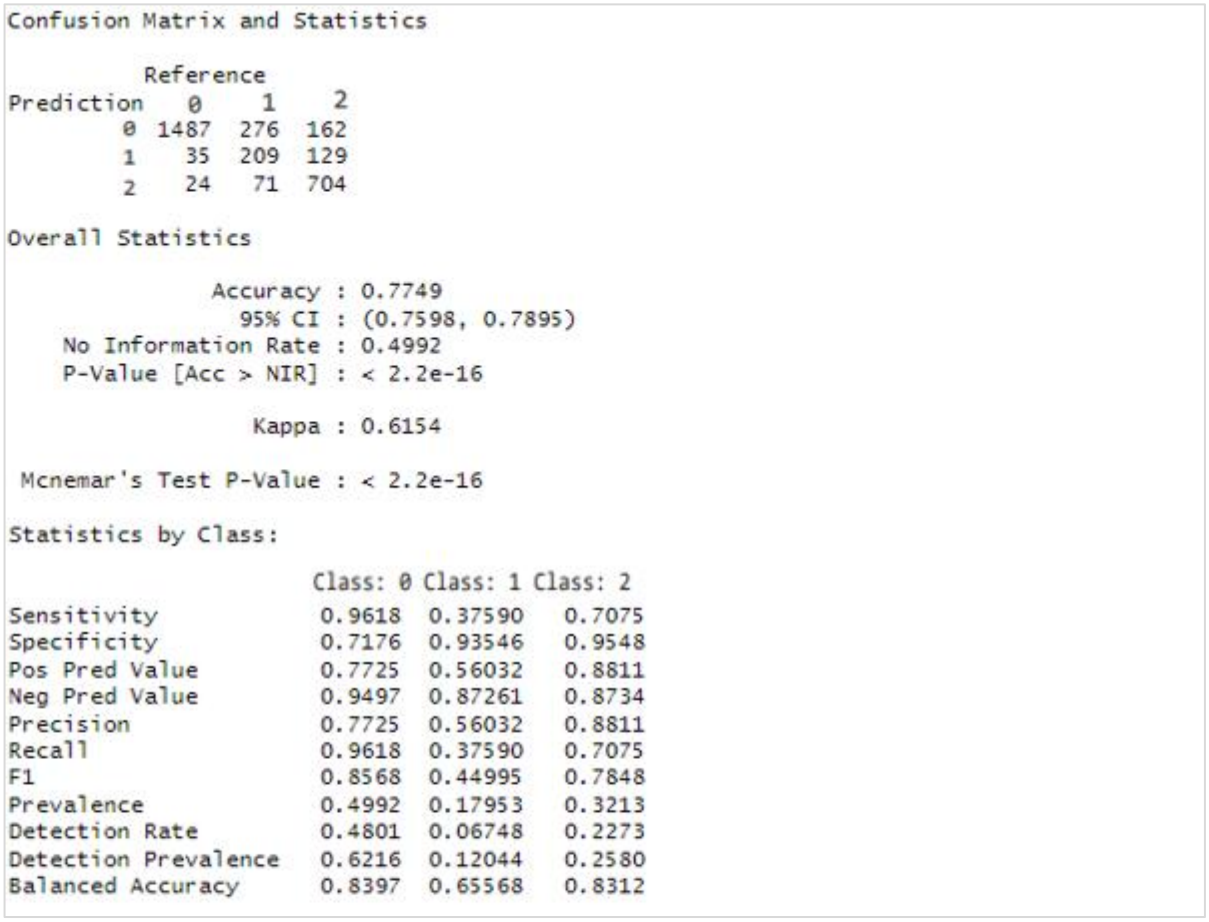
```
# Subsection 4.4:  Building a Decision Tree Model

# Subsection 4.4.1: Initial Model Training

# Develop a Decision Tree model using the training dataset
set.seed(123)
baseline_decision_tree <- rpart(Target ~ ., data = training_data, method = "class")
```

The implementation of the baseline Decision Tree model began by first specifying the seed number, followed by the defining of the decision tree classifier, referred to as baseline_decision_tree in the source code above. The seed number is also defined to ensure that there is reproducibility.

5.4.1.1 Evaluating the Baseline Model’s Performance (Training Data)



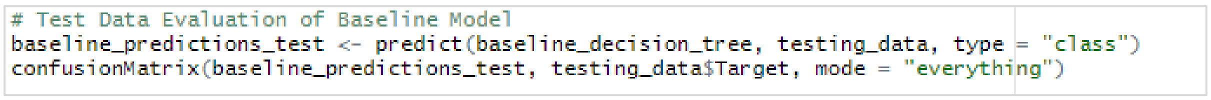
```
# Training Data Evaluation of Baseline Model
baseline_predictions_train <- predict(baseline_decision_tree, training_data, type = "class")
confusionMatrix(baseline_predictions_train, training_data$Target, mode = "everything")
```



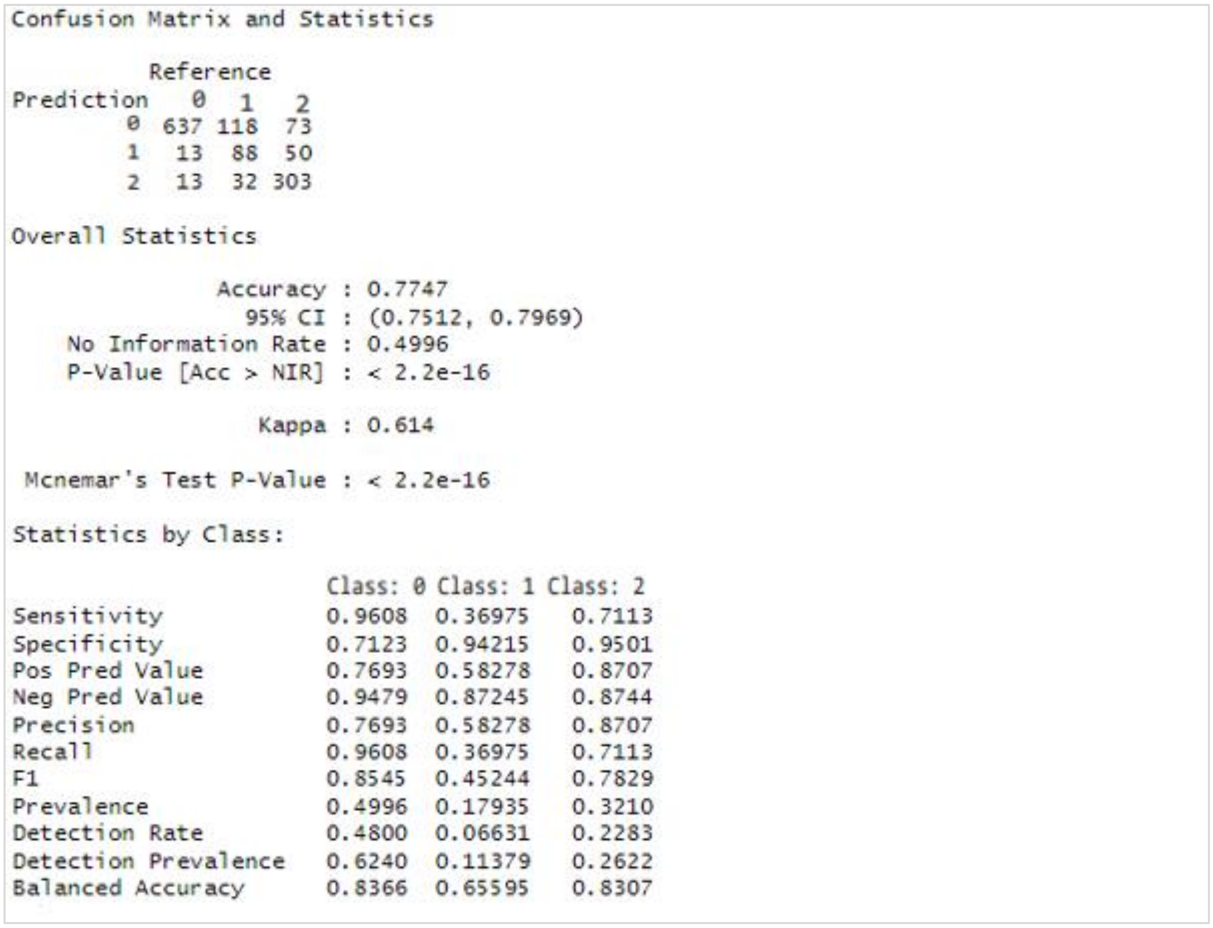
```
Confusion Matrix and Statistics

          Reference
Prediction    0    1    2
         0 1487  276  162
         1   35  209  129
         2   24   71  704

Overall Statistics

               Accuracy : 0.7749
                 95% CI : (0.7598, 0.7895)
    No Information Rate : 0.4992
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6154

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            0.9618  0.37590   0.7075
Specificity            0.7176  0.93546   0.9548
Pos Pred Value         0.7725  0.56032   0.8811
Neg Pred Value         0.9497  0.87261   0.8734
Precision              0.7725  0.56032   0.8811
Recall                 0.9618  0.37590   0.7075
F1                     0.8568  0.44995   0.7848
Prevalence             0.4992  0.17953   0.3213
Detection Rate         0.4801  0.06748   0.2273
Detection Prevalence   0.6216  0.12044   0.2580
Balanced Accuracy      0.8397  0.65568   0.8312
```

Using the classifier that was previously created, the baseline Decision Tree model is then trained on the training dataset. The performance is of the model is then outlined by the confusion matrix. Upon completion of the training process, the performance of the model on the training data is then evaluated. Making reference to the confusion matrix above, the baseline model has attained an accuracy of 77.49% with the training dataset. Precision and Recall across the target classes appeared to be considerably high, with the exception for the Class 1 (Enrolled) where both metrics were considerably lower compared the other two classes. Nevertheless, since the priority is on students at risk of dropping out, the high Precision and Recall values for the Class 2 is favorable.

5.4.1.2 Evaluating the Baseline Model's Performance (Testing Data)



```
# Test Data Evaluation of Baseline Model
baseline_predictions_test <- predict(baseline_decision_tree, testing_data, type = "class")
confusionMatrix(baseline_predictions_test, testing_data$Target, mode = "everything")
```

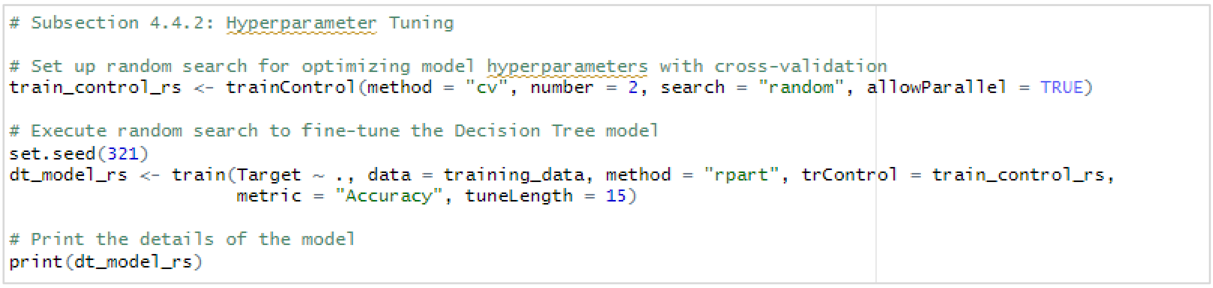


```
Confusion Matrix and Statistics

          Reference
Prediction   0   1   2
         0 637 118  73
         1  13  88  50
         2  13  32 303

Overall Statistics

               Accuracy : 0.7747
                 95% CI : (0.7512, 0.7969)
    No Information Rate : 0.4996
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.614

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            0.9608  0.36975   0.7113
Specificity            0.7123  0.94215   0.9501
Pos Pred Value         0.7693  0.58278   0.8707
Neg Pred Value         0.9479  0.87245   0.8744
Precision              0.7693  0.58278   0.8707
Recall                 0.9608  0.36975   0.7113
F1                     0.8545  0.45244   0.7829
Prevalence             0.4996  0.17935   0.3210
Detection Rate         0.4800  0.06631   0.2283
Detection Prevalence   0.6240  0.11379   0.2622
Balanced Accuracy      0.8366  0.65595   0.8307
```

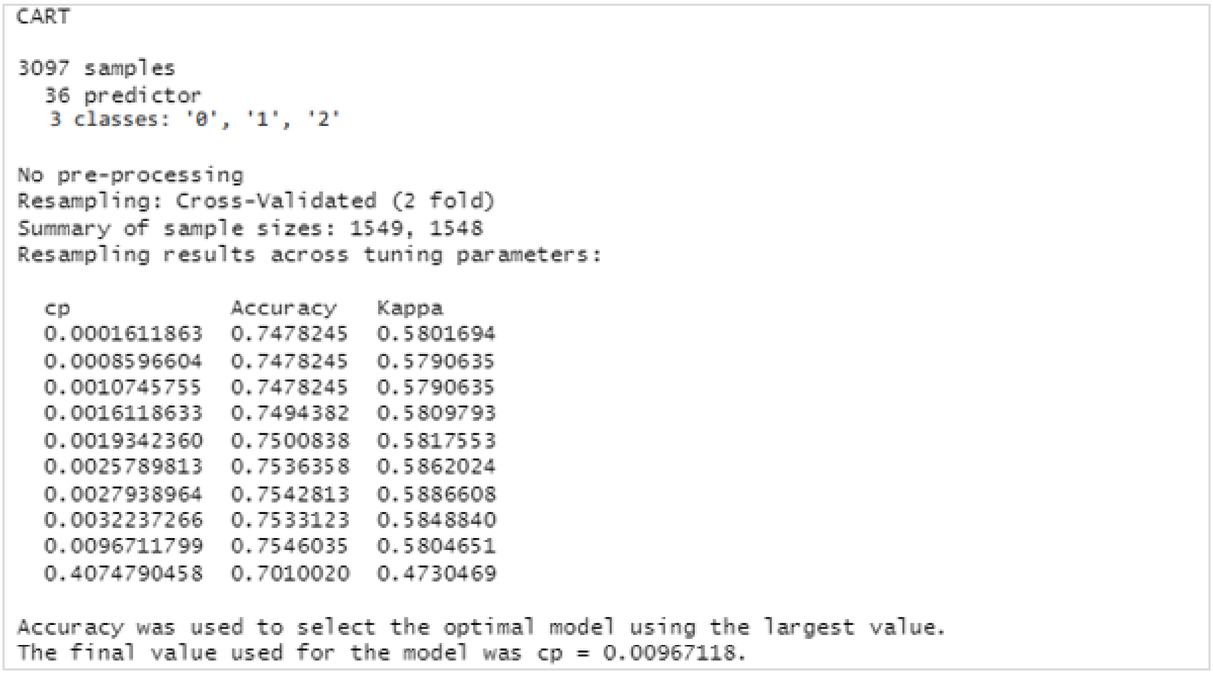
Following the successful training of the model using the training dataset, the trained baseline model is then employed to make predictions with the testing dataset as the input. The performance of the model on the testing dataset could be found in the confusion matrix shown above. Accuracy value stood at 77.47%, a reasonably acceptable accuracy. Precision and Recall values remained reasonably high as well for both the Class 0 (Graduate) and Class 2 (Dropout) . Similar patterns as was noted in the evaluation above, in terms of the lower Precision and Recall values for the Class 1 is observed here too.

5.4.2 Refined Decision Tree Model

5.4.2.1 First Model Refinement Through Hyperparameter Tuning (Random Search)



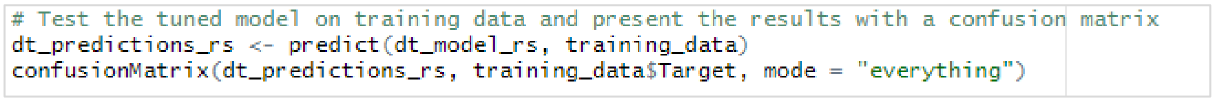
```
# Subsection 4.4.2: Hyperparameter Tuning

# Set up random search for optimizing model hyperparameters with cross-validation
train_control_rs <- trainControl(method = "cv", number = 2, search = "random", allowParallel = TRUE)

# Execute random search to fine-tune the Decision Tree model
set.seed(321)
dt_model_rs <- train(Target ~ ., data = training_data, method = "rpart", trControl = train_control_rs,
                     metric = "Accuracy", tuneLength = 15)

# Print the details of the model
print(dt_model_rs)
```



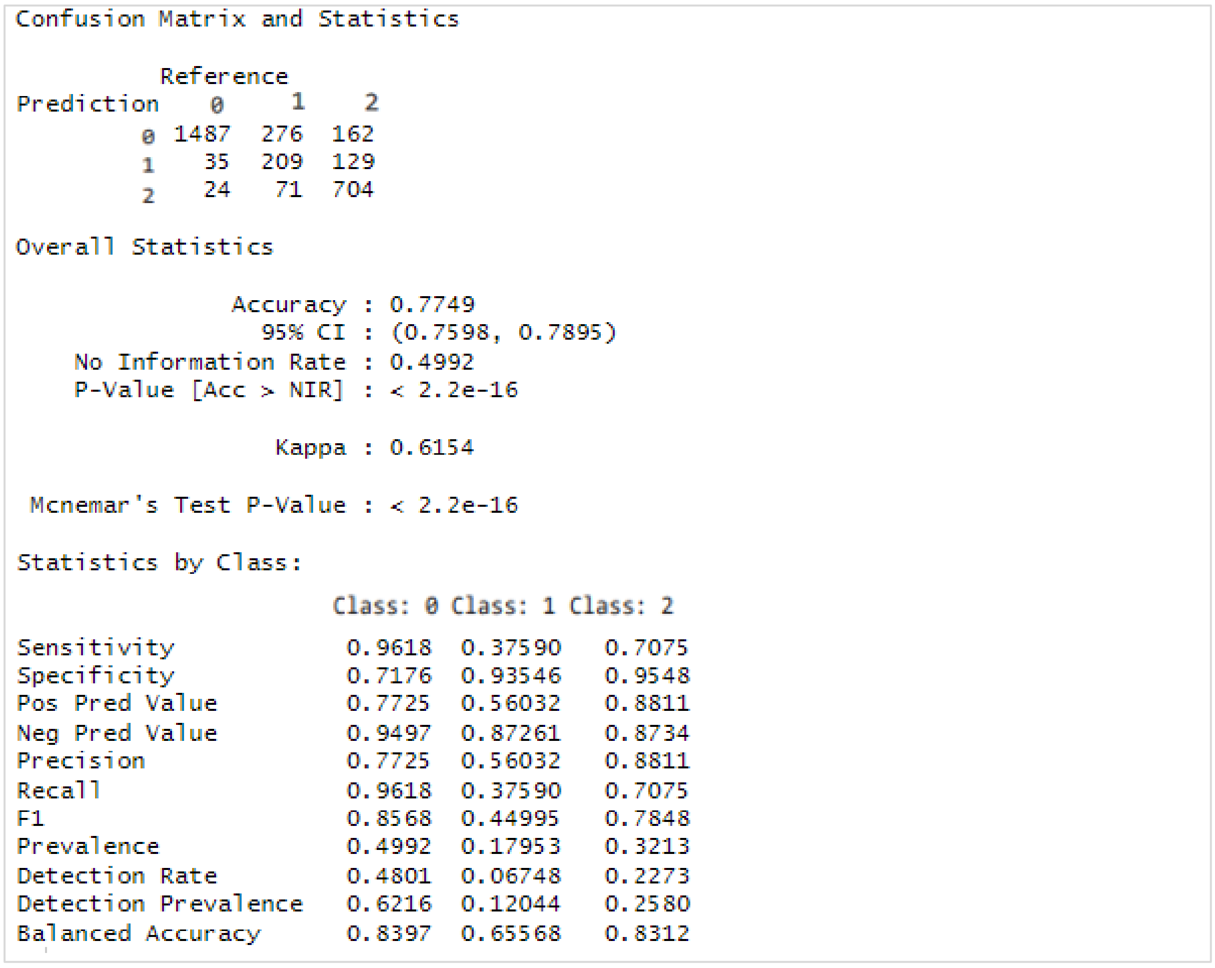
```
CART

3097 samples
  36 predictor
   3 classes: '0', '1', '2'

No pre-processing
Resampling: Cross-Validated (2 fold)
Summary of sample sizes: 1549, 1548
Resampling results across tuning parameters:

  cp            Accuracy   Kappa
  0.0001611863  0.7478245  0.5801694
  0.0008596604  0.7478245  0.5790635
  0.0010745755  0.7478245  0.5790635
  0.0016118633  0.7494382  0.5809793
  0.0019342360  0.7500838  0.5817553
  0.0025789813  0.7536358  0.5862024
  0.0027938964  0.7542813  0.5886608
  0.0032237266  0.7533123  0.5848840
  0.0096711799  0.7546035  0.5804651
  0.4074790458  0.7010020  0.4730469

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was cp = 0.00967118.
```

Similar to the hyperparameter tuning process previously implemented for both XGBoost and Random Forest models, hyperparameters tuning is primarily carried to enhance the model's performance. Consistent with the flow of that which was previously implemented on the Random Forest model, for the hyperparameter tuning process, the random search technique will first be employed, followed by the evaluation of the models based on the hyperparameters identified by the random search technique. Before that, the random search function with 2-folds cross validation was created and assigned to as *train_control.* tuneLength equals to 15 meant that a total of 15 random sets of hyperparameters will be tested for. The hyperparameters of the best performing model will then be saved to *dt_model_rs.* The final output after the process has been completed showed that the optimal "cp" (complexity parameter) should be set at cp = 0.00967118 after multiple cross validation of determining the best model performance. This "cp" value identified is used to finalize the model.

5.4.2.1.1 Evaluating the Refined Model’s Performance (Training Data)



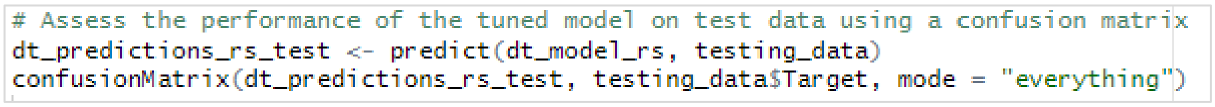
```r
# Test the tuned model on training data and present the results with a confusion matrix
dt_predictions_rs <- predict(dt_model_rs, training_data)
confusionMatrix(dt_predictions_rs, training_data$Target, mode = "everything")
```



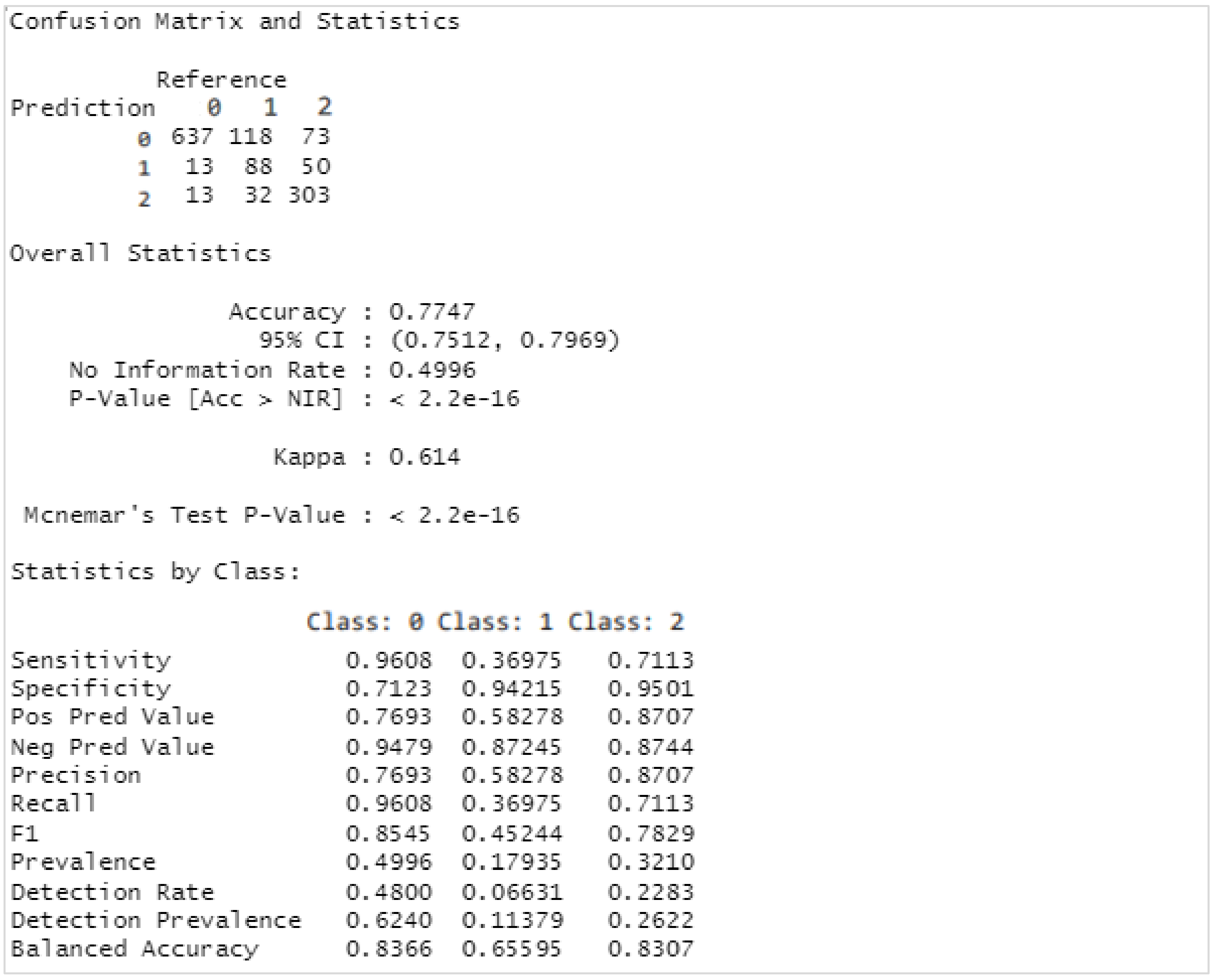
```
Confusion Matrix and Statistics

          Reference
Prediction    0    1    2
         0 1487  276  162
         1   35  209  129
         2   24   71  704

Overall Statistics

               Accuracy : 0.7749
                 95% CI : (0.7598, 0.7895)
    No Information Rate : 0.4992
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6154

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            0.9618  0.37590   0.7075
Specificity            0.7176  0.93546   0.9548
Pos Pred Value         0.7725  0.56032   0.8811
Neg Pred Value         0.9497  0.87261   0.8734
Precision              0.7725  0.56032   0.8811
Recall                 0.9618  0.37590   0.7075
F1                     0.8568  0.44995   0.7848
Prevalence             0.4992  0.17953   0.3213
Detection Rate         0.4801  0.06748   0.2273
Detection Prevalence   0.6216  0.12044   0.2580
Balanced Accuracy      0.8397  0.65568   0.8312
```

Having now identified the best hyperparameters (cp = 0.00967118) according to the Random Search function, the model is now tested on the training dataset using the hyperparameter previously identified. The confusion matrix outlines the refined model’s performance where Accuracy of the model stood at 77.49%% while Precision and Recall remained reasonably satisfactory, with the exception of the those for Class 1.

5.4.2.1.2 Evaluating the Refined Model’s Performance (Testing Data)



```r
# Assess the performance of the tuned model on test data using a confusion matrix
dt_predictions_rs_test <- predict(dt_model_rs, testing_data)
confusionMatrix(dt_predictions_rs_test, testing_data$Target, mode = "everything")
```



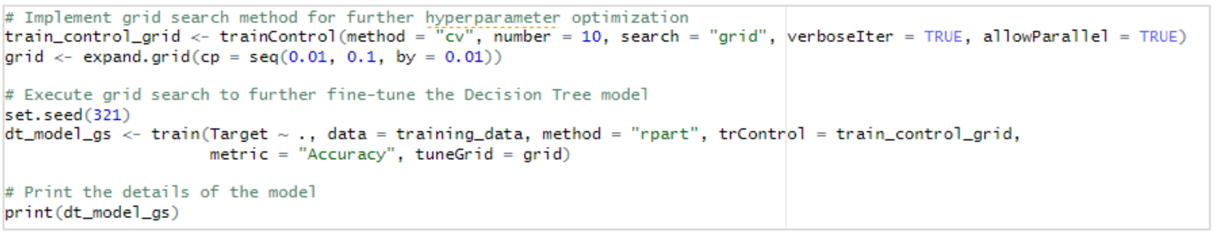
```
Confusion Matrix and Statistics

          Reference
Prediction   0   1   2
         0 637 118  73
         1  13  88  50
         2  13  32 303

Overall Statistics

               Accuracy : 0.7747
                 95% CI : (0.7512, 0.7969)
    No Information Rate : 0.4996
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.614

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            0.9608  0.36975   0.7113
Specificity            0.7123  0.94215   0.9501
Pos Pred Value         0.7693  0.58278   0.8707
Neg Pred Value         0.9479  0.87245   0.8744
Precision              0.7693  0.58278   0.8707
Recall                 0.9608  0.36975   0.7113
F1                     0.8545  0.45244   0.7829
Prevalence             0.4996  0.17935   0.3210
Detection Rate         0.4800  0.06631   0.2283
Detection Prevalence   0.6240  0.11379   0.2622
Balanced Accuracy      0.8366  0.65595   0.8307
```

Similar to the process of the training of the refined model discussed above, except that now predictions are made using the testing dataset, overall accuracy of the model output was slightly lower, standing at 77.47% compared to 77.49%. Similar patterns of the Precision and Recall metrics among the three classes were observed. That is, the Class 1 (Enrolled) once again has had slightly lower values for both the said metrics.

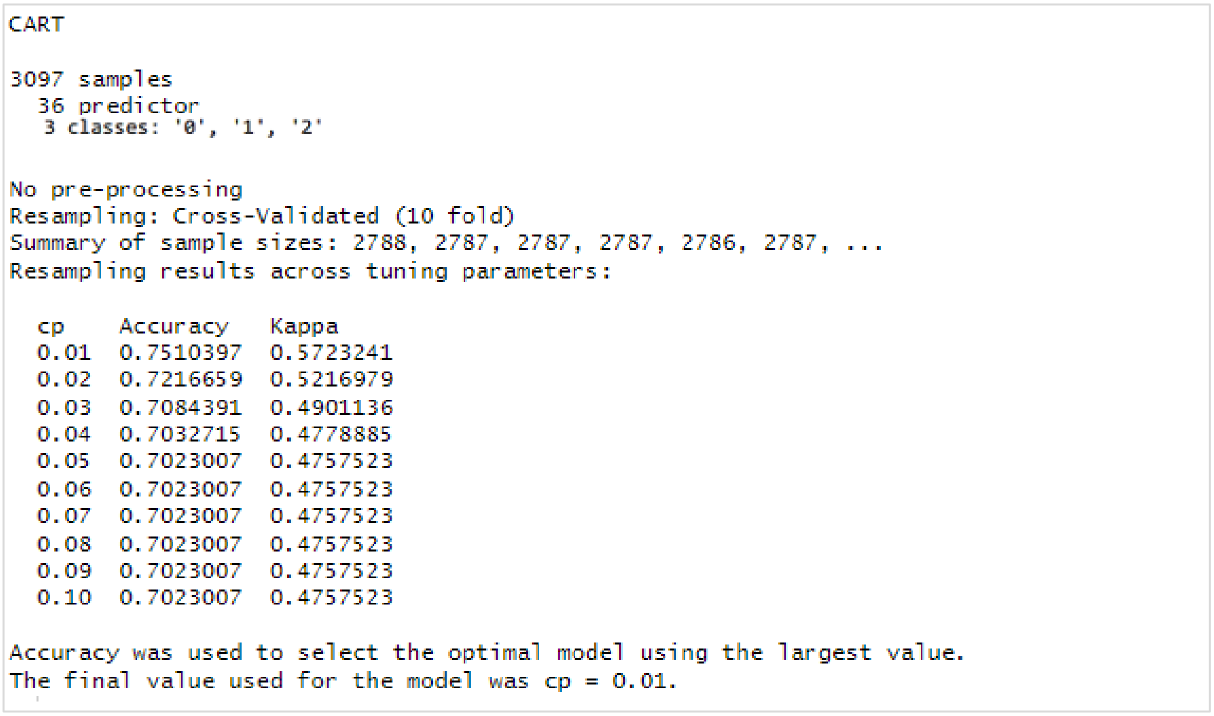
Metrics	**Baseline Model (Decision Tree)**	**Refined Model (Decision Tree)**
Accuracy	77.49%	77.47%
Precision (Class 0)	0.7693	0.7693
Precision (Class 1)	0.5827	0.5827
Precision (Class 2)	0.8707	0.8707
Recall (Class 0)	0.9608	0.9608
Recall (Class 1)	0.3697	0.3697
Recall (Class 2)	0.7113	0.7113

From the summary table above, the results between the baseline model and that of the refined model showed almost perfect similarities, except for the slight reduction in the Accuracy metric of the refined model. This finding suggests that the hyperparameter tuning performed might have had negligible impact on the performance of the model. Given that, further optimization of the hyperparameters will be conducted in the proceeding subsection, to make certain if that was the case.

5.4.2.2 Second Model Refinement Through Hyperparameter Tuning (Grid Search)



```
# Implement grid search method for further hyperparameter optimization
train_control_grid <- trainControl(method = "cv", number = 10, search = "grid", verboseIter = TRUE, allowParallel = TRUE)
grid <- expand.grid(cp = seq(0.01, 0.1, by = 0.01))

# Execute grid search to further fine-tune the Decision Tree model
set.seed(321)
dt_model_gs <- train(Target ~ ., data = training_data, method = "rpart", trControl = train_control_grid,
                     metric = "Accuracy", tuneGrid = grid)

# Print the details of the model
print(dt_model_gs)
```



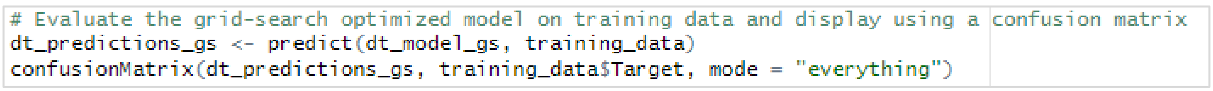
```
CART

3097 samples
  36 predictor
   3 classes: '0', '1', '2'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 2788, 2787, 2787, 2787, 2786, 2787, ...
Resampling results across tuning parameters:

  cp    Accuracy   Kappa
  0.01  0.7510397  0.5723241
  0.02  0.7216659  0.5216979
  0.03  0.7084391  0.4901136
  0.04  0.7032715  0.4778885
  0.05  0.7023007  0.4757523
  0.06  0.7023007  0.4757523
  0.07  0.7023007  0.4757523
  0.08  0.7023007  0.4757523
  0.09  0.7023007  0.4757523
  0.10  0.7023007  0.4757523

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was cp = 0.01.
```

In order to further optimize the predictive power of the model, the next step is to implement a Grid Search by taking the output of hyperparameters obtained from the previously performed Random Search. Recall that the best cp suggested by the Random Search function was 0.00967118. While it is supposedly the case that the range of the hyperparameter values to be carried out by the Grid Search function must contain this value, the cp value was increased to 0.01 instead, for the purpose of obtaining a more obvious result. The grid would start from cp equals to 0.01 and end at cp equals to 0.1, with an increment of 0.01 cp values every split.

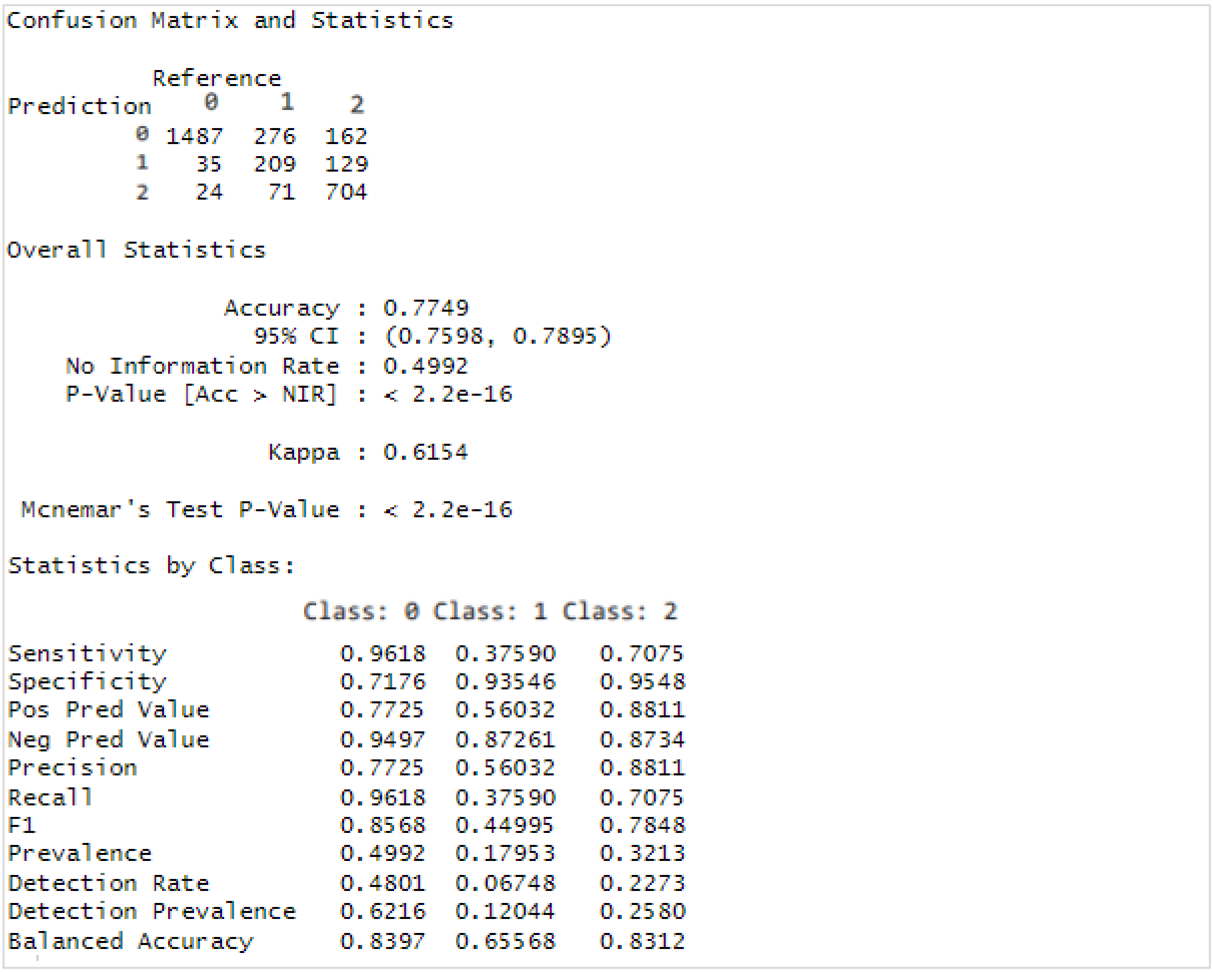
Before defining the grid range however, the training control function must be specified first, where a 10-folds cross validation is to be implemented during the training process.

Once the train control and grid functions are defined, the seed number is then set to “321” for reproducibility purposes, and the model is subsequently trained using the specified grid on the training dataset.

5.4.2.2.1 Evaluating the Refined Model’s Performance (Training Data)



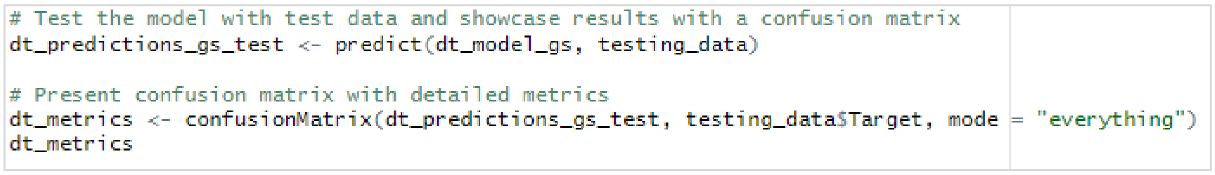
```
# Evaluate the grid-search optimized model on training data and display using a confusion matrix
dt_predictions_gs <- predict(dt_model_gs, training_data)
confusionMatrix(dt_predictions_gs, training_data$Target, mode = "everything")
```



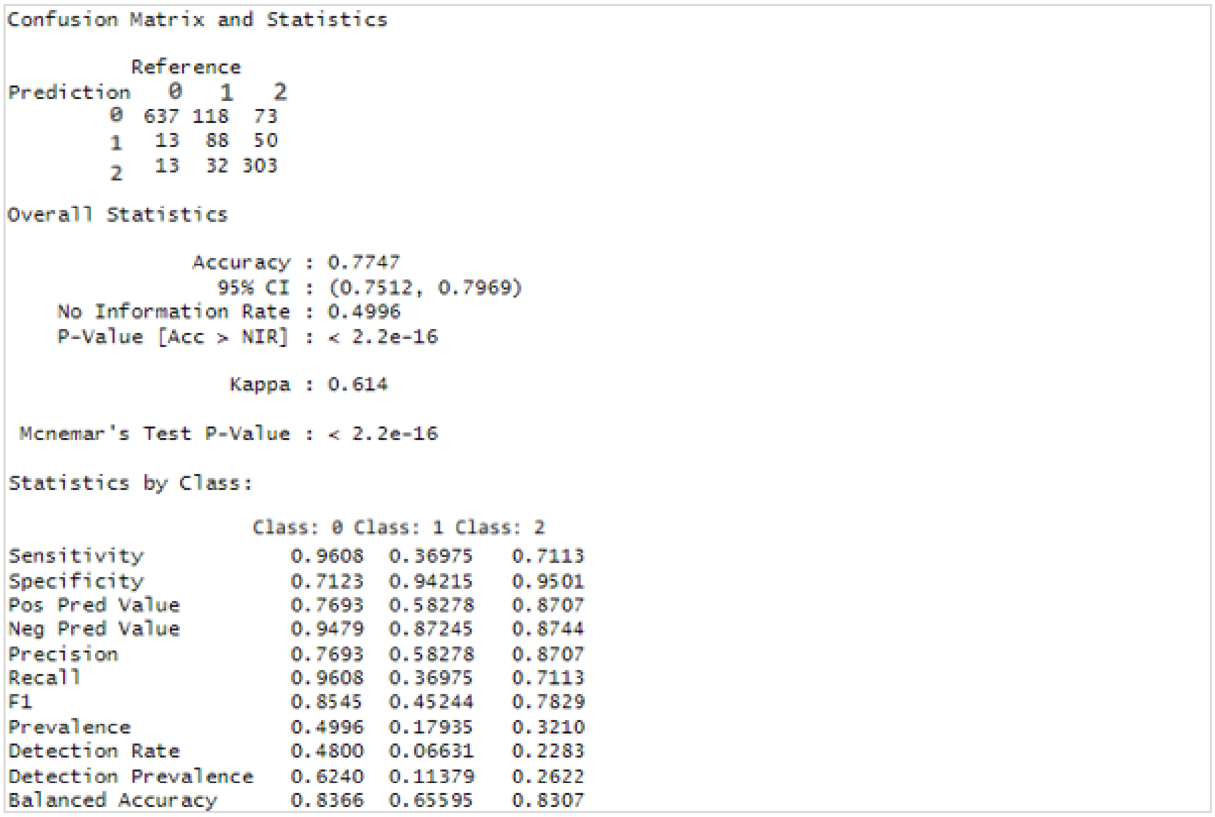
```
Confusion Matrix and Statistics

          Reference
Prediction    0    1    2
         0 1487  276  162
         1   35  209  129
         2   24   71  704

Overall Statistics

               Accuracy : 0.7749
                 95% CI : (0.7598, 0.7895)
    No Information Rate : 0.4992
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6154

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            0.9618  0.37590   0.7075
Specificity            0.7176  0.93546   0.9548
Pos Pred Value         0.7725  0.56032   0.8811
Neg Pred Value         0.9497  0.87261   0.8734
Precision              0.7725  0.56032   0.8811
Recall                 0.9618  0.37590   0.7075
F1                     0.8568  0.44995   0.7848
Prevalence             0.4992  0.17953   0.3213
Detection Rate         0.4801  0.06748   0.2273
Detection Prevalence   0.6216  0.12044   0.2580
Balanced Accuracy      0.8397  0.65568   0.8312
```

The model’s performance recorded a 77.49% accuracy in its predictions. Similar patterns of the Precision and Recall metrics among the three classes were observed. That is, both metrics remained high for the Class 0 (Graduate) and 2 (Dropout), while both metrics remained slightly lower for the Class 1 (Enrolled).

5.4.2.2.2 Evaluating the Refined Model’s Performance (Testing Data)



```
# Test the model with test data and showcase results with a confusion matrix
dt_predictions_gs_test <- predict(dt_model_gs, testing_data)

# Present confusion matrix with detailed metrics
dt_metrics <- confusionMatrix(dt_predictions_gs_test, testing_data$Target, mode = "everything")
dt_metrics
```



```
Confusion Matrix and Statistics

          Reference
Prediction   0   1   2
         0 637 118  73
         1  13  88  50
         2  13  32 303

Overall Statistics

               Accuracy : 0.7747
                 95% CI : (0.7512, 0.7969)
    No Information Rate : 0.4996
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.614

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            0.9608  0.36975   0.7113
Specificity            0.7123  0.94215   0.9501
Pos Pred Value         0.7693  0.58278   0.8707
Neg Pred Value         0.9479  0.87245   0.8744
Precision              0.7693  0.58278   0.8707
Recall                 0.9608  0.36975   0.7113
F1                     0.8545  0.45244   0.7829
Prevalence             0.4996  0.17935   0.3210
Detection Rate         0.4800  0.06631   0.2283
Detection Prevalence   0.6240  0.11379   0.2622
Balanced Accuracy      0.8366  0.65595   0.8307
```

When the model is being tested using the testing dataset, its accuracy decreased to 77.47%, a reasonably high accuracy still. While the Precision and Recall values among the classes are varying, the priority dropout class (Class 2) recorded considerably high values. Such is the case for the Class 0 (Graduate) as well.

```
# Measure model performance using Area Under the Curve

# Create a numeric vector of the true class levels
true_classes <- as.numeric(testing_data$Target) - 1

# Predict probabilities for each class
dt_prob_predictions_test <- predict(dt_model_gs, testing_data, type = "prob")

# Calculate the AUC for each class using a one-vs-all approach
auc_values <- sapply(seq_along(dt_prob_predictions_test[1, ]), function(i) {
  roc_auc <- roc(response = ifelse(true_classes == (i-1), 1, 0),
                 predictor = as.numeric(dt_prob_predictions_test[, i]))
  auc(roc_auc)
})

# Output the AUC values for each class
print(auc_values)
```

```
[1] 0.8716211 0.7432634 0.8971930
```

The AUC values for the Class 0, 1 and 2 are 0.8716, 0.7432, and 0.8971 respectively.

```
# Calculate the average AUC across all classes
avg_auc <- mean(auc_values)
print(avg_auc)
```

```
[1] 0.8373592
```

The model's average AUC value is recorded at 0.8373, implying that the model is proficient in discriminating between the classes. Higher AUC values are desired where the closer the value is to 1, the greater is the discriminatory power of the model.

Metrics	**Refined Model using Random Search (Decision Tree)**	**Refined Model using Grid Search (Decision Tree)**
Accuracy	77.47%	77.47%
Precision (Class 0)	0.7693	0.7693
Precision (Class 1)	0.5827	0.5827
Precision (Class 2)	0.8707	0.8707
Recall (Class 0)	0.9608	0.9608
Recall (Class 1)	0.3697	0.3697
Recall (Class 2)	0.7113	0.7113

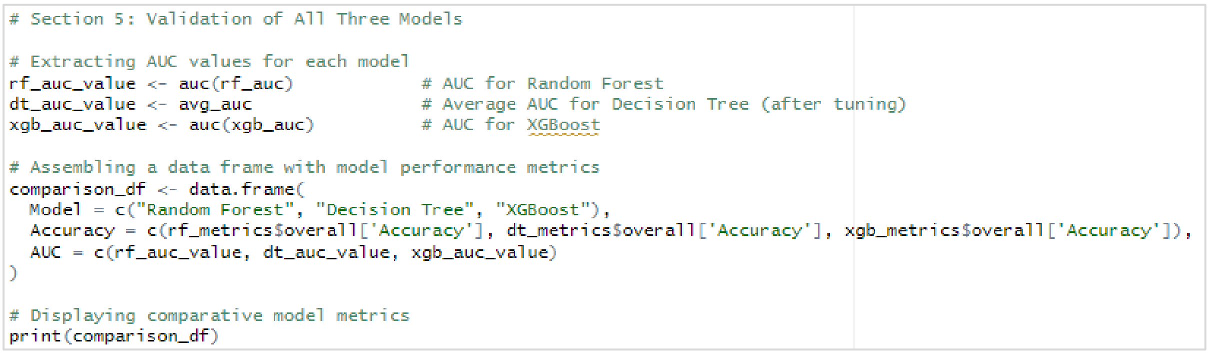
Making reference to the table above, interestingly, all aspects of the performance metrics for both refined models of Random Search and Grid Search respectively have had same values.

This suggests that there is no difference at all between the choice of either using Random Search or Grid Search when performing hyperparameter tuning, and when in the context of using the Decision Tree model.

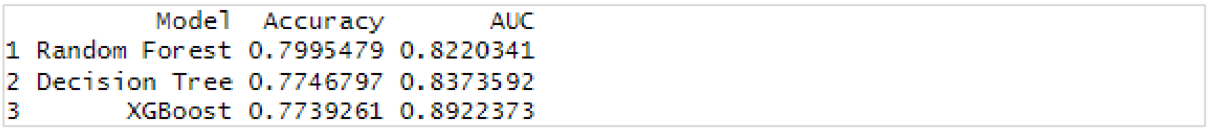
6.0 Model Evaluation

6.1 Comparison Across Implemented Models

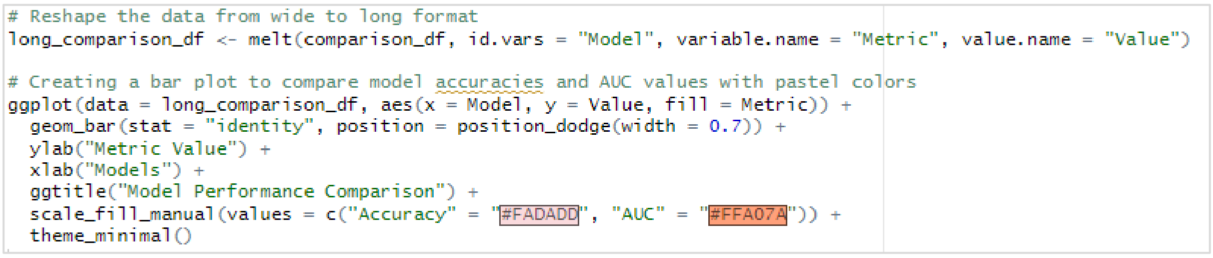
In the section, comparisons will be made across all of the performances of the models which are implemented within this study.



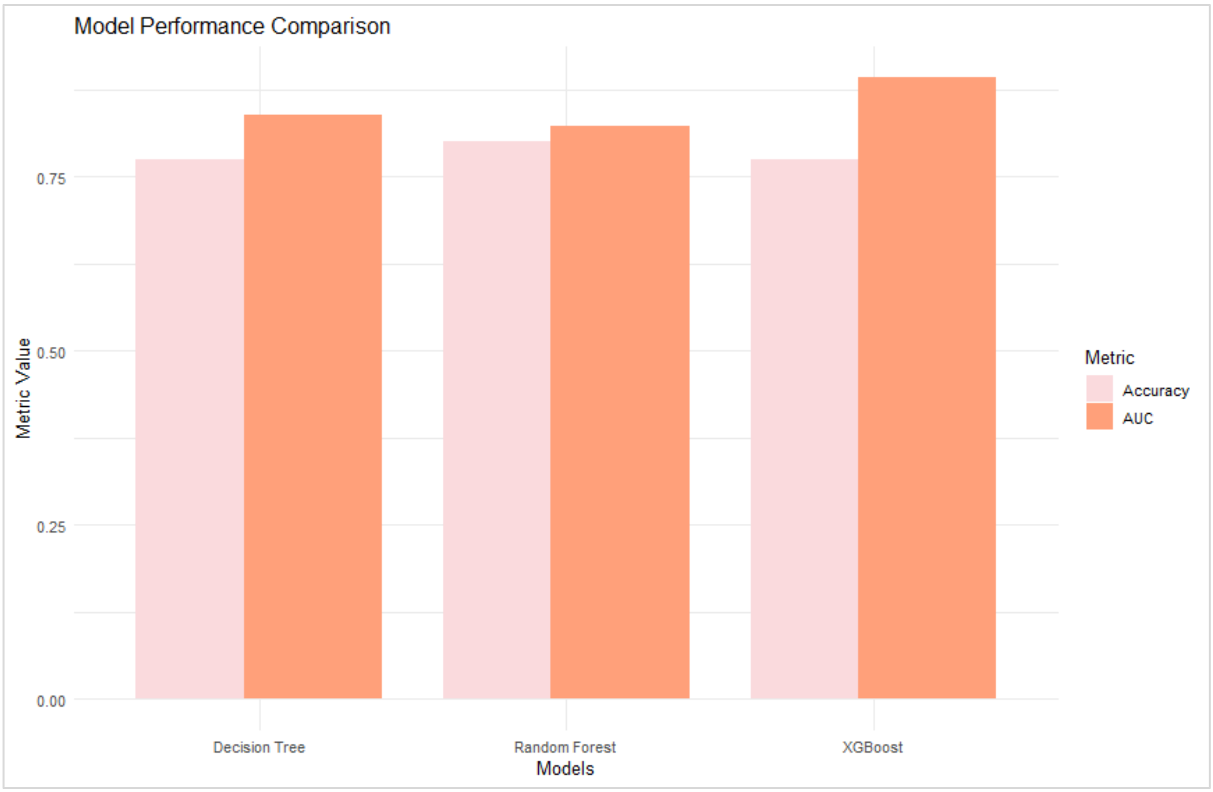
```
# Section 5: Validation of All Three Models

# Extracting AUC values for each model
rf_auc_value <- auc(rf_auc)              # AUC for Random Forest
dt_auc_value <- avg_auc                  # Average AUC for Decision Tree (after tuning)
xgb_auc_value <- auc(xgb_auc)            # AUC for XGBoost

# Assembling a data frame with model performance metrics
comparison_df <- data.frame(
  Model = c("Random Forest", "Decision Tree", "XGBoost"),
  Accuracy = c(rf_metrics$overall['Accuracy'], dt_metrics$overall['Accuracy'], xgb_metrics$overall['Accuracy']),
  AUC = c(rf_auc_value, dt_auc_value, xgb_auc_value)
)

# Displaying comparative model metrics
print(comparison_df)
```



```
          Model  Accuracy       AUC
1 Random Forest 0.7995479 0.8220341
2 Decision Tree 0.7746797 0.8373592
3       XGBoost 0.7739261 0.8922373
```



```
# Reshape the data from wide to long format
long_comparison_df <- melt(comparison_df, id.vars = "Model", variable.name = "Metric", value.name = "Value")

# Creating a bar plot to compare model accuracies and AUC values with pastel colors
ggplot(data = long_comparison_df, aes(x = Model, y = Value, fill = Metric)) +
  geom_bar(stat = "identity", position = position_dodge(width = 0.7)) +
  ylab("Metric Value") +
  xlab("Models") +
  ggtitle("Model Performance Comparison") +
  scale_fill_manual(values = c("Accuracy" = "#FADADD", "AUC" = "#FFA07A")) +
  theme_minimal()
```



From the table and grouped bar chart showed above, it is evident that the XGBoost model has attained the highest AUC values, but has lost out to the Random Forest model in terms of Accuracy. That said, while the Accuracy and AUC metrics serve as a good overview of the performances of the models, it is however important as well to delve deeper into the class-specific metrics to have a more granular perspective of the model's proficiency.

	Precision			Recall		
	Class 0	**Class 1**	**Class 2**	**Class 0**	**Class 1**	**Class 2**
XGBoost	0.8785	0.4620	0.8889	0.8401	0.6639	0.7324
Random Forest	0.8056	0.6557	0.8305	0.9623	0.3361	0.8052
Decision Tree	0.7693	0.5827	0.8707	0.9608	0.3697	0.7113

The table above shows a more granular overview of the all three of the implemented model's class-specific Precision and Recall metric. Recall that a high Precision and Recall is desired. As was mentioned several times within this report, the primary focus will be on the groups of students who are at risk of dropping out of school. The Class 2 represents such category of students. In terms of Precision, XGBoost has managed to obtain the highest value among the rest of the models, in the context of the Dropout class. In terms of Recall however, XGBoost lost out quite a bit to the Random Forest model where its Recall value stood at 0.8052. Random Forest on the other hand has had got the highest Recall value within the Class 2, while it took second place in terms of Precision.

Essentially, if one model had to be picked among the rest of the two models, and since the primary priority is to accurately predict students at risk of dropping out, XGBoost would then be the best choice. There are several reasonings to justify for it and they are as follows:

1. Given that XGBoost has the highest precision for the Dropout category (0.8889), it then meant that it is the best at making sure that when it predicts that a student will drop out, the student is indeed likely to drop out. The
2. While not being the one with the highest scores, a Recall score of 0.7324 suggests that XGBoost does have a strong Recall still. This implies that it is still quite good proficient at identifying actual dropouts. While it may not capture as many dropouts as the Random Forest model (which has a recall 0f 0.8052 for Class 2), XGBoost remained superior because there is a balance strike between Recall and Precision.
3. Also, with regards to Class 1 (Enrolled students), Precision is notably higher in XGBoost compared to both its counterparts. This higher value suggests that it is better at correctly identifying students who continue their studies.
4. In terms of Class 0 (Graduate), even though XGBoost's Recall value is slightly lower than its two counterparts, its significantly higher Precision does makes up for the lower

Recall value. A high Precision is necessary because it then means that there will be fewer false positive in predicting graduates.

To summarize, the proposition for XGBoost to be chosen over the two other models is rooted in the fact that it was able to provide a more balanced approach, that is effectively identifying dropouts without a high rate of false positives, and is also able to perform well in distinguishing between the other classes (the highest AUC score). In essence, this balance is instrumental in the context of practical application, because it helps in minimizing the risk of wrongly classifying students, which could come at the cost of uncalled intervention or worst still, missed opportunities for support.

6.2 Comparisons Among Peer Models

Comparisons of models' performances to that of existing models in the similar domain, or if possible, with the exact same dataset would serves to be a benchmark to ascertain the proficiencies of the models implements in solving the task at hand. However, before that, it is important to note that the for the two peers' models listed in the table below, comparisons are not entirely one-to-one correspondence. This is because even though the dataset used were of exactly the same, data as well as the models could all be handled differently.

Sources	Highest Accuracy	Model
This Study	77.39%	XGBoost
(Martins et al., 2021)	73%	XGBoost
(Martins et al., 2023)	74.8%	SVMSMOTE + RF

After having analyzed and reviewed literature work done by the two authors mentioned above in the table, several observations were made:

1. Superiority of the Model of this Study: While both of the literature work is working on the same dataset as this study, with the same number of observations and attributes, the model implemented in this study was still able to distinguish itself from the models initially implemented by other academics. Such distinction could be due to the different approaches when addressing the similar task at hand.
2. Comprehensive Initial Data Exploration: The initial data exploration which was carried out within the study enabled for the detection of entries which did not make sense. This then allowed for the proper treatment of these problematic entries. Had there not been

any detection of these entries, results of the analysis would then face the potential risk of being distorted from its true meaning. In both the literature work outlined in the table above, it is important to note that no mentions were made with regards to the problematic entries highlighted in this study.

3. Effective Data Preprocessing: Another possible reason for the disparity observed in the performance of models could due to the different approached when preprocessing the dataset. For instance, in their work, and even though sharing the exact same dataset (4424 observations) as this study, Martins et al. (2021) decided that the dataset was imbalanced. Hence, they proceeded to perform the Synthetic Minority Over Sampling Technique (SMOTE) to deal with their dataset.

7.0 Analysis and Recommendations

7.1 Critical Evaluations of Model Outcomes

Even though the recommended model from this study is the XGBoost model, it is important to take note of some of the critics towards the model.

1. **Generalizability:** Even though the XGBoost within this study was found to have had exemplary performance as the model was trained and evaluated using the dataset, the model's performance when unseen data is from another geographical context remains a mystery. While certain models may work well with certain data, there is really no one-size-fits-all solution. Hence, before officially deploying this model in a new context, it is suggested to first retrain and validate the model with local data to ensure that its predictive power holds true across different settings. This process helps to facilitate for the adaptation of the model to the specifics of the new data, which may be different from the initial dataset for reasons of economical, educational, regional, or cultural factors.
2. **Complexity of Hyperparameters:** There are a multitude of hyperparameters that belong to the model, thereby complicating the tuning process. The challenge comes from having enough domain-specific knowledge to adequately tune these parameters. The tuning of these parameters requires meticulous experimentation and validation in order to appropriately optimize the model's performance. Attempts of such in this report is serves only as a naïve starting point for what appears to be just the tip of the iceberg in hyperparameter tuning.
3. Interpretability of the Model: Even though the XGBoost model is equipped for feature importance analysis, interpretation of the individual decisions trees and the overall ensemble model could be a daunting task, more so if stakeholders are keener with straightforward explanations over the complicated ensemble methods. While this study attempts to provide as much clarity as possible to the readers, interpretation of the models in greater details were unfortunately not adequately addressed in this report.

7.2 Surprises or Anomalies

While working on the data analysis of the given classification task, there were several surprises or anomalies encountered. These surprises include:

1. **Computationally Intensive Aspects:** When performing the necessary cleaning of the dataset, the employment of the missForest algorithm for imputation purposes was unexpectedly time-consuming. That is, although the dataset in this study was considered as relatively small in the context of big data, considerable computational resources and time were already greatly taxed. This emphasizes the intensive computational demands of the missForest algorithm, which can be exacerbated as dataset become larger. Given that, there is a need for meticulous consideration in terms of computational efficiency, more so if the dataset employed is large, or when time or computing power is limited.
2. **Anomalies in the Dataset:** The many entries where attributes pertaining the Curricular units for both semesters were recorded as 0, was indeed an interesting encounter. Considerable research and reading had to be done, especially in the context of where and the dataset was sourced, before arriving at the conclusion that these entries were indeed illogical.
3. **Attributes Recognized as Highly Important:** Results of the feature important analysis which indicated that attribute namely Father.qualification and Tuition.fees.up.to.date were interesting findings which was not at all expected. One would have expected factors which are more traditionally emphasized such as peer influence or a student's attitude and motivation to be considered as highly influential instead of the two which were identified. In essence, this proposes that socioeconomic factors as well as monetary factors may play a more instrumental role than previously assumed.

8.0 Conclusion

Upon completion of the data analysis process, several invaluable takeaways were obtained. While piecing together the source code and ensuring that it is functioning well and that there is coherency to it is a challenge on itself, knowing how to interpret the outputs of these source codes is however another aspect of the challenge. A coherent and logical workflow in data analysis would begin with the dataset, followed by an initial data exploration, data preprocessing, exploratory data analysis, data preparation for modelling, model implementation and lastly, the workflow ends with model evaluation. A comprehensive initial data exploration and subsequently a thorough exploratory data analysis is pivotal in ensuring that analysis carried out is impactful, efficient and meaningful. Future work includes extending the exploration of the hyperparameters tuning process for each model of this study, the refinement of the models discussed such as the implementation of ensemble methods with other algorithms, and lastly the exploration of potential feature engineering of the attributes within the dataset to enable for new patterns and insights to be discovered.

References

Al-Razgan, M., Al-Khalifa, A S., & Al-Khalifa, H S. (2013, December 15). Educational Data Mining: A Systematic Review of the Published Literature 2006-2013. Lecture notes in electrical engineering, 711-719. https://doi.org/10.1007/978-981-4585-18-7_80

Capuno, R. M. M., Ferrer, C. J. M., Manaloto, B. T. L., & Villafria, S. R. (2023, March 7-9). *Towards predicting student's dropout in higher education using supervised machine learning techniques*. [Paper presentation]. International Conference on Industrial Engineering and Operations Management, Manila, Philippines. https://ieomsociety.org/proceedings/2023manila/654.pdf?CFID=44820ce9-bafa-48e3-9138-23a1b6bced2b&CFTOKEN=0

Dasi, H., & Kanakala, S. (2022). Student dropout prediction using machine learning techniques. *International Journal of Intelligent Systems and Applications in Engineering, 10*(4), 408–414. https://ijisae.org/index.php/IJISAE/article/view/2276

Jiménez, O., Jesús, A., & Wong, L. (2023, May, 24-26). *Model for the prediction of dropout in higher education in Peru applying machine learning algorithms: Random Forest, Decision Tree, Neural Network and Support Vector Machine*. [Paper presentation]. 33rd Conference of Open Innovations Association (FRUCT), Zilina, Slovakia. doi: 10.23919/FRUCT58615.2023.10143068.

Martins, M. V., Baptista, L., Machado, J., & Realinho, V. (2023). Multi-Class Phased Prediction of Academic Performance and Dropout in Higher Education. *Applied Sciences (Switzerland)*, *13*(8). https://doi.org/10.3390/app13084702

Martins, M. V., Baptista, L., Machado, J., & Realinho, V. (2023). Multi-Class Phased Prediction of Academic Performance and Dropout in Higher Education. *Applied Sciences, 13*(8), 4702. https://doi.org/10.3390/app13084702

Martins, M. V., Tolledo, D., Machado, J., Baptista, L. M. T., & Realinho, V. (2021). Early Prediction of student's Performance in Higher Education: A Case Study. *Advances in Intelligent Systems and Computing*, *1365 AIST*, 166–175. https://doi.org/10.1007/978-3-030-72657-7_16/COVER

Martins, M. V., Tolledo, D., Machado, J., Baptista, L., & Valentim Realinho. (2021). early prediction of student’s performance in higher education: A case study. *Advances in Intelligent Systems and Computing*, 166–175. https://doi.org/10.1007/978-3-030-72657-7_16

Realinho, V., Machado, J., Baptista, L., & Martins, M. V. (2022). Predicting Student Dropout and Academic Success. *Data*, *7*(11), 146. https://doi.org/10.3390/data7110146

UCI Machine Learning Repository. (2021). *Predict students' dropout and academic success*. https://archive.ics.uci.edu/dataset/697/predict+students+dropout+and+academic+success

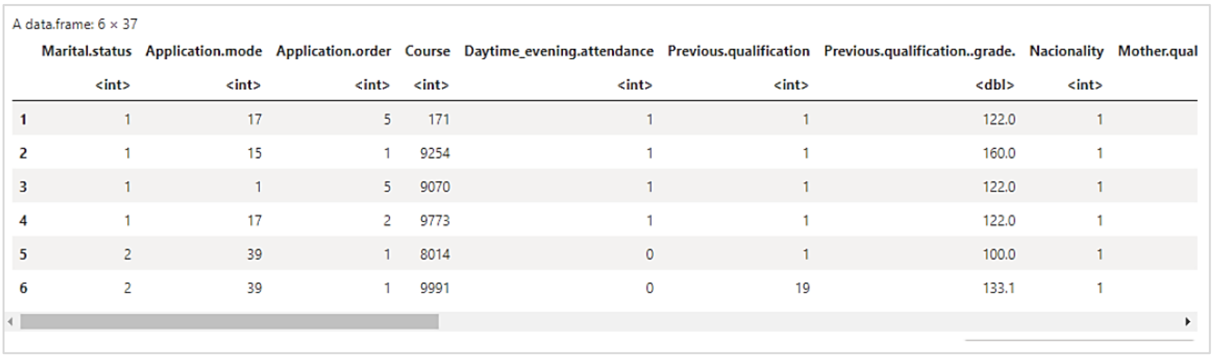
Appendix

```
# Section 1: Setting Up the Environment

# Subsection 1.1: Installing and Loading the Necessary Packages

# Install packages
install.packages("dplyr", dependencies = TRUE)
install.packages("missForest", dependencies = TRUE)
install.packages("DataExplorer", dependencies = TRUE)
install.packages("caret", dependencies = TRUE)
install.packages("ROCR", dependencies = TRUE)
install.packages("ggplot2", dependencies = TRUE)
install.packages("caTools", dependencies = TRUE)
install.packages("xgboost", dependencies = TRUE)
install.packages("randomForest", dependencies = TRUE)
install.packages("pROC", dependencies = TRUE)
install.packages("MLmetrics", dependencies = TRUE)
install.packages("rpart", dependencies = TRUE)
install.packages("reshape2")
```

```
# Load the installed packages
library(dplyr)
library(missForest)
library(DataExplorer)
library(caret)
library(ROCR)
library(ggplot2)
library(caTools)
library(xgboost)
library(randomForest)
library(pROC)
library(MLmetrics)
library(rpart)
library(reshape2)
```

```
# Subsection 1.2: Importing the dataset
ds <- read.csv("predict-student-dropout.csv")
```

```
# Section 2.0: Data Preprocessing

# Subsection 2.1 Initial exploration of the dataset

# Show initial entries of the dataset for a quick preview
head(ds)
```



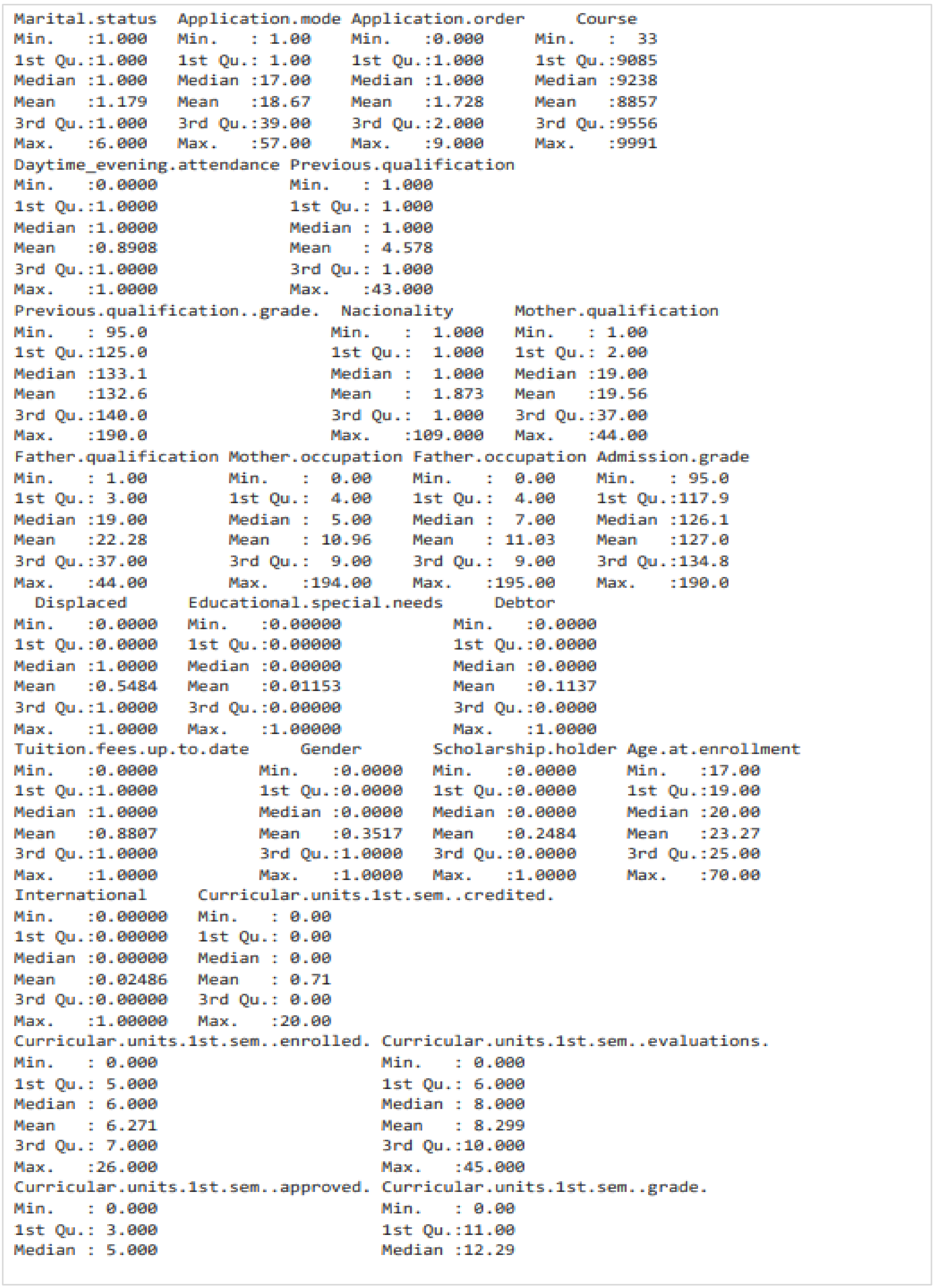
A data.frame: 6 × 37

	Marital.status	Application.mode	Application.order	Course	Daytime_evening.attendance	Previous.qualification	Previous.qualification..grade.	Nacionality	Mother.qual
	<int>	<int>	<int>	<int>	<int>	<int>	<dbl>	<int>	
1	1	17	5	171	1	1	122.0	1	
2	1	15	1	9254	1	1	160.0	1	
3	1	1	5	9070	1	1	122.0	1	
4	1	17	2	9773	1	1	122.0	1	
5	2	39	1	8014	0	1	100.0	1	
6	2	39	1	9991	0	19	133.1	1	

```
# Examine the dataset's structure, including data types and column information
str(ds)
```

```
'data.frame':   4424 obs. of  37 variables:
 $ Marital.status                              : int  1 1 1 1 2 2 1 1 1 1 ...
 $ Application.mode                            : int  17 15 1 17 39 39 1 18 1 1
...
 $ Application.order                           : int  5 1 5 2 1 1 1 4 3 1 ...
 $ Course                                      : int  171 9254 9070 9773 8014 9
991 9500 9254 9238 9238 ...
 $ Daytime_evening.attendance                  : int  1 1 1 1 0 0 1 1 1 1 ...
 $ Previous.qualification                      : int  1 1 1 1 1 19 1 1 1 1 ...
 $ Previous.qualification..grade.              : num  122 160 122 122 100 ...
 $ Nacionality                                 : int  1 1 1 1 1 1 1 1 62 1 ...
 $ Mother.qualification                        : int  19 1 37 38 37 37 19 37 1
1 ...
 $ Father.qualification                        : int  12 3 37 37 38 37 38 37 1
19 ...
 $ Mother.occupation                           : int  5 3 9 5 9 9 7 9 9 4 ...
 $ Father.occupation                           : int  9 3 9 3 9 7 10 9 9 7 ...
 $ Admission.grade                             : num  127 142 125 120 142 ...
 $ Displaced                                   : int  1 1 1 1 0 0 1 1 0 1 ...
 $ Educational.special.needs                   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ Debtor                                      : int  0 0 0 0 0 1 0 0 0 1 ...
 $ Tuition.fees.up.to.date                     : int  1 0 0 1 1 1 1 0 1 0 ...
 $ Gender                                      : int  1 1 1 0 0 1 0 1 0 0 ...
 $ Scholarship.holder                          : int  0 0 0 0 0 0 1 0 1 0 ...
 $ Age.at.enrollment                           : int  20 19 19 20 45 50 18 22 2
1 18 ...
 $ International                               : int  0 0 0 0 0 0 0 0 1 0 ...
 $ Curricular.units.1st.sem..credited.         : int  0 0 0 0 0 0 0 0 0 0 ...
 $ Curricular.units.1st.sem..enrolled.         : int  0 6 6 6 6 5 7 5 6 6 ...
 $ Curricular.units.1st.sem..evaluations.      : int  0 6 0 8 9 10 9 5 8 9 ...
 $ Curricular.units.1st.sem..approved.         : int  0 6 0 6 5 5 7 0 6 5 ...
 $ Curricular.units.1st.sem..grade.            : num  0 14 0 13.4 12.3 ...
 $ Curricular.units.1st.sem..without.evaluations.: int  0 0 0 0 0 0 0 0 0 0 ...
 $ Curricular.units.2nd.sem..credited.         : int  0 0 0 0 0 0 0 0 0 0 ...
 $ Curricular.units.2nd.sem..enrolled.         : int  0 6 6 6 6 5 8 5 6 6 ...
 $ Curricular.units.2nd.sem..evaluations.      : int  0 6 0 10 6 17 8 5 7 14
...
 $ Curricular.units.2nd.sem..approved.         : int  0 6 0 5 6 5 8 0 6 2 ...
 $ Curricular.units.2nd.sem..grade.            : num  0 13.7 0 12.4 13 ...
 $ Curricular.units.2nd.sem..without.evaluations.: int  0 0 0 0 0 5 0 0 0 0 ...
 $ Unemployment.rate                           : num  10.8 13.9 10.8 9.4 13.9 1
6.2 15.5 15.5 16.2 8.9 ...
 $ Inflation.rate                              : num  1.4 -0.3 1.4 -0.8 -0.3 0.
3 2.8 2.8 0.3 1.4 ...
 $ GDP                                         : num  1.74 0.79 1.74 -3.12 0.79
-0.92 -4.06 -4.06 -0.92 3.51 ...
 $ Target                                      : chr  "Dropout" "Graduate" "Dro
pout" "Graduate" ...
```

```
# Summarize the dataset with statistics for each column
summary(ds)
```



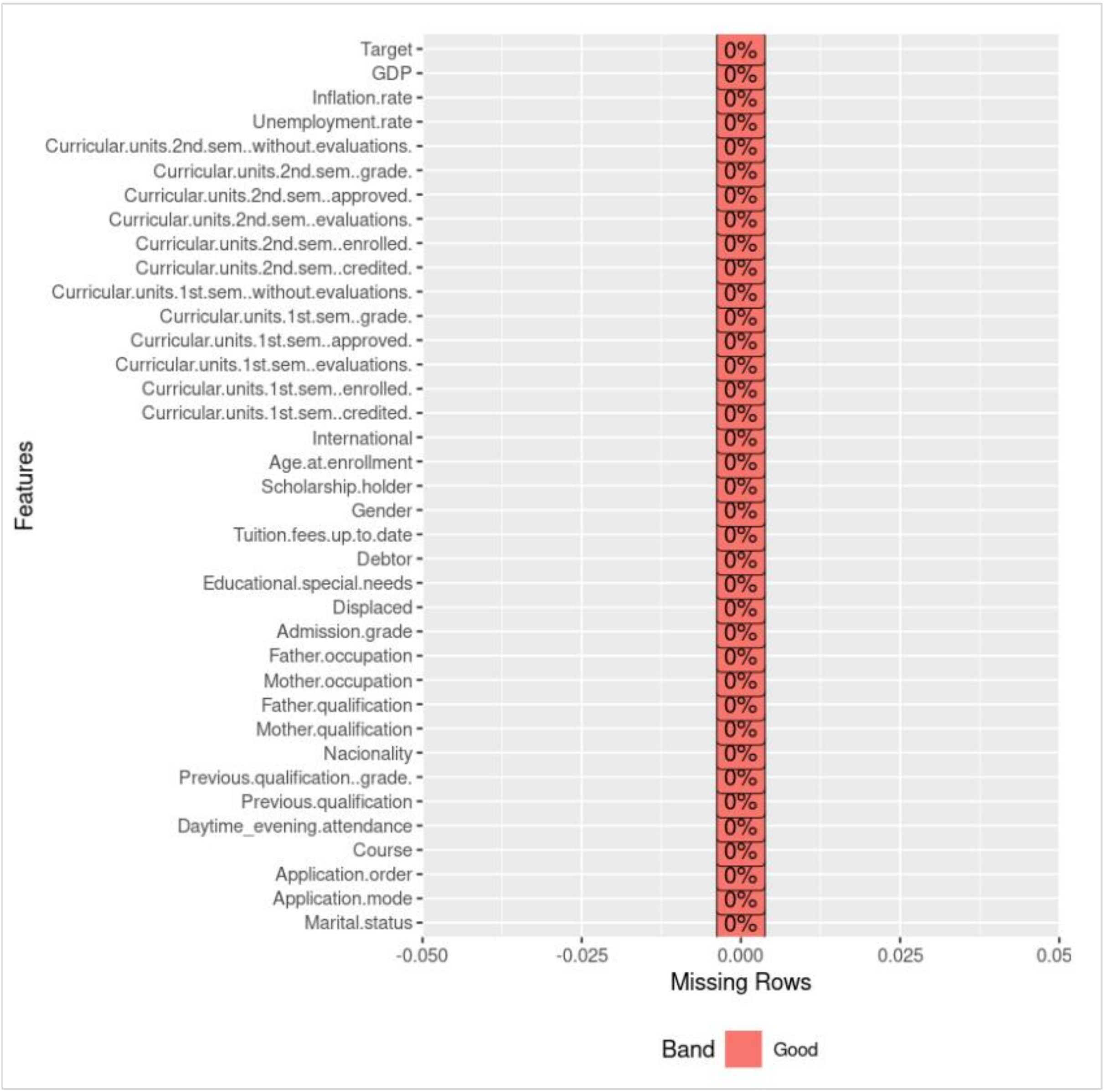
```
Marital.status  Application.mode Application.order     Course
Min.   :1.000   Min.   : 1.00    Min.   :0.000      Min.   :  33
1st Qu.:1.000   1st Qu.: 1.00    1st Qu.:1.000      1st Qu.:9085
Median :1.000   Median :17.00    Median :1.000      Median :9238
Mean   :1.179   Mean   :18.67    Mean   :1.728      Mean   :8857
3rd Qu.:1.000   3rd Qu.:39.00    3rd Qu.:2.000      3rd Qu.:9556
Max.   :6.000   Max.   :57.00    Max.   :9.000      Max.   :9991
Daytime_evening.attendance Previous.qualification
Min.   :0.0000             Min.   : 1.000
1st Qu.:1.0000             1st Qu.: 1.000
Median :1.0000             Median : 1.000
Mean   :0.8908             Mean   : 4.578
3rd Qu.:1.0000             3rd Qu.: 1.000
Max.   :1.0000             Max.   :43.000
Previous.qualification..grade.  Nacionality      Mother.qualification
Min.   : 95.0                  Min.   :  1.000   Min.   : 1.00
1st Qu.:125.0                  1st Qu.:  1.000   1st Qu.: 2.00
Median :133.1                  Median :  1.000   Median :19.00
Mean   :132.6                  Mean   :  1.873   Mean   :19.56
3rd Qu.:140.0                  3rd Qu.:  1.000   3rd Qu.:37.00
Max.   :190.0                  Max.   :109.000   Max.   :44.00
Father.qualification Mother.occupation Father.occupation Admission.grade
Min.   : 1.00        Min.   :  0.00    Min.   :  0.00    Min.   : 95.0
1st Qu.: 3.00        1st Qu.:  4.00    1st Qu.:  4.00    1st Qu.:117.9
Median :19.00        Median :  5.00    Median :  7.00    Median :126.1
Mean   :22.28        Mean   : 10.96    Mean   : 11.03    Mean   :127.0
3rd Qu.:37.00        3rd Qu.:  9.00    3rd Qu.:  9.00    3rd Qu.:134.8
Max.   :44.00        Max.   :194.00    Max.   :195.00    Max.   :190.0
  Displaced      Educational.special.needs      Debtor
Min.   :0.0000   Min.   :0.00000           Min.   :0.0000
1st Qu.:0.0000   1st Qu.:0.00000           1st Qu.:0.0000
Median :1.0000   Median :0.00000           Median :0.0000
Mean   :0.5484   Mean   :0.01153           Mean   :0.1137
3rd Qu.:1.0000   3rd Qu.:0.00000           3rd Qu.:0.0000
Max.   :1.0000   Max.   :1.00000           Max.   :1.0000
Tuition.fees.up.to.date      Gender       Scholarship.holder Age.at.enrollment
Min.   :0.0000          Min.   :0.0000   Min.   :0.0000     Min.   :17.00
1st Qu.:1.0000          1st Qu.:0.0000   1st Qu.:0.0000     1st Qu.:19.00
Median :1.0000          Median :0.0000   Median :0.0000     Median :20.00
Mean   :0.8807          Mean   :0.3517   Mean   :0.2484     Mean   :23.27
3rd Qu.:1.0000          3rd Qu.:1.0000   3rd Qu.:0.0000     3rd Qu.:25.00
Max.   :1.0000          Max.   :1.0000   Max.   :1.0000     Max.   :70.00
International     Curricular.units.1st.sem..credited.
Min.   :0.00000   Min.   : 0.00
1st Qu.:0.00000   1st Qu.: 0.00
Median :0.00000   Median : 0.00
Mean   :0.02486   Mean   : 0.71
3rd Qu.:0.00000   3rd Qu.: 0.00
Max.   :1.00000   Max.   :20.00
Curricular.units.1st.sem..enrolled. Curricular.units.1st.sem..evaluations.
Min.   : 0.000                      Min.   : 0.000
1st Qu.: 5.000                      1st Qu.: 6.000
Median : 6.000                      Median : 8.000
Mean   : 6.271                      Mean   : 8.299
3rd Qu.: 7.000                      3rd Qu.:10.000
Max.   :26.000                      Max.   :45.000
Curricular.units.1st.sem..approved. Curricular.units.1st.sem..grade.
Min.   : 0.000                      Min.   : 0.00
1st Qu.: 3.000                      1st Qu.:11.00
Median : 5.000                      Median :12.29
```

```
Mean   : 4.707                      Mean   :10.64
3rd Qu.: 6.000                      3rd Qu.:13.40
Max.   :26.000                      Max.   :18.88
Curricular.units.1st.sem..without.evaluations.
Min.   : 0.0000
1st Qu.: 0.0000
Median : 0.0000
Mean   : 0.1377
3rd Qu.: 0.0000
Max.   :12.0000
Curricular.units.2nd.sem..credited. Curricular.units.2nd.sem..enrolled.
Min.   : 0.0000                     Min.   : 0.000
1st Qu.: 0.0000                     1st Qu.: 5.000
Median : 0.0000                     Median : 6.000
Mean   : 0.5418                     Mean   : 6.232
3rd Qu.: 0.0000                     3rd Qu.: 7.000
Max.   :19.0000                     Max.   :23.000
Curricular.units.2nd.sem..evaluations. Curricular.units.2nd.sem..approved.
Min.   : 0.000                         Min.   : 0.000
1st Qu.: 6.000                         1st Qu.: 2.000
Median : 8.000                         Median : 5.000
Mean   : 8.063                         Mean   : 4.436
3rd Qu.:10.000                         3rd Qu.: 6.000
Max.   :33.000                         Max.   :20.000
Curricular.units.2nd.sem..grade.
Min.   : 0.00
1st Qu.:10.75
Median :12.20
Mean   :10.23
3rd Qu.:13.33
Max.   :18.57
Curricular.units.2nd.sem..without.evaluations. Unemployment.rate
Min.   : 0.0000                                Min.   : 7.60
1st Qu.: 0.0000                                1st Qu.: 9.40
Median : 0.0000                                Median :11.10
Mean   : 0.1503                                Mean   :11.57
3rd Qu.: 0.0000                                3rd Qu.:13.90
Max.   :12.0000                                Max.   :16.20
Inflation.rate        GDP                Target
Min.   :-0.800   Min.   :-4.060000   Length:4424
1st Qu.: 0.300   1st Qu.:-1.700000   Class :character
Median : 1.400   Median : 0.320000   Mode  :character
Mean   : 1.228   Mean   : 0.001969
3rd Qu.: 2.600   3rd Qu.: 1.790000
Max.   : 3.700   Max.   : 3.510000
```

```
# Check for the dimensions of the dataset, namely the count of rows and columns
dim(ds)
```

```
4424 · 37
```

```
# Graphically represent the presence of missing entries across all columns
plot_missing(ds)
```



```
# Determine the count of distinct entries in each column
sapply(ds, function(x) length(unique(x)))
```

```
Marital.status:         6 Application.mode:        18 Application.order:        8 Course:        17 Daytime_evening.attendance:        2 Previous.qualification:        17
 Previous.qualification..grade.:        101 Nacionality:        21 Mother.qualification:        29 Father.qualification:        34 Mother.occupation:        32
 Father.occupation:        46 Admission.grade:        620 Displaced:        2 Educational.special.needs:        2 Debtor:        2 Tuition.fees.up.to.date:        2
 Gender:        2 Scholarship.holder:        2 Age.at.enrollment:        46 International:        2 Curricular.units.1st.sem..credited.:        21
 Curricular.units.1st.sem..enrolled.:        23 Curricular.units.1st.sem..evaluations.:        35 Curricular.units.1st.sem..approved.:        23
 Curricular.units.1st.sem..grade.:        797 Curricular.units.1st.sem..without.evaluations.:        11 Curricular.units.2nd.sem..credited.:        19
 Curricular.units.2nd.sem..enrolled.:        22 Curricular.units.2nd.sem..evaluations.:        30 Curricular.units.2nd.sem..approved.:        20
 Curricular.units.2nd.sem..grade.:        782 Curricular.units.2nd.sem..without.evaluations.:        10 Unemployment.rate:        10 Inflation.rate:        9 GDP:        10
 Target:        3
```

```r
# Subsection 2.2: Data Cleaning

# Subsection 2.2.1: Renaming of the Column Label

# Change the column label from 'Nacionality' to the correct spelling 'Nationality'
ds <- ds %>%
  rename(Nationality = Nacionality)
```

```r
# Subsection 2.2.2: Conversion of Variables Into Factor Data

# Specify columns to be converted into factor data type
categorical_columns <- c(
  "Marital.status", "Application.mode", "Application.order", "Course", "Daytime_evening.attendance",
  "Previous.qualification", "Nationality", "Mother.qualification", "Father.qualification",
  "Mother.occupation", "Father.occupation", "Displaced", "Educational.special.needs", "Debtor",
  "Tuition.fees.up.to.date", "Gender", "Scholarship.holder", "Age.at.enrollment", "International",
  "Curricular.units.1st.sem..credited.", "Curricular.units.1st.sem..enrolled.",
  "Curricular.units.1st.sem..evaluations.", "Curricular.units.1st.sem..approved.",
  "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..without.evaluations.", "Target"
)
```

```r
cleaned_ds[columns] <- lapply(cleaned_ds[columns], factor)
```

```r
# Subsection 2.2.3: Detection and Treatment of Outliers

# Identify continuous variables to examine for potential outliers
continuous_variables <- names(ds)[sapply(ds, is.numeric)]
print(continuous_variables)
```

```
[1] "Previous.qualification..grade."   "Admission.grade"
[3] "Curricular.units.1st.sem..grade." "Curricular.units.2nd.sem..grade."
[5] "Unemployment.rate"                "Inflation.rate"
[7] "GDP"
```

```r
# Create a function to pinpoint outliers based on the interquartile range method
outliers_function <- function(data, column) {
  Q1 <- quantile(data[[column]], 0.25)
  Q3 <- quantile(data[[column]], 0.75)
  IQR <- Q3 - Q1
  lower_bound <- Q1 - 3 * IQR
  upper_bound <- Q3 + 3 * IQR
  outliers <- data[data[[column]] < lower_bound | data[[column]] > upper_bound, ]
  return(nrow(outliers))
}

# Count and report the number of outliers in each continuous column
outliers_count <- sapply(continuous_variables, outliers_function, data = ds)
print(outliers_count)
```

```
  Previous.qualification..grade.                  Admission.grade 
                               3                                3 
Curricular.units.1st.sem..grade. Curricular.units.2nd.sem..grade. 
                             718                              870 
               Unemployment.rate                   Inflation.rate 
                               0                                0 
                             GDP 
                               0 
```

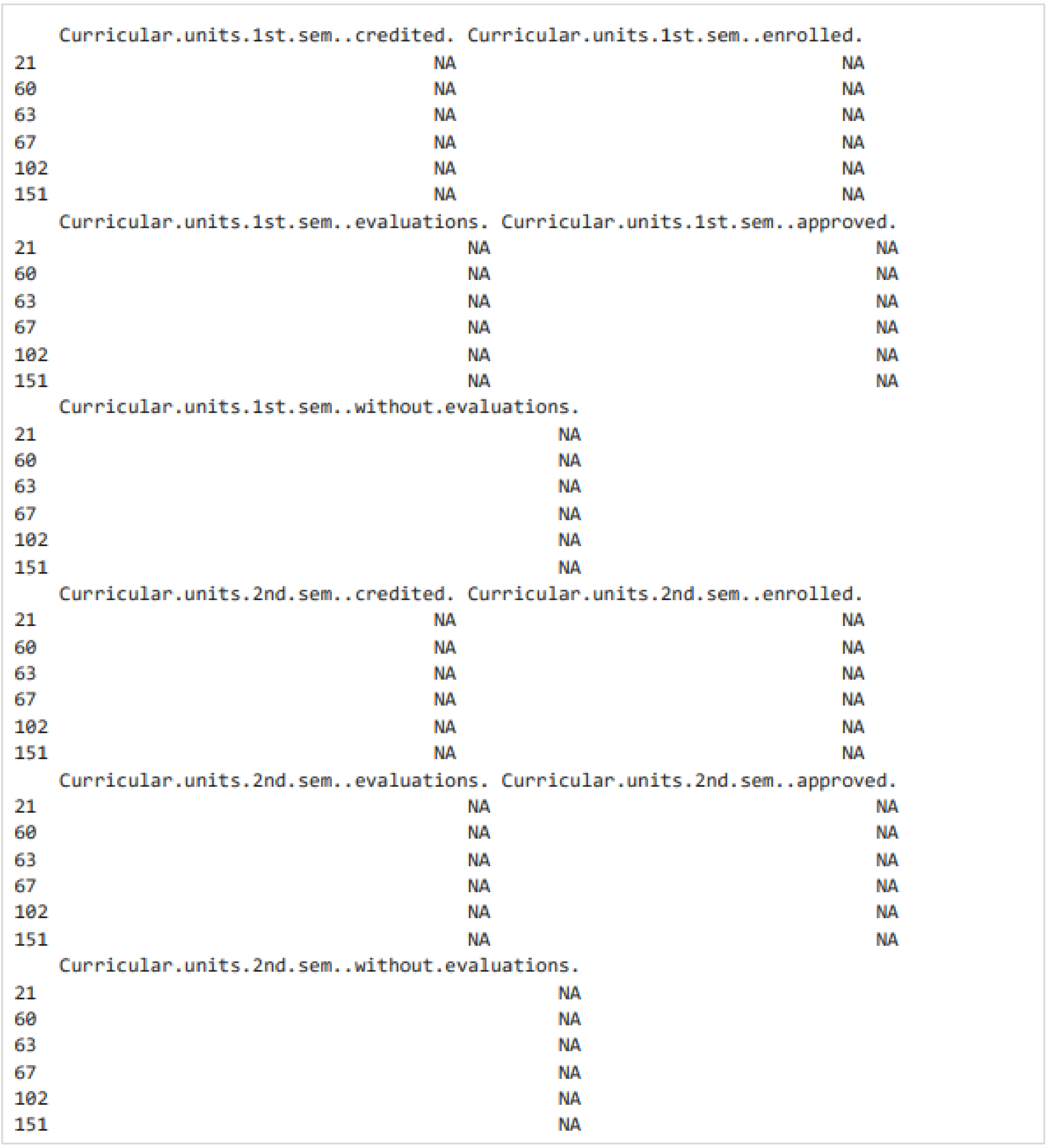
```
# Implement a procedure to adjust outlier values to the column mean, thereby mitigating their effect
columns_to_treat <- c('Previous.qualification..grade.', 'Admission.grade')
mean_treatment_function <- function(data, column) {
  Q1 <- quantile(data[[column]], 0.25)
  Q3 <- quantile(data[[column]], 0.75)
  IQR <- Q3 - Q1
  lower_bound <- Q1 - 3 * IQR
  upper_bound <- Q3 + 3 * IQR

  is_outlier <- data[[column]] < lower_bound | data[[column]] > upper_bound
  mean_value <- mean(data[[column]], na.rm = TRUE)
  data[is_outlier, column] <- mean_value

  return(data)
}
```

```
# Apply the outlier adjustment function to the relevant columns.
for (column in columns_to_treat) {
  ds <- mean_treatment_function(ds, column)
}

# Reevaluate the columns to confirm the adjustment of outliers
post_outliers <- sapply(columns_to_treat, outliers_function, data = ds)
print(post_outliers)
```

```
Previous.qualification..grade.                 Admission.grade
                             0                               0
```

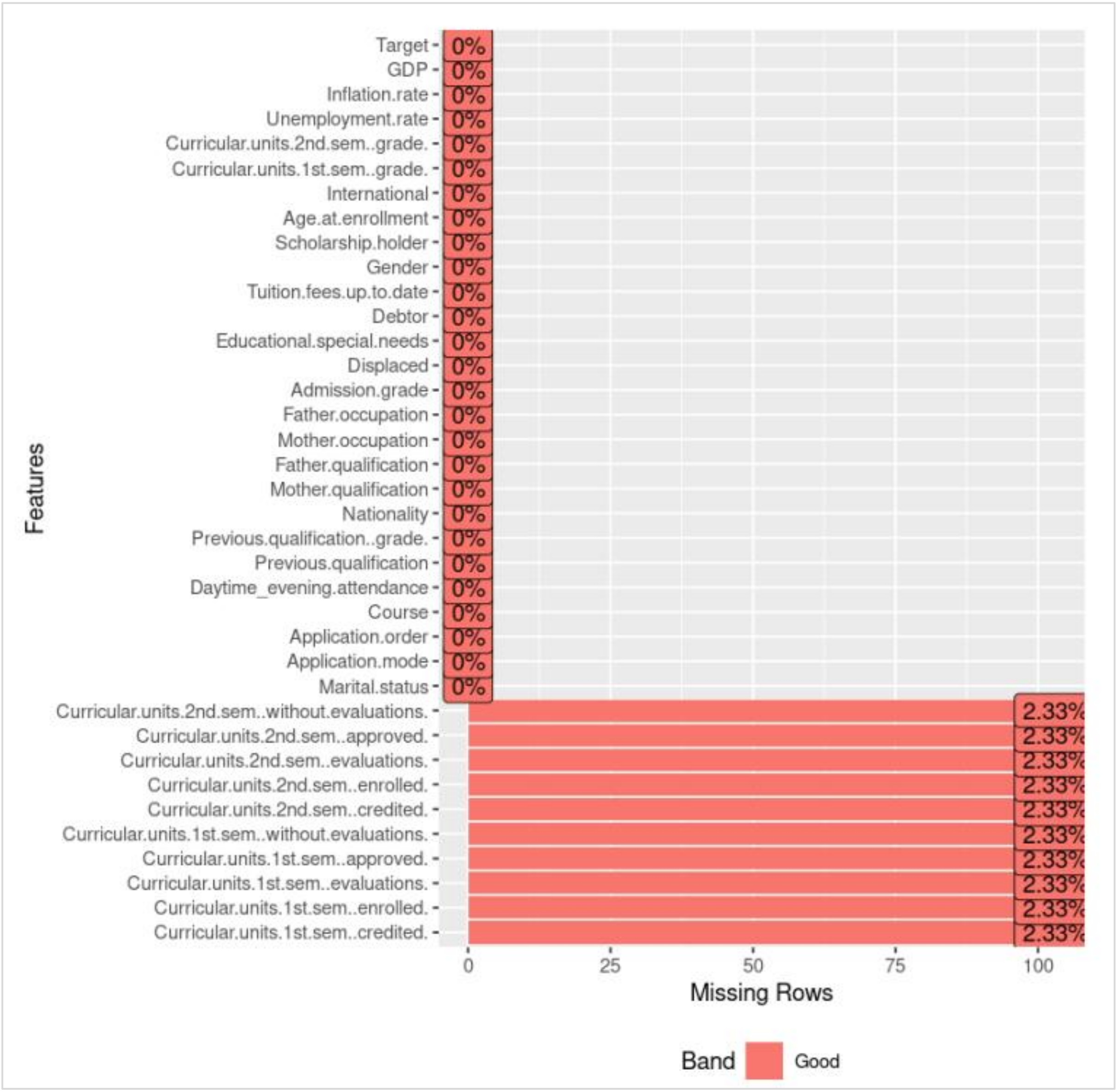
```
# Subsection 2.2.5: Treatment of Problematic Entries

# Apply filters to identify and label specific academic data as missing (NA).
problematic_columns <- c(
  "Curricular.units.1st.sem..credited.", "Curricular.units.1st.sem..enrolled.",
  "Curricular.units.1st.sem..evaluations.", "Curricular.units.1st.sem..approved.",
  "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..without.evaluations."
)
```

```
# Process the dataset by marking academic records as missing (NA) when all specified academic
# columns have zero values, specifically for rows where the target status is 'Graduate' or 'Enrolled'.
# Then, remove the temporary column used for calculations.
ds <- ds %>%
  mutate(academic_sum = rowSums(select(., all_of(problematic_columns)) == 0)) %>%
  mutate_at(vars(all_of(problematic_columns)), ~ifelse(Target %in% c('Graduate', 'Enrolled')
                                                     & academic_sum == length(problematic_columns), NA, .)) %>%
  select(-academic_sum)  # Remove the temporary helper column

# Preview the modified dataset where certain records are now marked as missing.
filtered_indexes <- which(ds$Target %in% c('Graduate', 'Enrolled') &
                            rowSums(ds[problematic_columns], na.rm = TRUE) == 0)
print(head(ds[filtered_indexes, problematic_columns]))
```



```
    Curricular.units.1st.sem..credited. Curricular.units.1st.sem..enrolled.
21                                   NA                                  NA
60                                   NA                                  NA
63                                   NA                                  NA
67                                   NA                                  NA
102                                  NA                                  NA
151                                  NA                                  NA
    Curricular.units.1st.sem..evaluations. Curricular.units.1st.sem..approved.
21                                      NA                                  NA
60                                      NA                                  NA
63                                      NA                                  NA
67                                      NA                                  NA
102                                     NA                                  NA
151                                     NA                                  NA
    Curricular.units.1st.sem..without.evaluations.
21                                              NA
60                                              NA
63                                              NA
67                                              NA
102                                             NA
151                                             NA
    Curricular.units.2nd.sem..credited. Curricular.units.2nd.sem..enrolled.
21                                   NA                                  NA
60                                   NA                                  NA
63                                   NA                                  NA
67                                   NA                                  NA
102                                  NA                                  NA
151                                  NA                                  NA
    Curricular.units.2nd.sem..evaluations. Curricular.units.2nd.sem..approved.
21                                      NA                                  NA
60                                      NA                                  NA
63                                      NA                                  NA
67                                      NA                                  NA
102                                     NA                                  NA
151                                     NA                                  NA
    Curricular.units.2nd.sem..without.evaluations.
21                                              NA
60                                              NA
63                                              NA
67                                              NA
102                                             NA
151                                             NA
```

```
# Visualize missing data across all columns of the dataset.
plot_missing(ds)
```



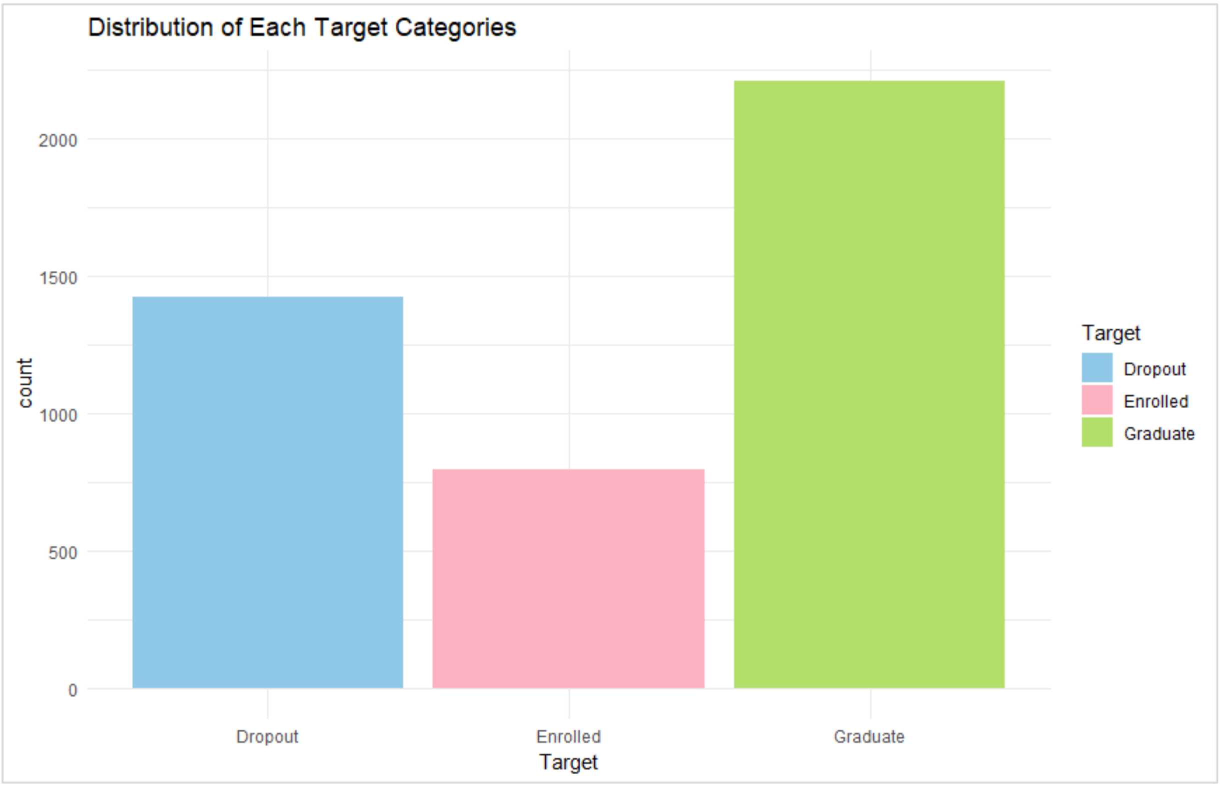
```
# Implement missForest for imputing the missing values identified
seed <- 123
set.seed(seed)
ds_cleaned <- missForest(ds, maxiter = 10, ntree = 100, variablewise = TRUE, decreasing = TRUE)
ds_cleaned <- missForest(ds)$ximp
```

```
# Save the dataset with imputed values as a new file
write.csv(ds_cleaned, file = "cleaned_data.csv", row.names = FALSE)

# Reload the dataset with the imputed values
cleaned_ds <- read.csv("cleaned_data.csv")
```

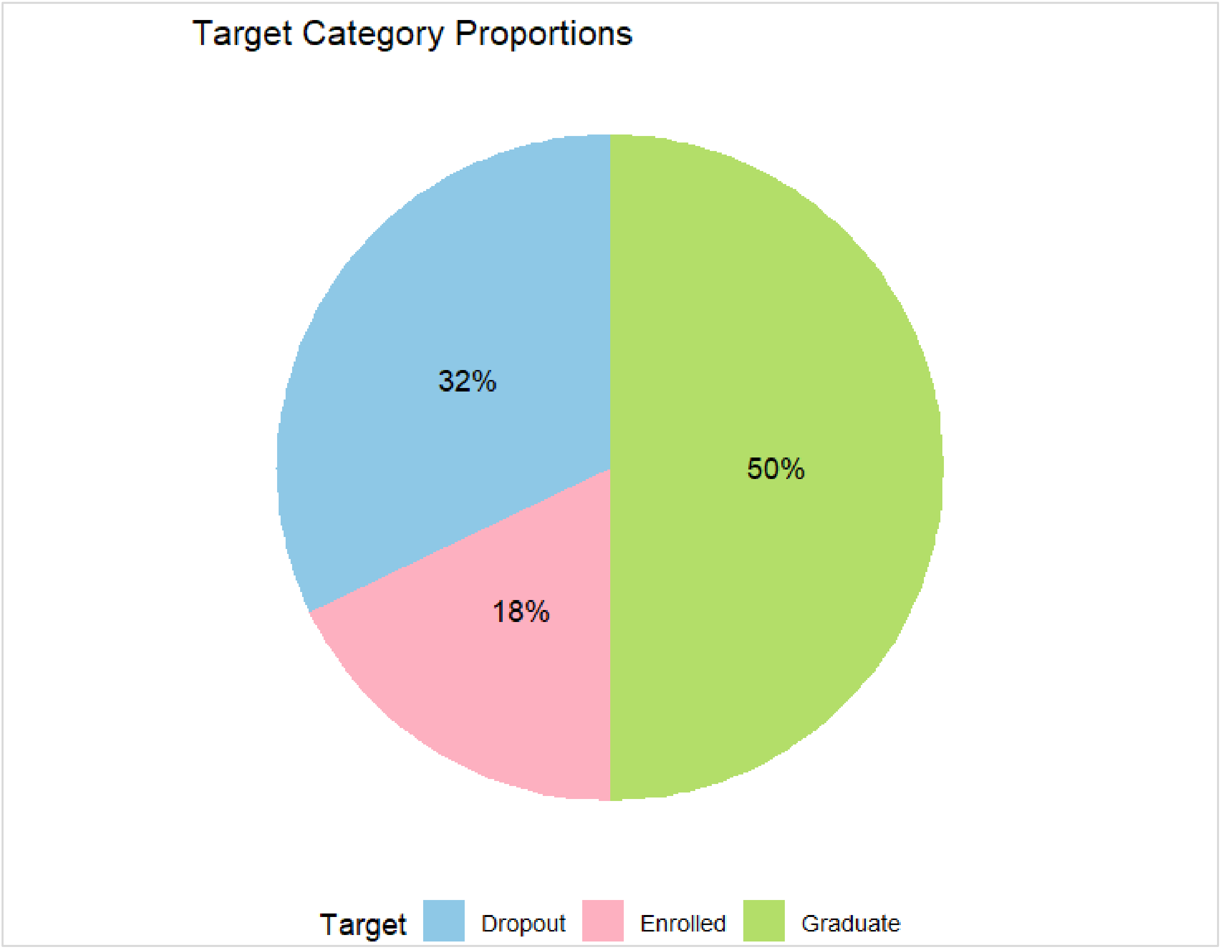
```
# Section 3.0: Exploratory Data Analysis (EDA)

# Subsection 3.1: Examining the Target Column

# Count and display the frequency of each target category
table(cleaned_ds$Target)
```

```
Dropout Enrolled Graduate
   1421      794     2209
```

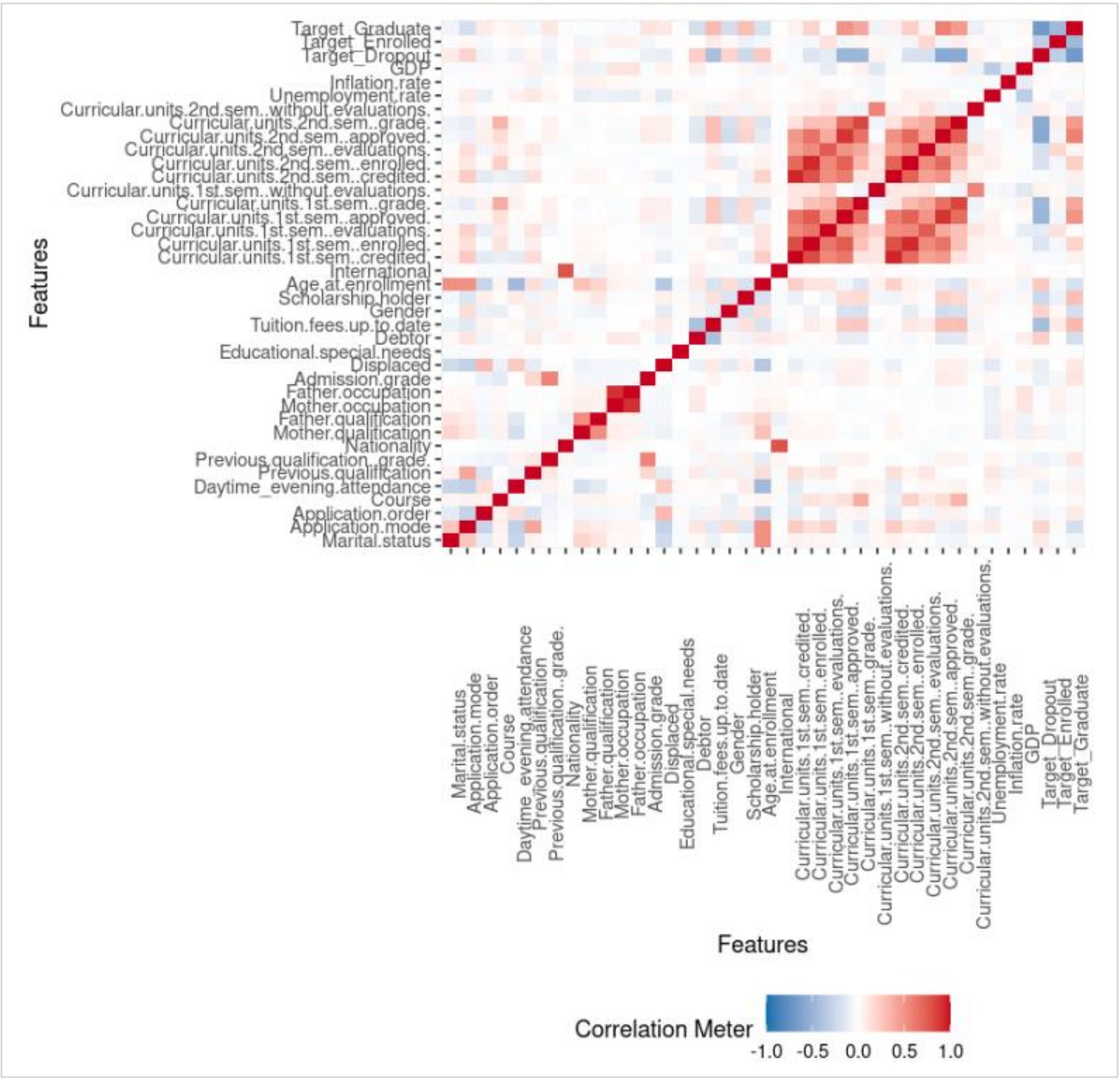
```
# Visual representation of target category distribution using a bar chart with eye-friendly colors
ggplot(cleaned_ds, aes(x = Target, fill = Target)) +
  geom_bar(stat = "count") +
  labs(title = "Distribution of Each Target Categories") +
  scale_fill_manual(values = c("#8EC8E6", "#FDB0C0", "#B3DE69")) + # Soft blue, soft pink, soft green
  theme_minimal()
```



```
# Represent target category proportions using a pie chart.
ggplot(cleaned_ds, aes(x = "", fill = Target)) +
  geom_bar(width = 1, stat = "count") +
  coord_polar("y", start = 0) +
  labs(title = "Target Category Proportions") +
  scale_fill_manual(values = c("#8EC8E6", "#FDB0C0", "#B3DE69")) +
  geom_text(aes(label = scales::percent(..count.. / sum(..count..))),
            stat = "count", position = position_stack(vjust = 0.5)) +
  theme_void() +
  theme(legend.position = "bottom")
```



```
# Subsection 3.2: Exploring Correlations in Data

# Generate a matrix to visualize the correlations between different variables in the dataset
plot_correlation(cleaned_ds)
```



```
# Section 4.0: Implementing the Model

# Subsection 4.1: Preparing the Data for Modelling

# Subsection 4.1.1: Preparing the Target Variable

# Show distinct categories in the target column
unique(cleaned_ds$Target)
```

```
'Dropout' · 'Graduate' · 'Enrolled'
```

```r
# Recode target column values to numerical labels: Graduate as 0, Enrolled as 1, Dropout as 2
cleaned_ds$Target <- factor(cleaned_ds$Target, levels = c("Graduate", "Enrolled", "Dropout"),
                            labels = c(0, 1, 2))

# Convert categorical columns to factors for modeling
columns <- c(
  "Marital.status", "Application.mode", "Application.order", "Course", "Daytime_evening.attendance",
  "Previous.qualification", "Nationality", "Mother.qualification", "Father.qualification",
  "Mother.occupation", "Father.occupation", "Displaced", "Educational.special.needs", "Debtor",
  "Tuition.fees.up.to.date", "Gender", "Scholarship.holder", "Age.at.enrollment", "International",
  "Curricular.units.1st.sem..credited.", "Curricular.units.1st.sem..enrolled.",
  "Curricular.units.1st.sem..evaluations.", "Curricular.units.1st.sem..approved.",
  "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..without.evaluations.", "Target"
)

cleaned_ds[columns] <- lapply(cleaned_ds[columns], factor)

# Check the recoded target column values
unique(cleaned_ds$Target)
```

```
[1] 2 0 1
Levels: 0 1 2
```

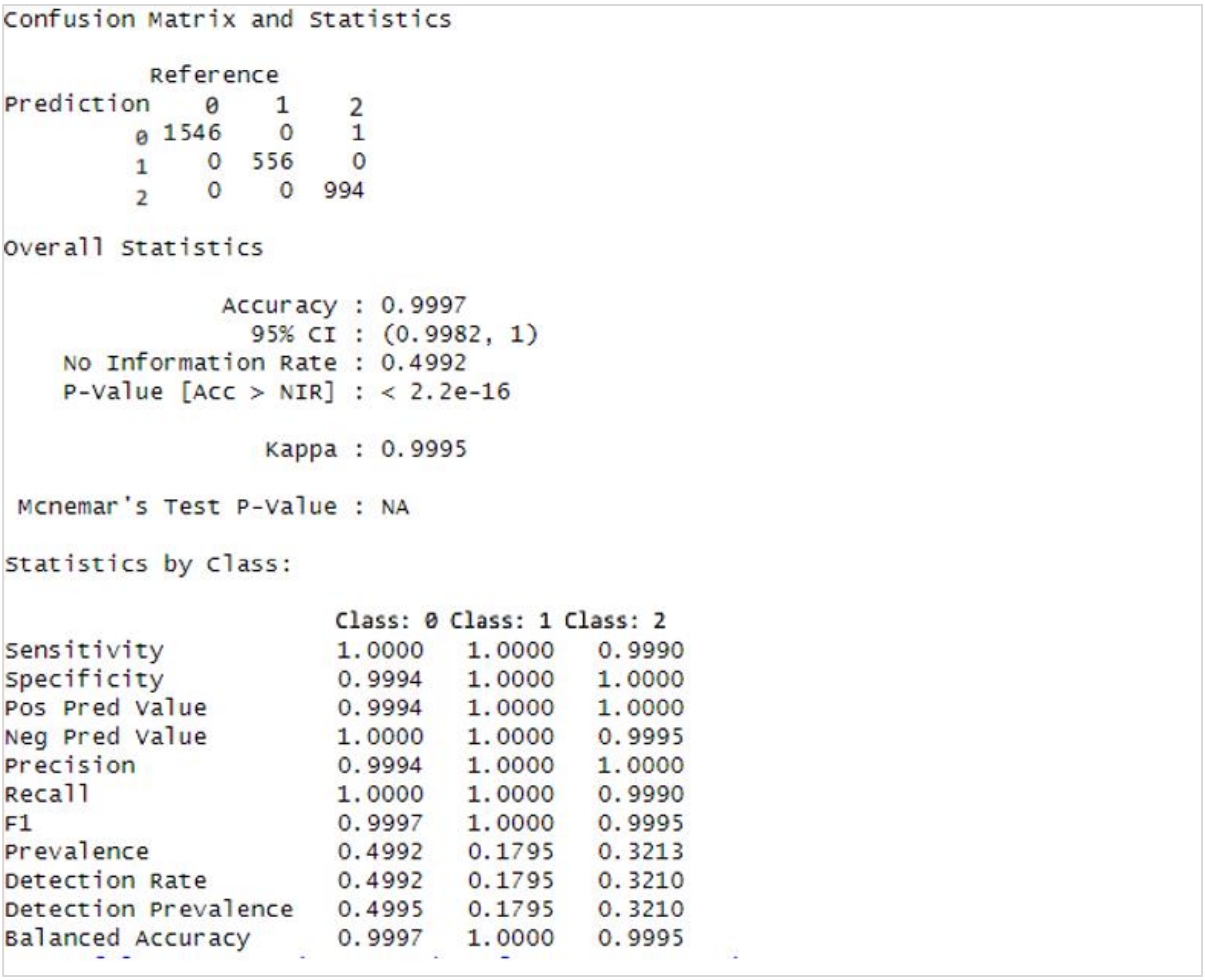
```r
# Subsection 4.1.2: Standardizing Numerical Data

# Identify and list columns with numeric data
numeric_columns <- names(cleaned_ds)[sapply(cleaned_ds, is.numeric)]

# Standardize these columns for uniform scaling
cleaned_ds[numeric_columns] <- lapply(cleaned_ds[numeric_columns], scale)

# Overview of standardized data
summary(cleaned_ds[numeric_columns])
```

```
Previous.qualification..grade..V1 Admission.grade.V1
Min.   :-2.867157                 Min.   :-2.219488
1st Qu.:-0.578001                 1st Qu.:-0.627953
Median : 0.040072                 Median :-0.058059
Mean   : 0.000000                 Mean   : 0.000000
3rd Qu.: 0.566578                 3rd Qu.: 0.546586
Max.   : 3.954530                 Max.   : 3.993753
Curricular.units.1st.sem..grade..V1 Curricular.units.2nd.sem..grade..V1
Min.   :-2.1968541                  Min.   :-1.9632667
1st Qu.: 0.0741543                  1st Qu.: 0.0997531
Median : 0.3395968                  Median : 0.3780209
Mean   : 0.0000000                  Mean   : 0.0000000
3rd Qu.: 0.5696470                  3rd Qu.: 0.5955176
Max.   : 1.6999898                  Max.   : 1.6007542
Unemployment.rate.V1  Inflation.rate.V1        GDP.V1
Min.   :-1.4888746   Min.   :-1.4667052   Min.   :-1.7894645
1st Qu.:-0.8131610   1st Qu.:-0.6711664   1st Qu.:-0.7497873
Median :-0.1749870   Median : 0.1243724   Median : 0.1401058
Mean   : 0.0000000   Mean   : 0.0000000   Mean   : 0.0000000
3rd Qu.: 0.8761230   3rd Qu.: 0.9922329   3rd Qu.: 0.7877013
Max.   : 1.7395349   Max.   : 1.7877717   Max.   : 1.5454321
```

```
# Subsection 4.1.3: Splitting Data into Training and Testing Sets

# Divide the dataset into training and testing sets using a 70:30 ratio
set.seed(321)
split = sample.split(cleaned_ds$Target, SplitRatio = 0.7)
training_data = subset(cleaned_ds, split == TRUE)
testing_data = subset(cleaned_ds, split == FALSE)
```

```
# Subsection 4.2: Building an XGBoost Model

# Subsection 4.2.1: Initial Model Training

# Setting default parameters for XGBoost
params <- list(
  booster = "gbtree",
  num_class = 3,
  objective = "multi:softprob",
  verbosity = 1
)

# Converting categorical variables in training data to numeric
training_data[] <- lapply(training_data, function(x) if (is.factor(x)) as.numeric(x) else x)

# Identifying the position of the target variable in the dataset
target_column_index <- which(colnames(training_data) == "Target")

# Preparing feature matrix and target vector for model input
features_matrix <- as.matrix(training_data[, -target_column_index, drop = FALSE])
labels_vector <- as.vector(training_data[, target_column_index]) - 1

# Creating DMatrix for XGBoost
xgb_data <- xgb.DMatrix(data = features_matrix, label = labels_vector)
```

```
# Executing the XGBoost training
set.seed(321)
xgb_base_model <- xgb.train(params = params, data = xgb_data, nrounds = 100)

# Assessing the Base Model on Training Data

# Predicting probabilities on training data
xgb_prob_predictions <- predict(xgb_base_model, as.matrix(training_data[, -target_column_index]))

# Transforming probabilities to predicted classes
xgb_class_predictions <- max.col(matrix(xgb_prob_predictions, nrow =
                                 length(training_data$Target), byrow = TRUE))

# Preparing actual and predicted values for confusion matrix
actuals <- as.factor(training_data$Target)
predictions <- as.factor(xgb_class_predictions)

# Generating and displaying training data confusion matrix
confusionMatrix(predictions, actuals, mode = "everything")
```



```
Confusion Matrix and Statistics

          Reference
Prediction    0    1    2
         0 1546    0    1
         1    0  556    0
         2    0    0  994

Overall Statistics

               Accuracy : 0.9997
                 95% CI : (0.9982, 1)
    No Information Rate : 0.4992
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9995

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            1.0000   1.0000   0.9990
Specificity            0.9994   1.0000   1.0000
Pos Pred Value         0.9994   1.0000   1.0000
Neg Pred Value         1.0000   1.0000   0.9995
Precision              0.9994   1.0000   1.0000
Recall                 1.0000   1.0000   0.9990
F1                     0.9997   1.0000   0.9995
Prevalence             0.4992   0.1795   0.3213
Detection Rate         0.4992   0.1795   0.3210
Detection Prevalence   0.4995   0.1795   0.3210
Balanced Accuracy      0.9997   1.0000   0.9995
```

```
# Model Evaluation on Test Data

# Converting test data factors to numeric
testing_data[] <- lapply(testing_data, function(x) if (is.factor(x)) as.numeric(x) else x)

# Making predictions on the test dataset
xgb_prob_predictions_test <- predict(xgb_base_model, as.matrix(testing_data[, -target_column_index]))

# Determining class labels from predicted probabilities
xgb_class_predictions_test <- max.col(matrix(xgb_prob_predictions_test, nrow =
                                         length(testing_data$Target), byrow = TRUE)) - 1
xgb_class_predictions_test_adjusted <- xgb_class_predictions_test + 1

# Preparing actual and predicted values for test data confusion matrix
actuals_test <- as.factor(testing_data$Target)
predictions_test <- as.factor(xgb_class_predictions_test_adjusted)

# Generating and displaying test data confusion matrix
confusionMatrix(predictions_test, actuals_test, mode = "everything")
```

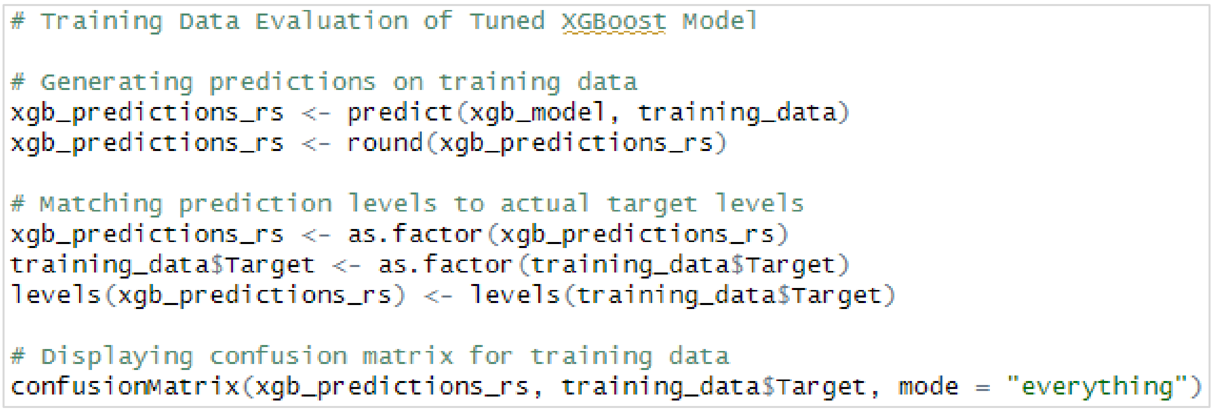
```
Confusion Matrix and Statistics

          Reference
Prediction   0   1   2
         0 625  90  52
         1  22 104  40
         2  16  44 334

Overall Statistics

               Accuracy : 0.8011
                 95% CI : (0.7785, 0.8222)
    No Information Rate : 0.4996
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6648

 Mcnemar's Test P-Value : 4.518e-13

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            0.9427  0.43697   0.7840
Specificity            0.7861  0.94307   0.9334
Pos Pred Value         0.8149  0.62651   0.8477
Neg Pred Value         0.9321  0.88458   0.9014
Precision              0.8149  0.62651   0.8477
Recall                 0.9427  0.43697   0.7840
F1                     0.8741  0.51485   0.8146
Prevalence             0.4996  0.17935   0.3210
Detection Rate         0.4710  0.07837   0.2517
Detection Prevalence   0.5780  0.12509   0.2969
Balanced Accuracy      0.8644  0.69002   0.8587
```

```
# Evaluating Model Using AUC Metric

# Organizing predicted probabilities into a matrix for AUC calculation
xgb_prob_matrix <- matrix(xgb_prob_predictions_test, nrow = length(testing_data$Target), byrow = TRUE)

# Labeling columns of the probability matrix for AUC calculation
colnames(xgb_prob_matrix) <- levels(as.factor(testing_data$Target))

# Calculating and displaying multi-class AUC
auc_result <- multiclass.roc(testing_data$Target, xgb_prob_matrix)
print(paste("Multi-class AUC:", auc_result$auc))
```

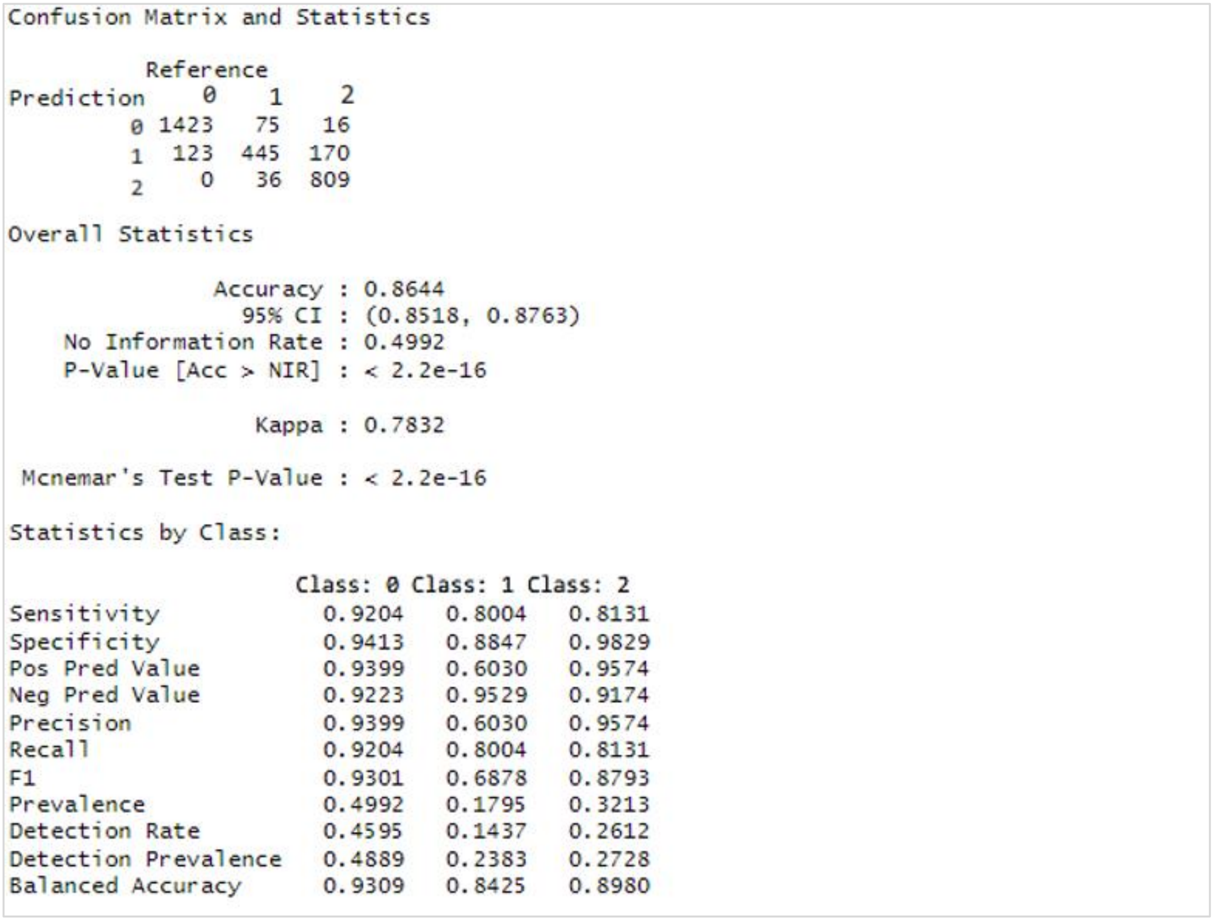
```
[1] "Multi-class AUC: 0.894430570151853"
```

```
# Subsection 4.2.2: Hyperparameter Tuning

# Defining cross-validation and search strategy
train_control <- trainControl(method = "cv", number = 5, search = "random")

# Specifying the range of hyperparameters to explore
tune_grid <- expand.grid(
  nrounds = c(100, 200),
  max_depth = c(3, 5, 7, 9),
  eta = c(0.01, 0.05, 0.1, 0.2),
  gamma = c(0, 0.1, 0.2),
  colsample_bytree = c(0.6, 0.8, 1),
  min_child_weight = c(1, 2, 3, 4),
  subsample = c(0.7, 0.8, 0.9)
)

# Conducting hyperparameter tuning on the XGBoost model
xgb_model <- train(
  Target ~ .,
  data = training_data,
  method = "xgbTree",
  trControl = train_control,
  tuneGrid = tune_grid,
  nthread = 16
)

# Outputting the best hyperparameters found
print(xgb_model$bestTune)
```

```
     nrounds max_depth  eta gamma colsample_bytree min_child_weight subsample
1228     200         5 0.05   0.2              0.6                1       0.8
```



```
# Training Data Evaluation of Tuned XGBoost Model

# Generating predictions on training data
xgb_predictions_rs <- predict(xgb_model, training_data)
xgb_predictions_rs <- round(xgb_predictions_rs)

# Matching prediction levels to actual target levels
xgb_predictions_rs <- as.factor(xgb_predictions_rs)
training_data$Target <- as.factor(training_data$Target)
levels(xgb_predictions_rs) <- levels(training_data$Target)

# Displaying confusion matrix for training data
confusionMatrix(xgb_predictions_rs, training_data$Target, mode = "everything")
```



```
Confusion Matrix and Statistics

          Reference
Prediction    0    1    2
         0 1423   75   16
         1  123  445  170
         2    0   36  809

Overall Statistics

               Accuracy : 0.8644
                 95% CI : (0.8518, 0.8763)
    No Information Rate : 0.4992
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7832

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            0.9204   0.8004   0.8131
Specificity            0.9413   0.8847   0.9829
Pos Pred Value         0.9399   0.6030   0.9574
Neg Pred Value         0.9223   0.9529   0.9174
Precision              0.9399   0.6030   0.9574
Recall                 0.9204   0.8004   0.8131
F1                     0.9301   0.6878   0.8793
Prevalence             0.4992   0.1795   0.3213
Detection Rate         0.4595   0.1437   0.2612
Detection Prevalence   0.4889   0.2383   0.2728
Balanced Accuracy      0.9309   0.8425   0.8980
```

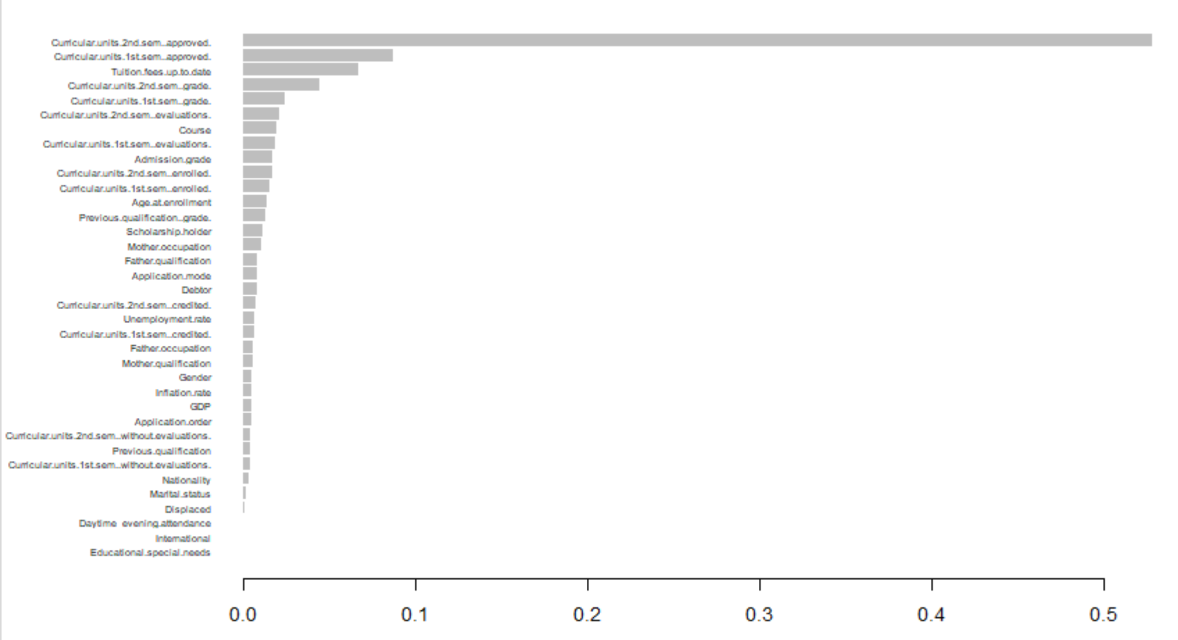
```
# Test Data Evaluation of Tuned XGBoost Model

# Generating predictions on test data
xgb_predictions_rs_test <- predict(xgb_model, testing_data)
xgb_predictions_rs_test <- round(xgb_predictions_rs_test)

# Aligning prediction levels with actual target levels for test data
xgb_predictions_rs_test <- as.factor(xgb_predictions_rs_test)
testing_data$Target <- as.factor(testing_data$Target)
levels(xgb_predictions_rs_test) <- levels(testing_data$Target)

# Displaying confusion matrix for test data
xgb_metrics <- confusionMatrix(xgb_predictions_rs_test, testing_data$Target, mode = "everything")
xgb_metrics
```

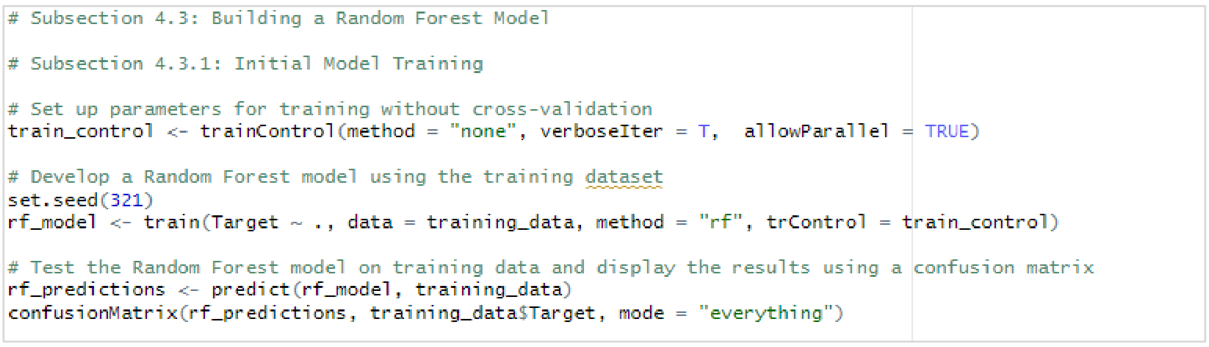
```
Confusion Matrix and Statistics

          Reference
Prediction   0   1   2
         0 557  44  33
         1 103 158  81
         2   3  36 312

Overall Statistics

               Accuracy : 0.7739
                 95% CI : (0.7505, 0.7962)
    No Information Rate : 0.4996
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6412

 Mcnemar's Test P-Value : 3.083e-14

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            0.8840   0.8310   0.9567
Specificity            0.8785   0.4620   0.8889
Pos Pred Value         0.8470   0.9188   0.8832
Neg Pred Value         0.8785   0.4620   0.8889
Precision              0.8401   0.6639   0.7324
Recall                 0.8589   0.5448   0.8031
F1                     0.4996   0.1794   0.3210
Prevalence             0.4197   0.1191   0.2351
Detection Rate         0.4778   0.2577   0.2645
Detection Prevalence   0.8621   0.7475   0.8446
Balanced Accuracy
```

```
# AUC Metric for Tuned XGBoost Model

# Predicting probabilities for test data and calculating AUC
xgb_prob_predictions_test <- predict(xgb_model, newdata = testing_data)
xgb_auc <- multiclass.roc(testing_data$Target, xgb_prob_predictions_test)
auc(xgb_auc)
```

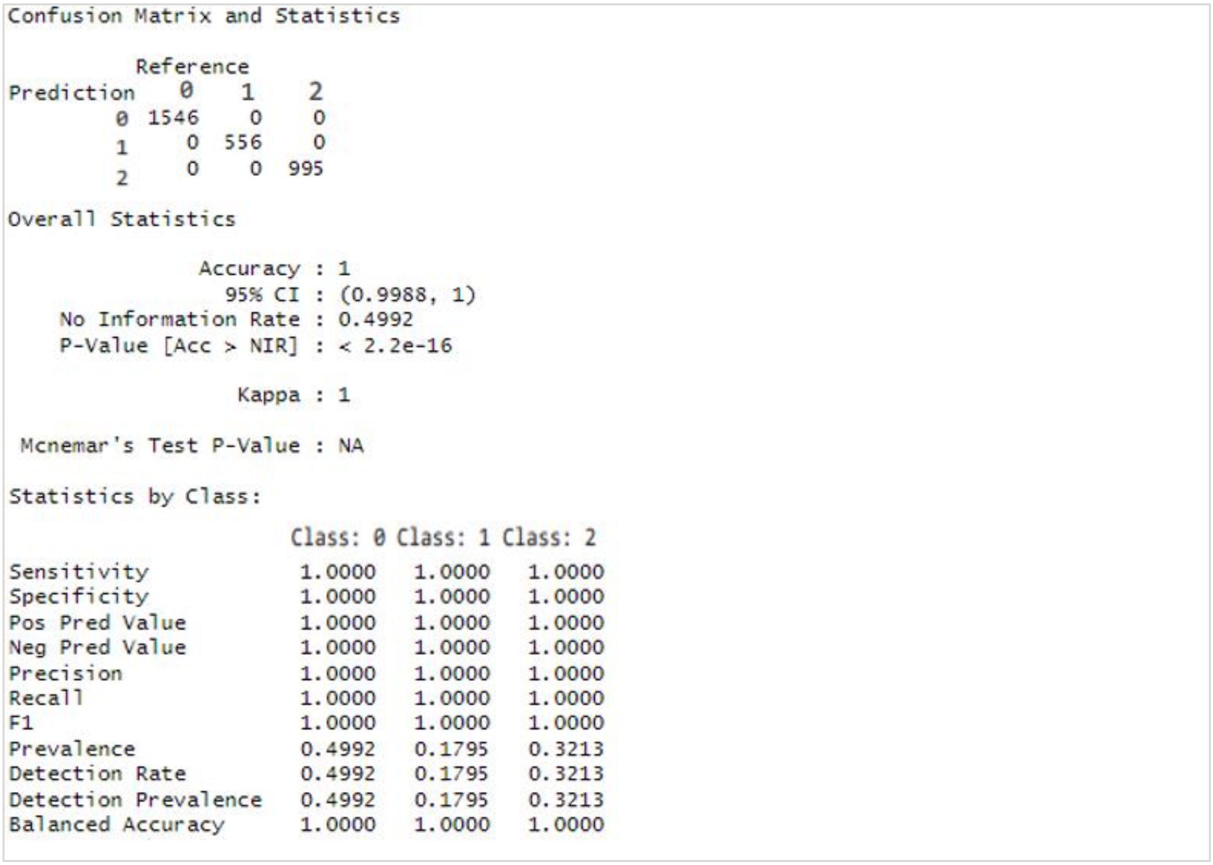
```
Multi-class area under the curve: 0.8922
```

```
# Subsection 4.2.3 Feature Importance Analysis

# Analyzing Feature Importance in XGBoost Model

# Retrieving the final XGBoost model
final_xgb_model <- xgb_model$finalModel

# Extracting and visualizing feature importance
feature_importance_matrix <- xgb.importance(model = final_xgb_model)
xgb.plot.importance(feature_importance_matrix)
```

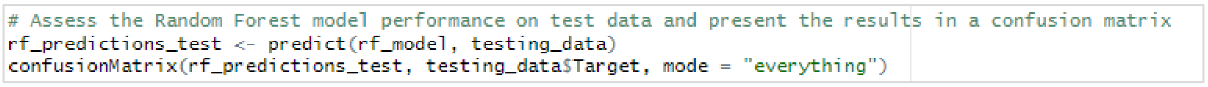




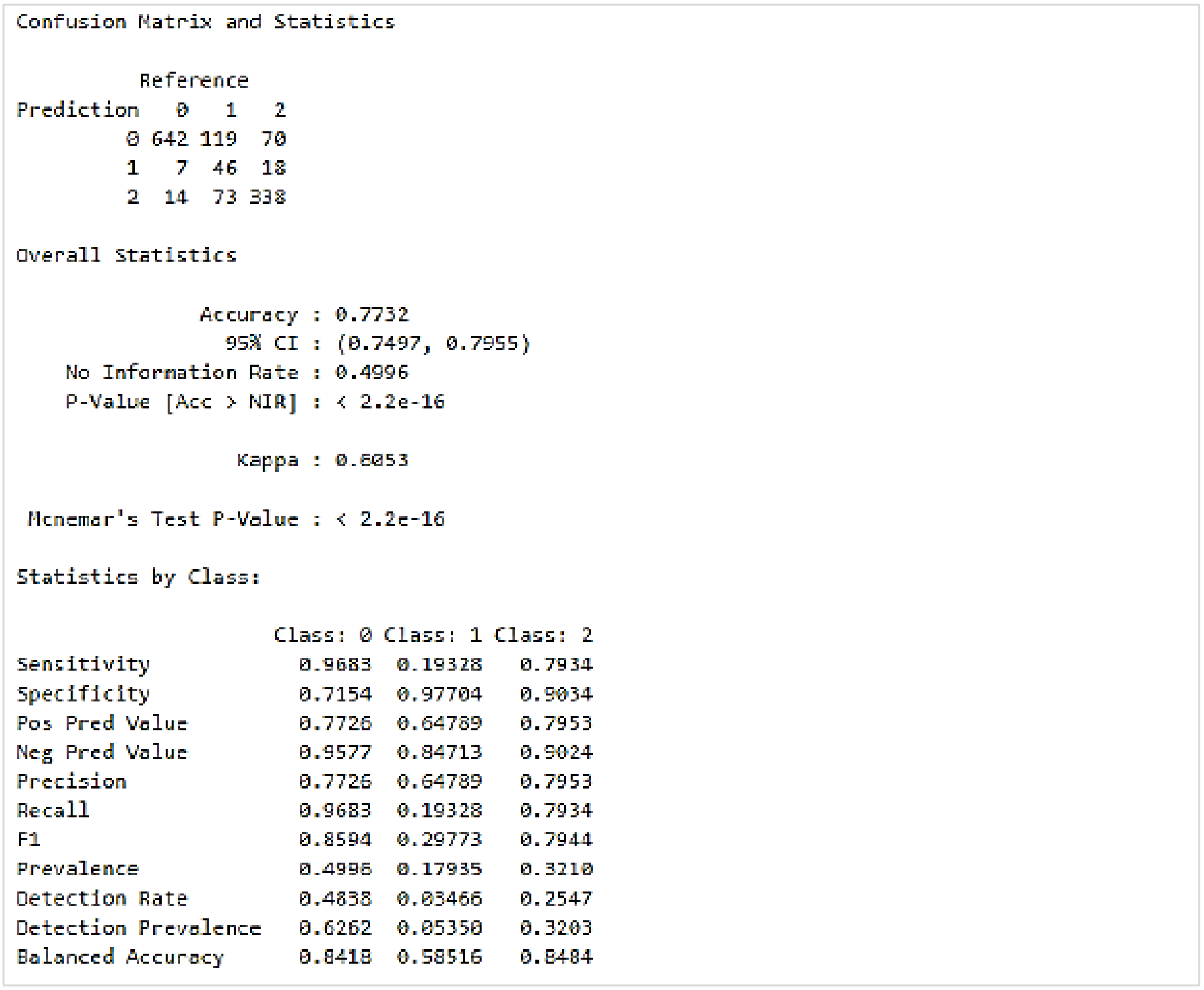
```
# Subsection 4.3: Building a Random Forest Model

# Subsection 4.3.1: Initial Model Training

# Set up parameters for training without cross-validation
train_control <- trainControl(method = "none", verboseIter = T,  allowParallel = TRUE)

# Develop a Random Forest model using the training dataset
set.seed(321)
rf_model <- train(Target ~ ., data = training_data, method = "rf", trControl = train_control)

# Test the Random Forest model on training data and display the results using a confusion matrix
rf_predictions <- predict(rf_model, training_data)
confusionMatrix(rf_predictions, training_data$Target, mode = "everything")
```



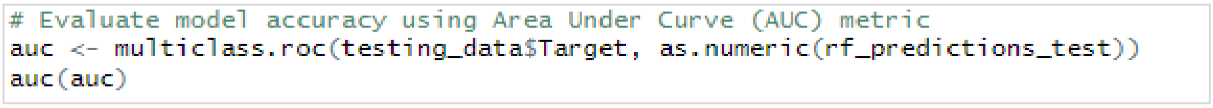
```
Confusion Matrix and Statistics

          Reference
Prediction    0    1    2
         0 1546    0    0
         1    0  556    0
         2    0    0  995

Overall Statistics

               Accuracy : 1
                 95% CI : (0.9988, 1)
    No Information Rate : 0.4992
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            1.0000   1.0000   1.0000
Specificity            1.0000   1.0000   1.0000
Pos Pred Value         1.0000   1.0000   1.0000
Neg Pred Value         1.0000   1.0000   1.0000
Precision              1.0000   1.0000   1.0000
Recall                 1.0000   1.0000   1.0000
F1                     1.0000   1.0000   1.0000
Prevalence             0.4992   0.1795   0.3213
Detection Rate         0.4992   0.1795   0.3213
Detection Prevalence   0.4992   0.1795   0.3213
Balanced Accuracy      1.0000   1.0000   1.0000
```



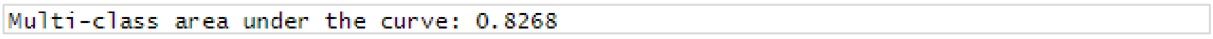
```
# Assess the Random Forest model performance on test data and present the results in a confusion matrix
rf_predictions_test <- predict(rf_model, testing_data)
confusionMatrix(rf_predictions_test, testing_data$Target, mode = "everything")
```



```
Confusion Matrix and Statistics

          Reference
Prediction   0   1   2
         0 642 119  70
         1   7  46  18
         2  14  73 338

Overall Statistics

               Accuracy : 0.7732
                 95% CI : (0.7497, 0.7955)
    No Information Rate : 0.4996
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6053

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            0.9683  0.19328   0.7934
Specificity            0.7154  0.97704   0.9034
Pos Pred Value         0.7726  0.64789   0.7953
Neg Pred Value         0.9577  0.84713   0.9024
Precision              0.7726  0.64789   0.7953
Recall                 0.9683  0.19328   0.7934
F1                     0.8594  0.29773   0.7944
Prevalence             0.4996  0.17935   0.3210
Detection Rate         0.4838  0.03466   0.2547
Detection Prevalence   0.6262  0.05350   0.3203
Balanced Accuracy      0.8418  0.58516   0.8484
```



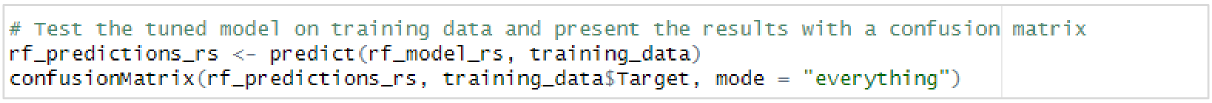
```
# Evaluate model accuracy using Area Under Curve (AUC) metric
auc <- multiclass.roc(testing_data$Target, as.numeric(rf_predictions_test))
auc(auc)
```



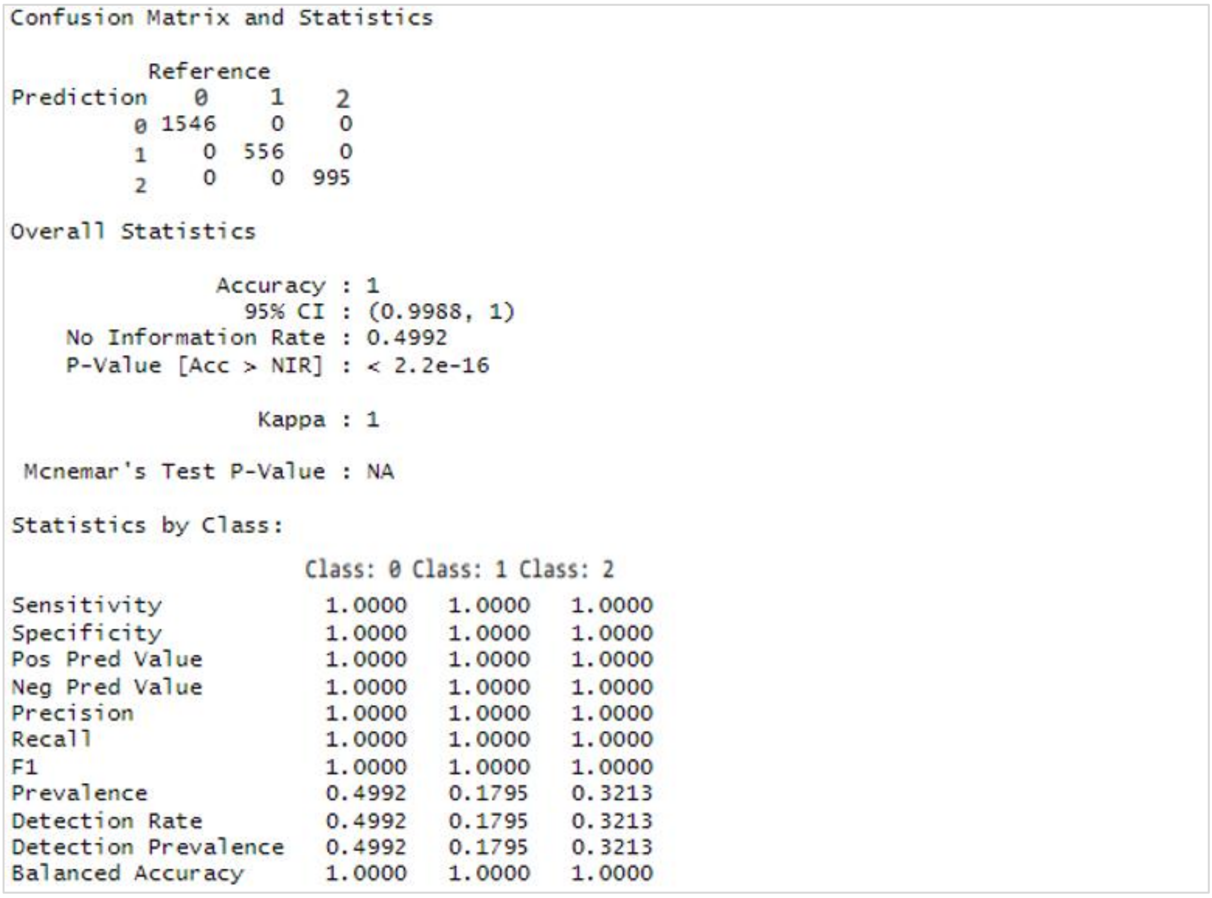
```
Multi-class area under the curve: 0.8268
```

```
# Subsection 4.3.2: Hyperparameter Tuning

# Set up random search for optimizing model hyperparameters with cross-validation
train_control <- trainControl(method = "cv", number = 2, search="random", allowParallel = TRUE)

# Execute random search to fine-tune the Random Forest model
set.seed(321)
rf_model_rs <- train(Target ~ ., data = training_data, method = "rf", trControl = train_control,
                     metric = "Accuracy", tuneLength = 15)

# Print the details of the model
print(rf_model_rs)
```

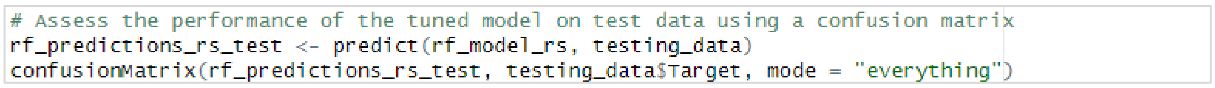
```
Random Forest

3097 samples
  36 predictor
   3 classes: '0', '1', '2'

No pre-processing
Resampling: Cross-Validated (2 fold)
Summary of sample sizes: 1549, 1548
Resampling results across tuning parameters:

  mtry  Accuracy   Kappa
  11    0.7781748  0.6262675
  13    0.7791425  0.6289887
  15    0.7739777  0.6207898
  16    0.7759140  0.6240351
  17    0.7784980  0.6285591
  18    0.7762381  0.6254905
  24    0.7746229  0.6227164
  25    0.7723623  0.6186776
  31    0.7707490  0.6161037
  32    0.7710724  0.6169267
  34    0.7720408  0.6189307
  36    0.7688123  0.6139098

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 13.
```



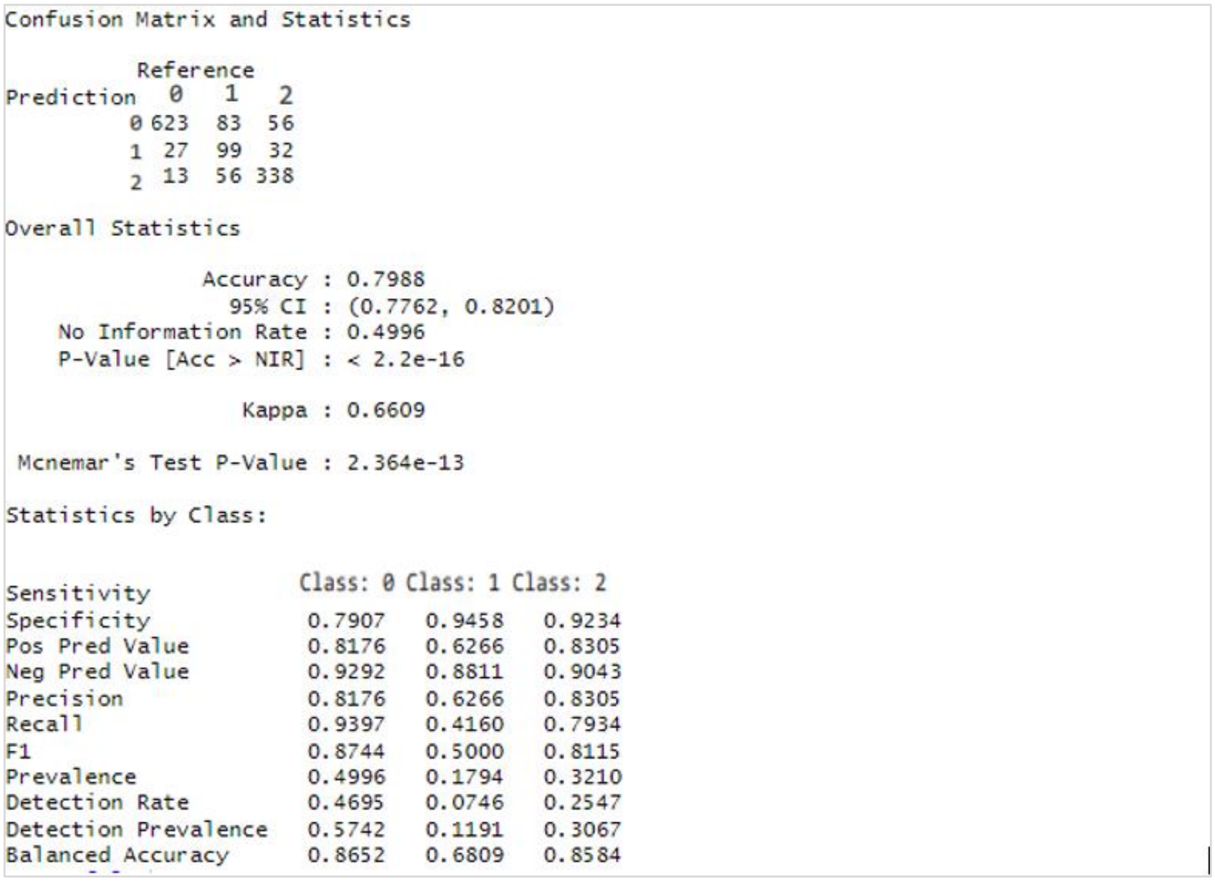
```
# Test the tuned model on training data and present the results with a confusion matrix
rf_predictions_rs <- predict(rf_model_rs, training_data)
confusionMatrix(rf_predictions_rs, training_data$Target, mode = "everything")
```



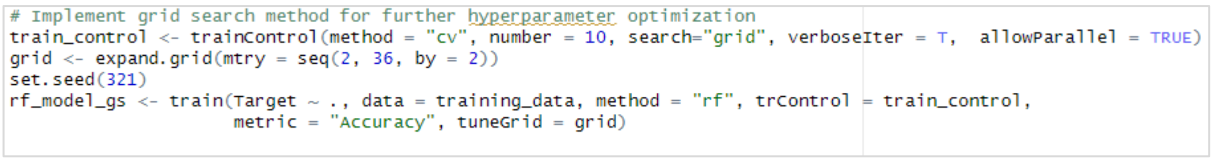
```
Confusion Matrix and Statistics

          Reference
Prediction    0    1    2
         0 1546    0    0
         1    0  556    0
         2    0    0  995

Overall Statistics

               Accuracy : 1
                 95% CI : (0.9988, 1)
    No Information Rate : 0.4992
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            1.0000   1.0000   1.0000
Specificity            1.0000   1.0000   1.0000
Pos Pred Value         1.0000   1.0000   1.0000
Neg Pred Value         1.0000   1.0000   1.0000
Precision              1.0000   1.0000   1.0000
Recall                 1.0000   1.0000   1.0000
F1                     1.0000   1.0000   1.0000
Prevalence             0.4992   0.1795   0.3213
Detection Rate         0.4992   0.1795   0.3213
Detection Prevalence   0.4992   0.1795   0.3213
Balanced Accuracy      1.0000   1.0000   1.0000
```



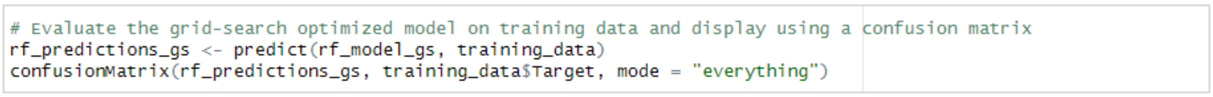
```
# Assess the performance of the tuned model on test data using a confusion matrix
rf_predictions_rs_test <- predict(rf_model_rs, testing_data)
confusionMatrix(rf_predictions_rs_test, testing_data$Target, mode = "everything")
```



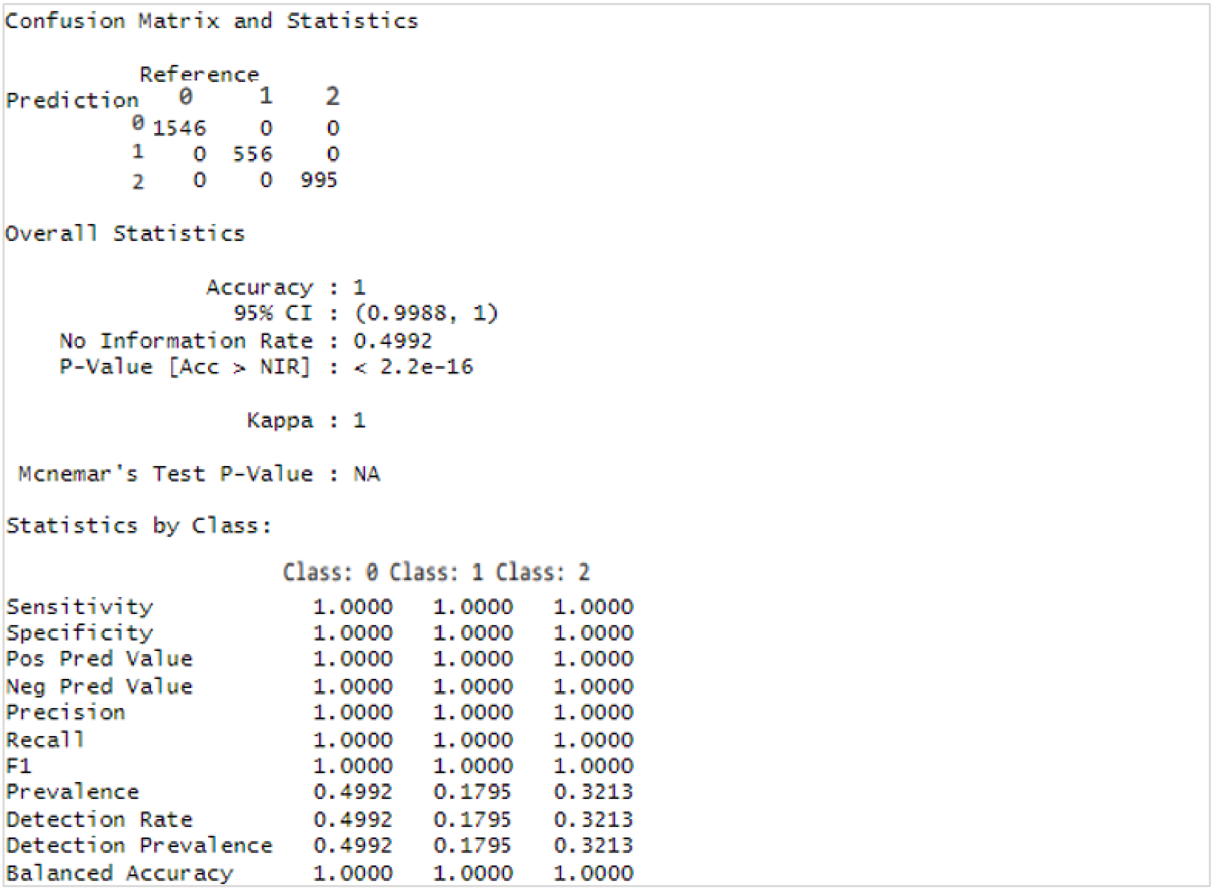
```
Confusion Matrix and Statistics

          Reference
Prediction   0   1   2
         0 623  83  56
         1  27  99  32
         2  13  56 338

Overall Statistics

               Accuracy : 0.7988
                 95% CI : (0.7762, 0.8201)
    No Information Rate : 0.4996
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6609

 Mcnemar's Test P-Value : 2.364e-13

Statistics by Class:

                      Class: 0 Class: 1 Class: 2
Sensitivity
Specificity             0.7907   0.9458   0.9234
Pos Pred Value          0.8176   0.6266   0.8305
Neg Pred Value          0.9292   0.8811   0.9043
Precision               0.8176   0.6266   0.8305
Recall                  0.9397   0.4160   0.7934
F1                      0.8744   0.5000   0.8115
Prevalence              0.4996   0.1794   0.3210
Detection Rate          0.4695   0.0746   0.2547
Detection Prevalence    0.5742   0.1191   0.3067
Balanced Accuracy       0.8652   0.6809   0.8584
```



```r
# Implement grid search method for further hyperparameter optimization
train_control <- trainControl(method = "cv", number = 10, search="grid", verboseIter = T,  allowParallel = TRUE)
grid <- expand.grid(mtry = seq(2, 36, by = 2))
set.seed(321)
rf_model_gs <- train(Target ~ ., data = training_data, method = "rf", trControl = train_control,
                     metric = "Accuracy", tuneGrid = grid)
```



```r
# Evaluate the grid-search optimized model on training data and display using a confusion matrix
rf_predictions_gs <- predict(rf_model_gs, training_data)
confusionMatrix(rf_predictions_gs, training_data$Target, mode = "everything")
```



```
Confusion Matrix and Statistics

          Reference
Prediction    0    1    2
         0 1546    0    0
         1    0  556    0
         2    0    0  995

Overall Statistics

               Accuracy : 1
                 95% CI : (0.9988, 1)
    No Information Rate : 0.4992
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            1.0000   1.0000   1.0000
Specificity            1.0000   1.0000   1.0000
Pos Pred Value         1.0000   1.0000   1.0000
Neg Pred Value         1.0000   1.0000   1.0000
Precision              1.0000   1.0000   1.0000
Recall                 1.0000   1.0000   1.0000
F1                     1.0000   1.0000   1.0000
Prevalence             0.4992   0.1795   0.3213
Detection Rate         0.4992   0.1795   0.3213
Detection Prevalence   0.4992   0.1795   0.3213
Balanced Accuracy      1.0000   1.0000   1.0000
```

```
# Test the model with test data and showcase results with a confusion matrix
rf_predictions_gs_test <- predict(rf_model_gs, testing_data)

# Present confusion matrix with detailed metrics
rf_metrics <- confusionMatrix(rf_predictions_gs_test, testing_data$Target, mode = "everything")
rf_metrics
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1   2
         0 638  97  57
         1  16  80  26
         2   9  61 343

Overall Statistics

               Accuracy : 0.7995
                 95% CI : (0.777, 0.8208)
    No Information Rate : 0.4996
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6576

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            0.9623  0.33613   0.8052
Specificity            0.7681  0.96143   0.9223
Pos Pred Value         0.8056  0.65574   0.8305
Neg Pred Value         0.9533  0.86888   0.9092
Precision              0.8056  0.65574   0.8305
Recall                 0.9623  0.33613   0.8052
F1                     0.8770  0.44444   0.8176
Prevalence             0.4996  0.17935   0.3210
Detection Rate         0.4808  0.06029   0.2585
Detection Prevalence   0.5968  0.09194   0.3112
Balanced Accuracy      0.8652  0.64878   0.8637
```

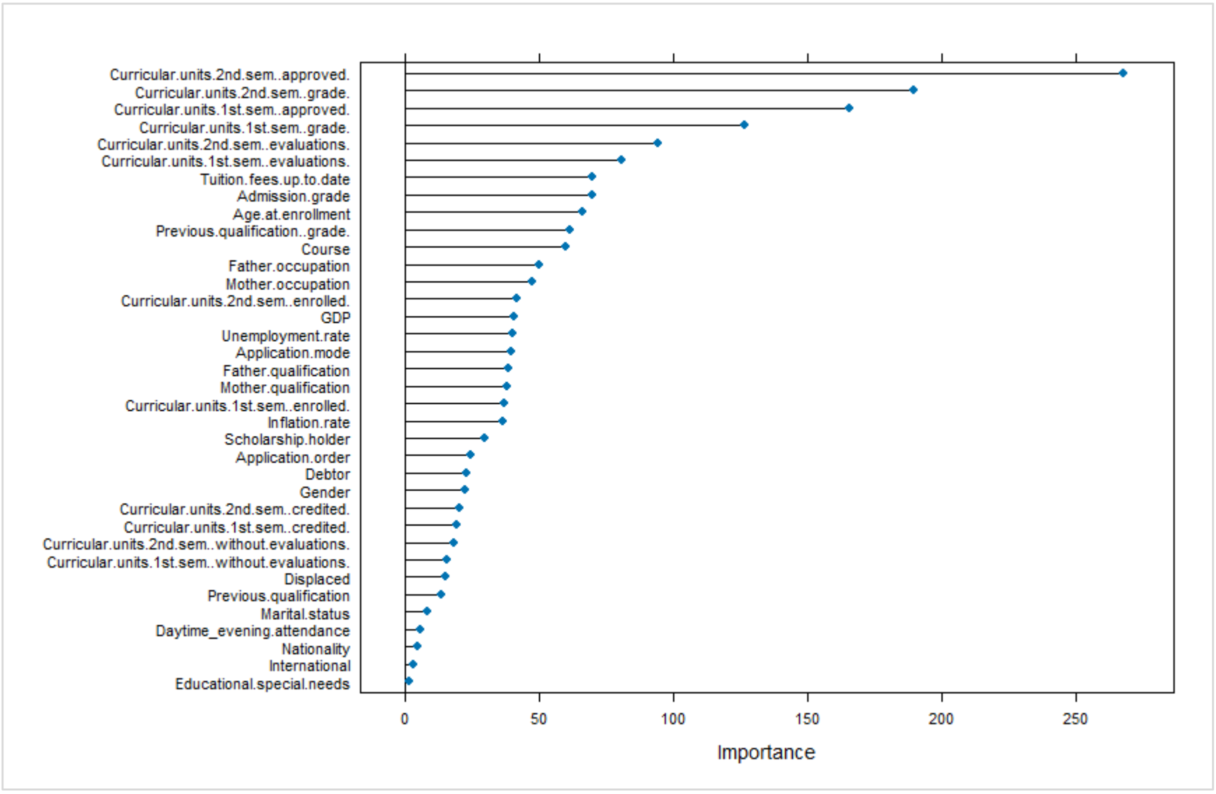
```
# Measure model performance using Area Under the Curve
# Employ the multiclass.roc() function for AUC calculation
rf_auc <- multiclass.roc(testing_data$Target, as.numeric(rf_predictions_gs_test))
auc(rf_auc)
```

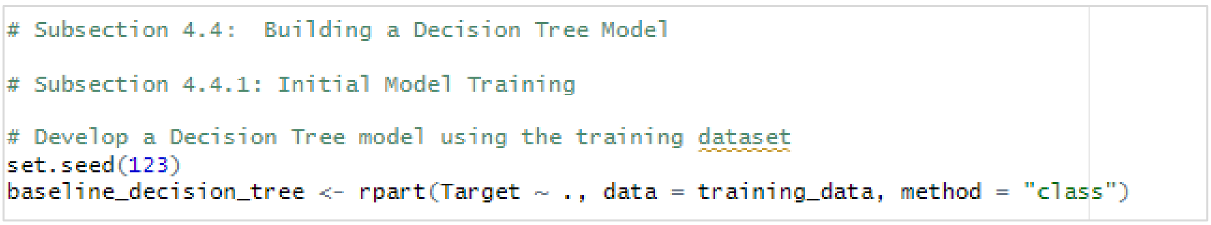
```
Multi-class area under the curve: 0.822
```

```
# Subsection 4.3.3: Feature Importance Analysis

# Assess and select key features based on their significance
# Determine the importance of each feature
feature_importance <- varImp(rf_model_gs, scale = FALSE)
print(feature_importance)
```

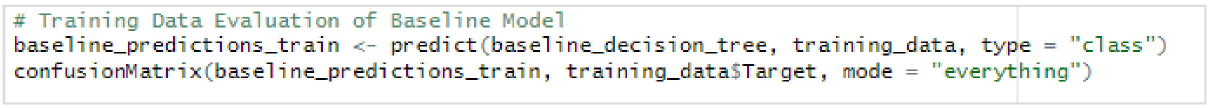
```
rf variable importance

  only 20 most important variables shown (out of 36)

                                          Overall
Curricular.units.2nd.sem..approved.        267.42
Curricular.units.2nd.sem..grade.           189.43
Curricular.units.1st.sem..approved.        165.58
Curricular.units.1st.sem..grade.           126.22
Curricular.units.2nd.sem..evaluations.      94.15
Curricular.units.1st.sem..evaluations.      80.71
Tuition.fees.up.to.date                     69.58
Admission.grade                             69.52
Age.at.enrollment                           65.89
Previous.qualification..grade.              61.05
Course                                      59.67
Father.occupation                           49.80
Mother.occupation                           47.43
Curricular.units.2nd.sem..enrolled.         41.38
GDP                                         40.58
Unemployment.rate                           39.98
Application.mode                            39.46
Father.qualification                        38.42
Mother.qualification                        37.63
Curricular.units.1st.sem..enrolled.         36.97
```

```
# Visualize the significance of each feature
plot(feature_importance)
```

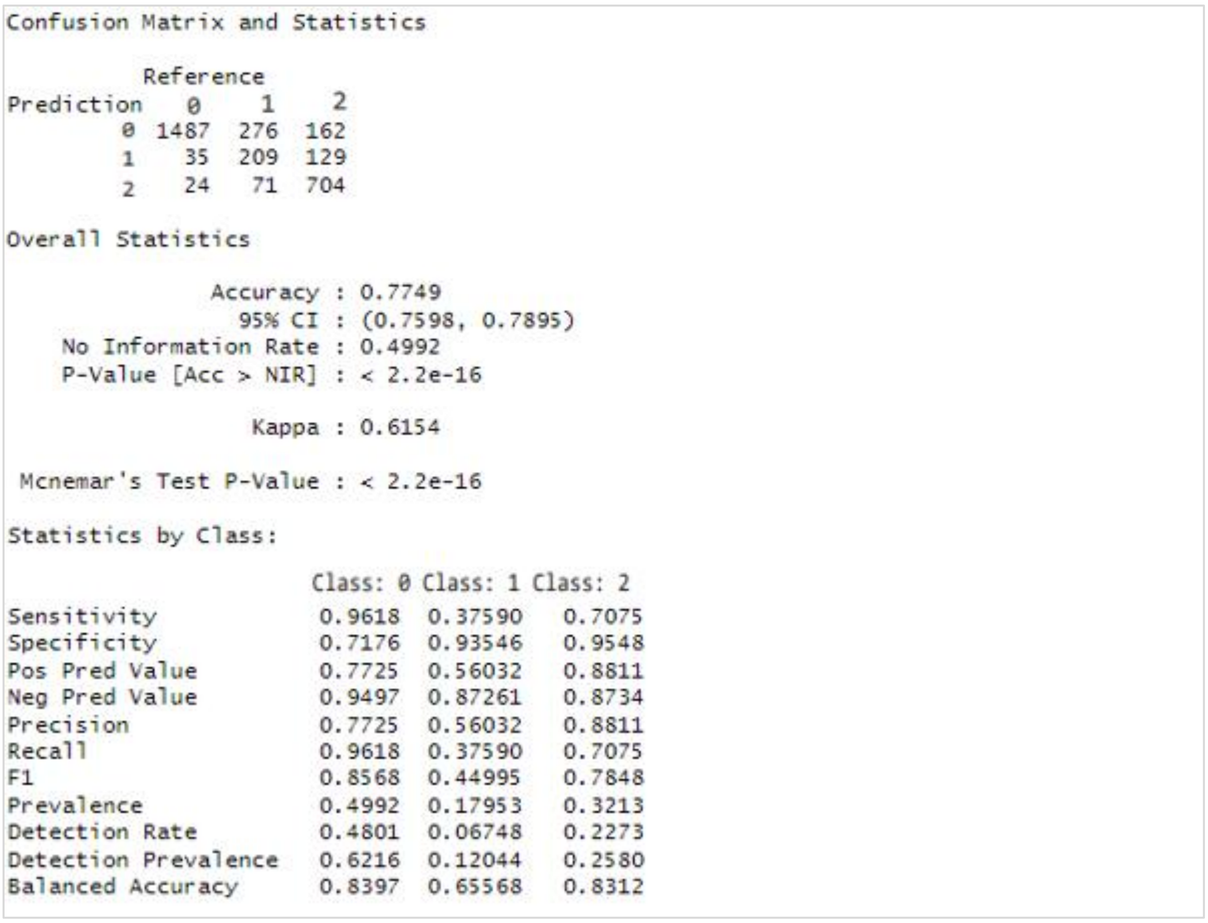




```
# Subsection 4.4:  Building a Decision Tree Model

# Subsection 4.4.1: Initial Model Training

# Develop a Decision Tree model using the training dataset
set.seed(123)
baseline_decision_tree <- rpart(Target ~ ., data = training_data, method = "class")
```



```
# Training Data Evaluation of Baseline Model
baseline_predictions_train <- predict(baseline_decision_tree, training_data, type = "class")
confusionMatrix(baseline_predictions_train, training_data$Target, mode = "everything")
```



```
Confusion Matrix and Statistics

          Reference
Prediction    0    1    2
         0 1487  276  162
         1   35  209  129
         2   24   71  704

Overall Statistics

               Accuracy : 0.7749
                 95% CI : (0.7598, 0.7895)
    No Information Rate : 0.4992
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6154

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            0.9618  0.37590   0.7075
Specificity            0.7176  0.93546   0.9548
Pos Pred Value         0.7725  0.56032   0.8811
Neg Pred Value         0.9497  0.87261   0.8734
Precision              0.7725  0.56032   0.8811
Recall                 0.9618  0.37590   0.7075
F1                     0.8568  0.44995   0.7848
Prevalence             0.4992  0.17953   0.3213
Detection Rate         0.4801  0.06748   0.2273
Detection Prevalence   0.6216  0.12044   0.2580
Balanced Accuracy      0.8397  0.65568   0.8312
```

```
# Test Data Evaluation of Baseline Model
baseline_predictions_test <- predict(baseline_decision_tree, testing_data, type = "class")
confusionMatrix(baseline_predictions_test, testing_data$Target, mode = "everything")
```

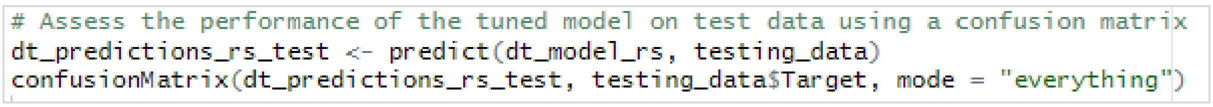
```
Confusion Matrix and Statistics

          Reference
Prediction   0   1   2
         0 637 118  73
         1  13  88  50
         2  13  32 303

Overall Statistics

               Accuracy : 0.7747
                 95% CI : (0.7512, 0.7969)
    No Information Rate : 0.4996
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.614

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            0.9608  0.36975   0.7113
Specificity            0.7123  0.94215   0.9501
Pos Pred Value         0.7693  0.58278   0.8707
Neg Pred Value         0.9479  0.87245   0.8744
Precision              0.7693  0.58278   0.8707
Recall                 0.9608  0.36975   0.7113
F1                     0.8545  0.45244   0.7829
Prevalence             0.4996  0.17935   0.3210
Detection Rate         0.4800  0.06631   0.2283
Detection Prevalence   0.6240  0.11379   0.2622
Balanced Accuracy      0.8366  0.65595   0.8307
```

```
# Subsection 4.4.2: Hyperparameter Tuning

# Set up random search for optimizing model hyperparameters with cross-validation
train_control_rs <- trainControl(method = "cv", number = 2, search = "random", allowParallel = TRUE)

# Execute random search to fine-tune the Decision Tree model
set.seed(321)
dt_model_rs <- train(Target ~ ., data = training_data, method = "rpart", trControl = train_control_rs,
                     metric = "Accuracy", tuneLength = 15)

# Print the details of the model
print(dt_model_rs)
```

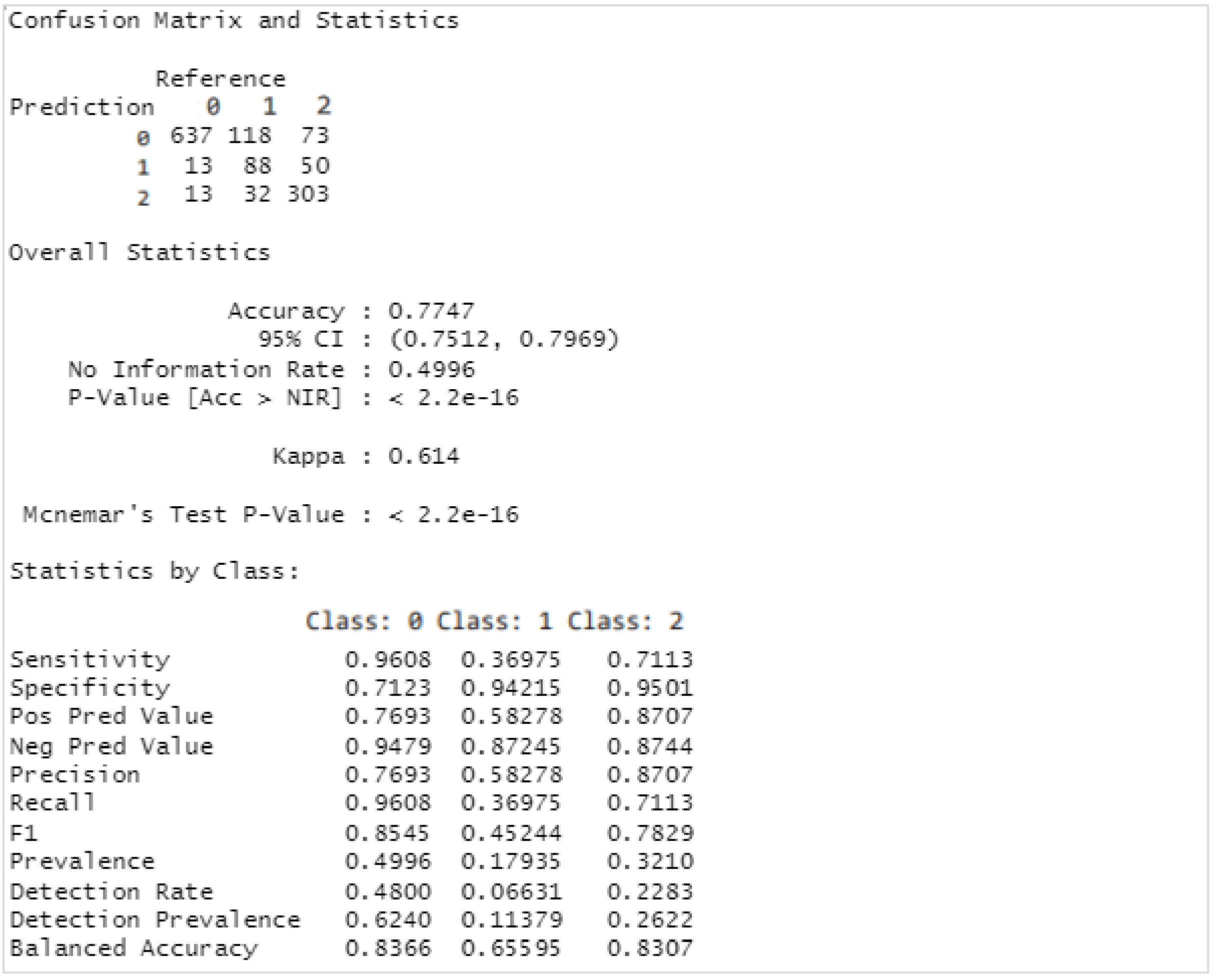
```
CART

3097 samples
  36 predictor
   3 classes: '0', '1', '2'

No pre-processing
Resampling: Cross-Validated (2 fold)
Summary of sample sizes: 1549, 1548
Resampling results across tuning parameters:

  cp            Accuracy   Kappa
  0.0001611863  0.7478245  0.5801694
  0.0008596604  0.7478245  0.5790635
  0.0010745755  0.7478245  0.5790635
  0.0016118633  0.7494382  0.5809793
  0.0019342360  0.7500838  0.5817553
  0.0025789813  0.7536358  0.5862024
  0.0027938964  0.7542813  0.5886608
  0.0032237266  0.7533123  0.5848840
  0.0096711799  0.7546035  0.5804651
  0.4074790458  0.7010020  0.4730469

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was cp = 0.00967118.
```

```
# Test the tuned model on training data and present the results with a confusion matrix
dt_predictions_rs <- predict(dt_model_rs, training_data)
confusionMatrix(dt_predictions_rs, training_data$Target, mode = "everything")
```

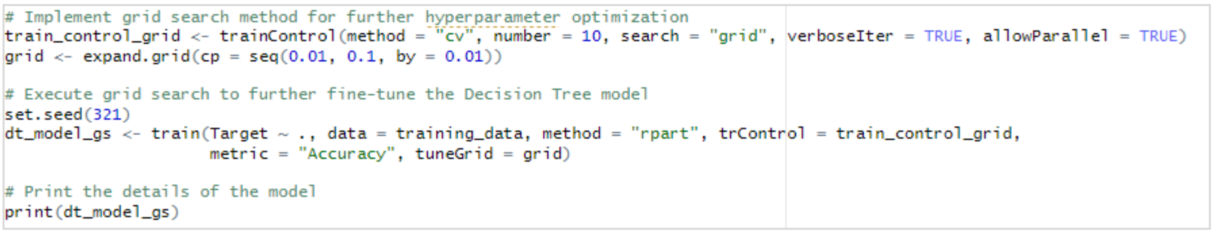
```
Confusion Matrix and Statistics

          Reference
Prediction    0    1    2
         0 1487  276  162
         1   35  209  129
         2   24   71  704

Overall Statistics

               Accuracy : 0.7749
                 95% CI : (0.7598, 0.7895)
    No Information Rate : 0.4992
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6154

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            0.9618  0.37590   0.7075
Specificity            0.7176  0.93546   0.9548
Pos Pred Value         0.7725  0.56032   0.8811
Neg Pred Value         0.9497  0.87261   0.8734
Precision              0.7725  0.56032   0.8811
Recall                 0.9618  0.37590   0.7075
F1                     0.8568  0.44995   0.7848
Prevalence             0.4992  0.17953   0.3213
Detection Rate         0.4801  0.06748   0.2273
Detection Prevalence   0.6216  0.12044   0.2580
Balanced Accuracy      0.8397  0.65568   0.8312
```



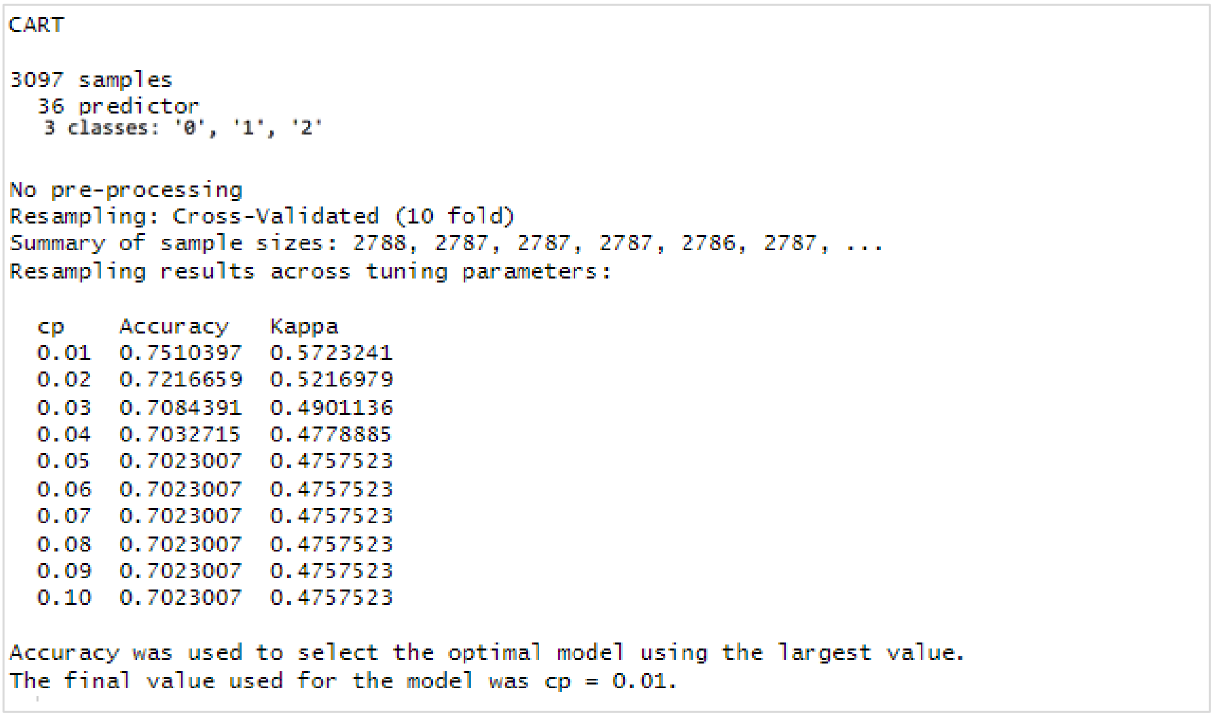
```
# Assess the performance of the tuned model on test data using a confusion matrix
dt_predictions_rs_test <- predict(dt_model_rs, testing_data)
confusionMatrix(dt_predictions_rs_test, testing_data$Target, mode = "everything")
```



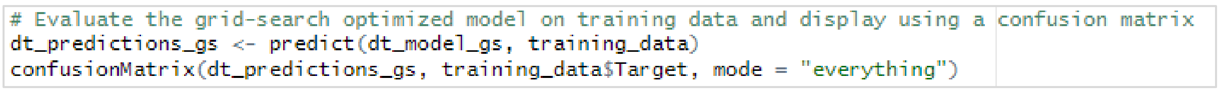
```
Confusion Matrix and Statistics

          Reference
Prediction   0   1   2
         0 637 118  73
         1  13  88  50
         2  13  32 303

Overall Statistics

               Accuracy : 0.7747
                 95% CI : (0.7512, 0.7969)
    No Information Rate : 0.4996
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.614

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            0.9608  0.36975   0.7113
Specificity            0.7123  0.94215   0.9501
Pos Pred Value         0.7693  0.58278   0.8707
Neg Pred Value         0.9479  0.87245   0.8744
Precision              0.7693  0.58278   0.8707
Recall                 0.9608  0.36975   0.7113
F1                     0.8545  0.45244   0.7829
Prevalence             0.4996  0.17935   0.3210
Detection Rate         0.4800  0.06631   0.2283
Detection Prevalence   0.6240  0.11379   0.2622
Balanced Accuracy      0.8366  0.65595   0.8307
```



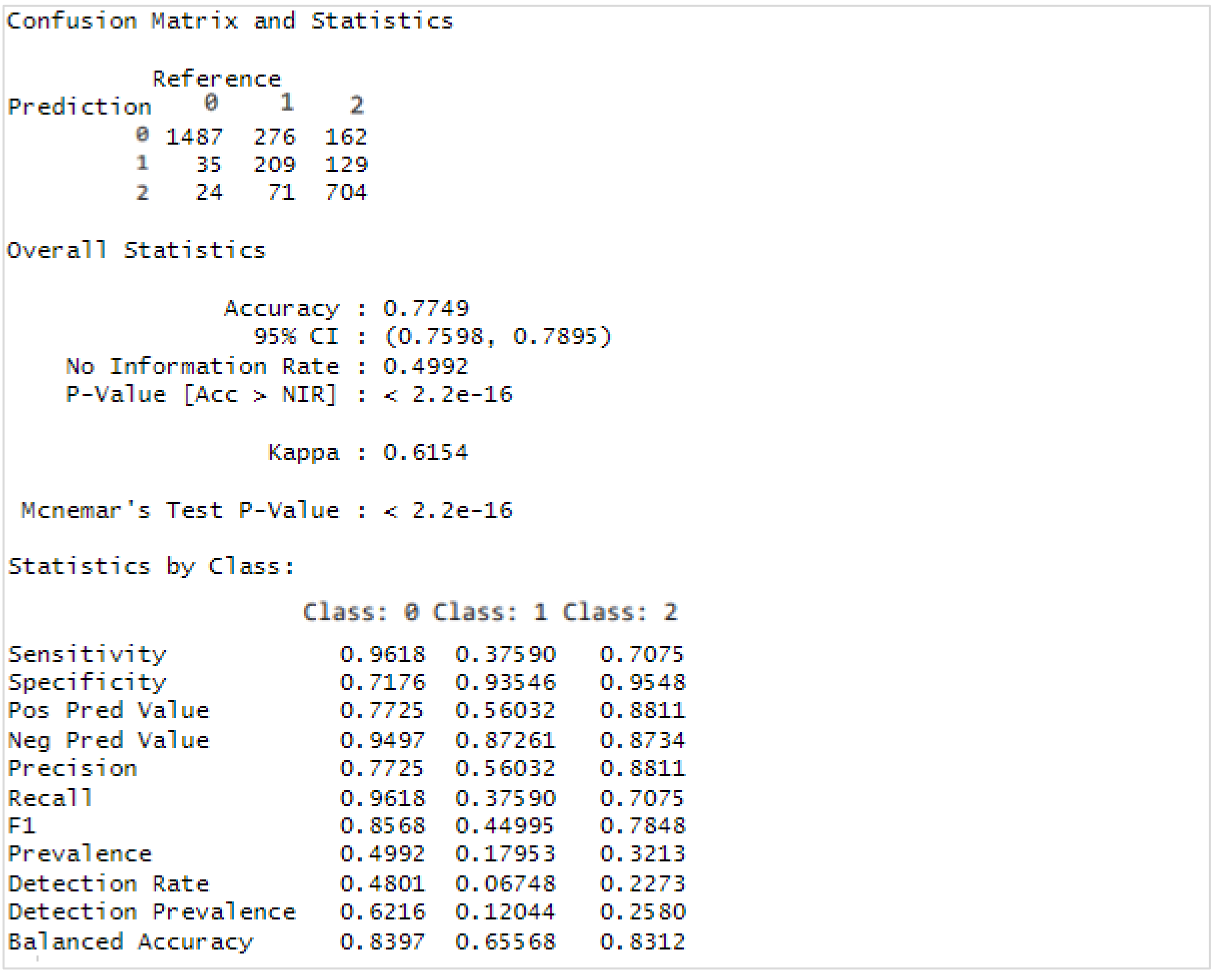
```
# Implement grid search method for further hyperparameter optimization
train_control_grid <- trainControl(method = "cv", number = 10, search = "grid", verboseIter = TRUE, allowParallel = TRUE)
grid <- expand.grid(cp = seq(0.01, 0.1, by = 0.01))

# Execute grid search to further fine-tune the Decision Tree model
set.seed(321)
dt_model_gs <- train(Target ~ ., data = training_data, method = "rpart", trControl = train_control_grid,
                     metric = "Accuracy", tuneGrid = grid)

# Print the details of the model
print(dt_model_gs)
```



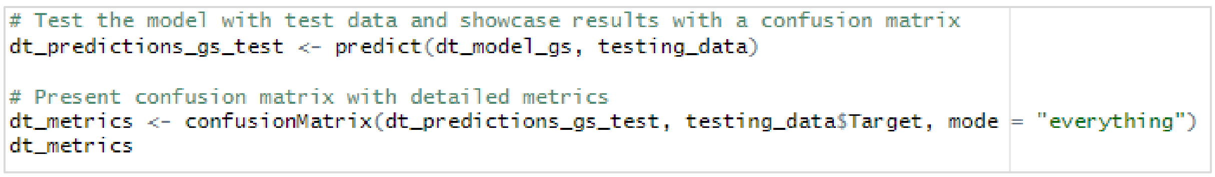
```
CART

3097 samples
  36 predictor
   3 classes: '0', '1', '2'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 2788, 2787, 2787, 2787, 2786, 2787, ...
Resampling results across tuning parameters:

  cp    Accuracy   Kappa
  0.01  0.7510397  0.5723241
  0.02  0.7216659  0.5216979
  0.03  0.7084391  0.4901136
  0.04  0.7032715  0.4778885
  0.05  0.7023007  0.4757523
  0.06  0.7023007  0.4757523
  0.07  0.7023007  0.4757523
  0.08  0.7023007  0.4757523
  0.09  0.7023007  0.4757523
  0.10  0.7023007  0.4757523

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was cp = 0.01.
```



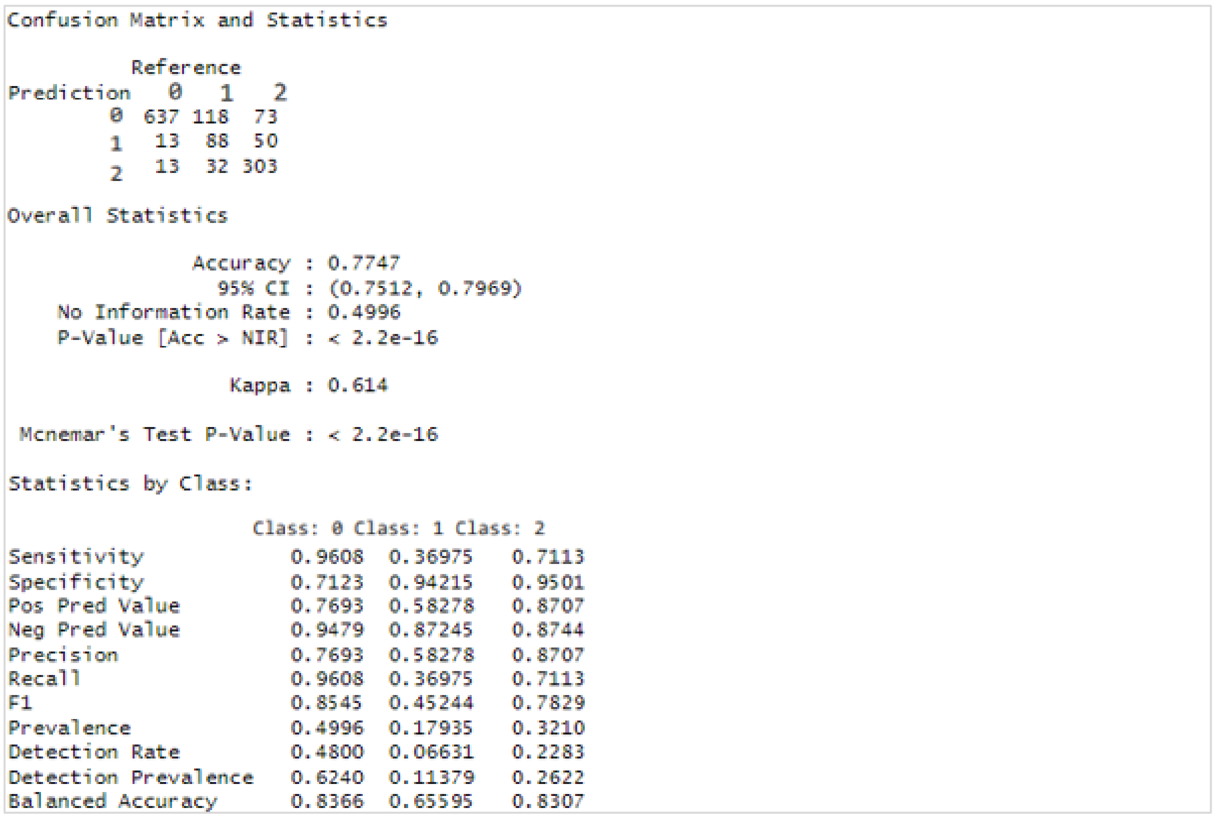
```
# Evaluate the grid-search optimized model on training data and display using a confusion matrix
dt_predictions_gs <- predict(dt_model_gs, training_data)
confusionMatrix(dt_predictions_gs, training_data$Target, mode = "everything")
```



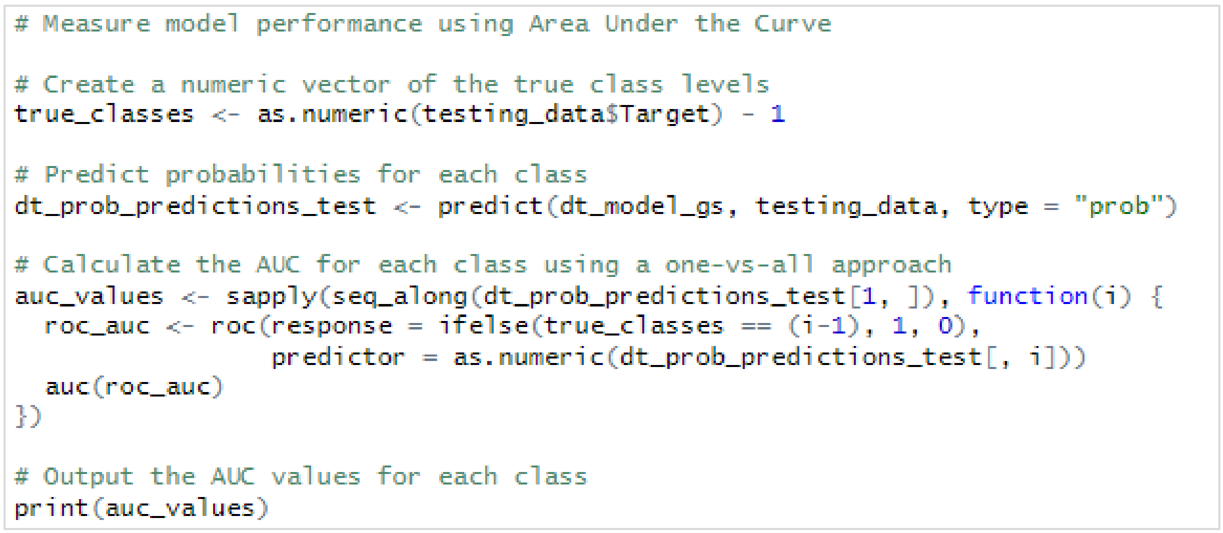
```
Confusion Matrix and Statistics

          Reference
Prediction    0    1    2
         0 1487  276  162
         1   35  209  129
         2   24   71  704

Overall Statistics

               Accuracy : 0.7749
                 95% CI : (0.7598, 0.7895)
    No Information Rate : 0.4992
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6154

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            0.9618  0.37590   0.7075
Specificity            0.7176  0.93546   0.9548
Pos Pred Value         0.7725  0.56032   0.8811
Neg Pred Value         0.9497  0.87261   0.8734
Precision              0.7725  0.56032   0.8811
Recall                 0.9618  0.37590   0.7075
F1                     0.8568  0.44995   0.7848
Prevalence             0.4992  0.17953   0.3213
Detection Rate         0.4801  0.06748   0.2273
Detection Prevalence   0.6216  0.12044   0.2580
Balanced Accuracy      0.8397  0.65568   0.8312
```



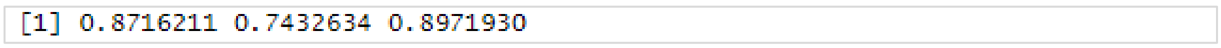
```
# Test the model with test data and showcase results with a confusion matrix
dt_predictions_gs_test <- predict(dt_model_gs, testing_data)

# Present confusion matrix with detailed metrics
dt_metrics <- confusionMatrix(dt_predictions_gs_test, testing_data$Target, mode = "everything")
dt_metrics
```



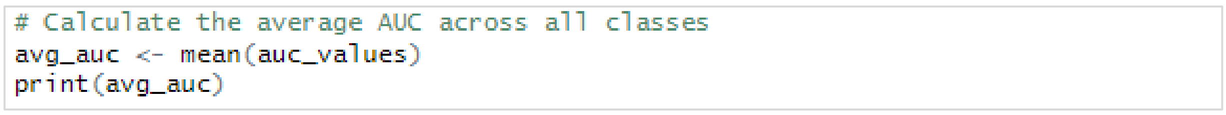
```
Confusion Matrix and Statistics

          Reference
Prediction   0   1   2
         0 637 118  73
         1  13  88  50
         2  13  32 303

Overall Statistics

               Accuracy : 0.7747
                 95% CI : (0.7512, 0.7969)
    No Information Rate : 0.4996
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.614

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: 0 Class: 1 Class: 2
Sensitivity            0.9608  0.36975   0.7113
Specificity            0.7123  0.94215   0.9501
Pos Pred Value         0.7693  0.58278   0.8707
Neg Pred Value         0.9479  0.87245   0.8744
Precision              0.7693  0.58278   0.8707
Recall                 0.9608  0.36975   0.7113
F1                     0.8545  0.45244   0.7829
Prevalence             0.4996  0.17935   0.3210
Detection Rate         0.4800  0.06631   0.2283
Detection Prevalence   0.6240  0.11379   0.2622
Balanced Accuracy      0.8366  0.65595   0.8307
```



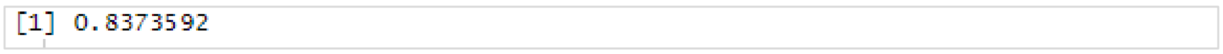
```
# Measure model performance using Area Under the Curve

# Create a numeric vector of the true class levels
true_classes <- as.numeric(testing_data$Target) - 1

# Predict probabilities for each class
dt_prob_predictions_test <- predict(dt_model_gs, testing_data, type = "prob")

# Calculate the AUC for each class using a one-vs-all approach
auc_values <- sapply(seq_along(dt_prob_predictions_test[1, ]), function(i) {
  roc_auc <- roc(response = ifelse(true_classes == (i-1), 1, 0),
                 predictor = as.numeric(dt_prob_predictions_test[, i]))
  auc(roc_auc)
})

# Output the AUC values for each class
print(auc_values)
```



```
 [1] 0.8716211 0.7432634 0.8971930
```



```
# Calculate the average AUC across all classes
avg_auc <- mean(auc_values)
print(avg_auc)
```

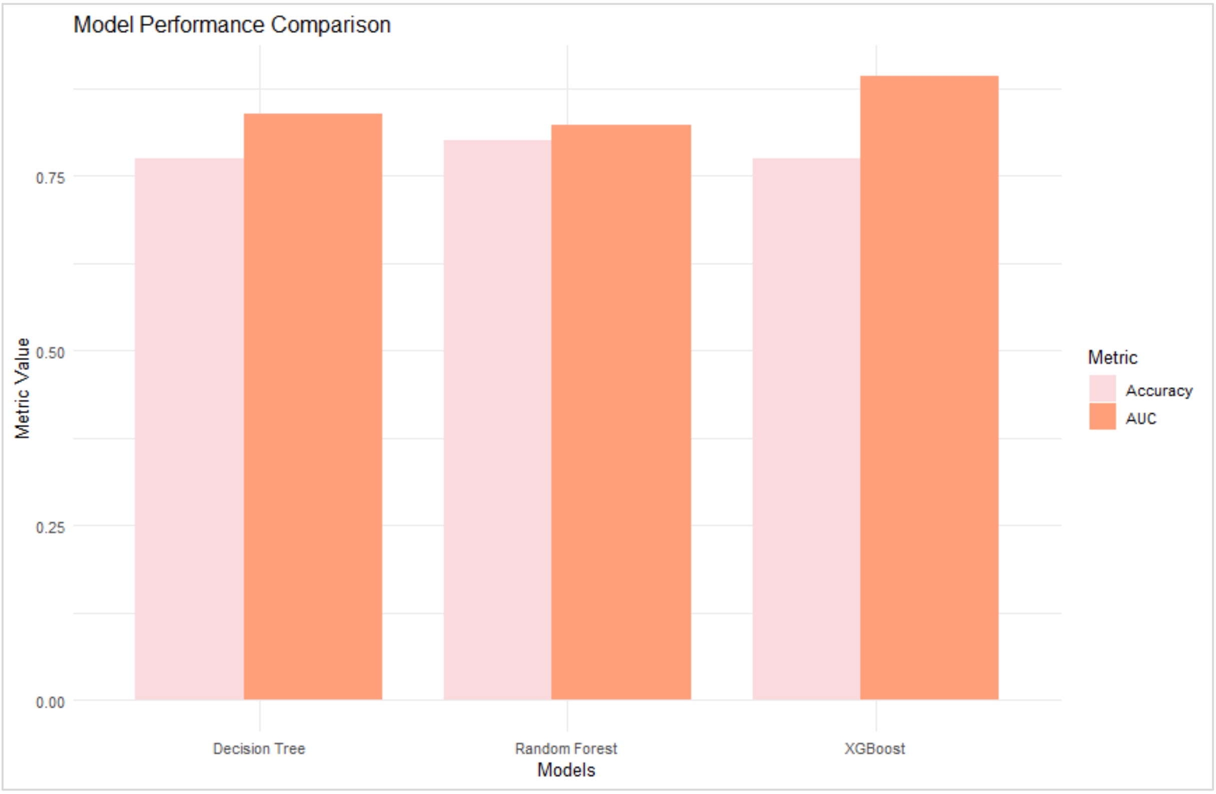


```
[1] 0.8373592
```

```
# Section 5: Validation of All Three Models

# Extracting AUC values for each model
rf_auc_value <- auc(rf_auc)              # AUC for Random Forest
dt_auc_value <- avg_auc                  # Average AUC for Decision Tree (after tuning)
xgb_auc_value <- auc(xgb_auc)            # AUC for XGBoost

# Assembling a data frame with model performance metrics
comparison_df <- data.frame(
  Model = c("Random Forest", "Decision Tree", "XGBoost"),
  Accuracy = c(rf_metrics$overall['Accuracy'], dt_metrics$overall['Accuracy'], xgb_metrics$overall['Accuracy']),
  AUC = c(rf_auc_value, dt_auc_value, xgb_auc_value)
)

# Displaying comparative model metrics
print(comparison_df)
```

```
          Model  Accuracy       AUC
1 Random Forest 0.7995479 0.8220341
2 Decision Tree 0.7746797 0.8373592
3       XGBoost 0.7739261 0.8922373
```

```
# Reshape the data from wide to long format
long_comparison_df <- melt(comparison_df, id.vars = "Model", variable.name = "Metric", value.name = "Value")

# Creating a bar plot to compare model accuracies and AUC values with pastel colors
ggplot(data = long_comparison_df, aes(x = Model, y = Value, fill = Metric)) +
  geom_bar(stat = "identity", position = position_dodge(width = 0.7)) +
  ylab("Metric Value") +
  xlab("Models") +
  ggtitle("Model Performance Comparison") +
  scale_fill_manual(values = c("Accuracy" = "#FADADD", "AUC" = "#FFA07A")) +
  theme_minimal()
```


Model Performance Comparison
Metric Value
Decision Tree
Random Forest
XGBoost
Models
Metric
Accuracy
AUC