MLG382: GUIDED PROJECT DOCUMENTATION

Team Members:
- Stiaan Megit (600819)
- Teleki Shai (601377)
- Jaden Blignaut (600750)
- Oarabile Mbewe (600239)

Problem Statement:

BrightPath Academy places great emphasis on both academic performance and extracurricular activities. Substantial data is collected both on curricular and extracurricular participation. Unfortunately, no standardised platform exists to integrate all this data for the purpose of producing impactful insights. This forms the greatest challenge, but it sums up other problems the institution faces. Untimely identification of at-risk students, the absence of targeted support strategies for struggling students, a vague understanding of the impact extracurricular activities have on overall performance, and an overwhelming reserve of student data with no utilisation strategy all serve as difficulties facing the team at BrightPath Academy.

This project seeks to address all these by introducing an algorithm that makes use of machine learning techniques to put the collected data from the student performance data CSV file to use. We aim to produce an algorithm that predicts academic risk and recommends individualised support strategies by producing timely insights to educators at BrightPath Academy. Using linear regression, XG boost, and random forest, we intend to compare the outcome of all three classification algorithms to identify and implement the most efficient and insightful algorithm. That model will then undergo deep learning to finally be deployed on Dash.

Hypothesis Generation:

	Hypothesis
Study Time and Academic Performance	Students who spend more hours studying every week are more likely to achieve higher Grade Class categories compared to those with lower study time.
The impact of Absence when it comes to grades	The higher number of absences are associated with lower GradeClass categories, this indicates a negative impact on academic performance.
Extracurricular activities and academic success	Participation in extracurricular activities positively correlates with higher GradeClass categories, this suggests that engagement in these activities supports academic performance.
Parental support and student outcomes	Higher levels of parental support are associated with better GradeClass categories, indicating that parental involvement is a significant factor in academic success.
Tutoring effectiveness	Students who receive tutoring are more likely to achieve higher GradeClass categories compared to those who do not, suggesting that tutoring is an effective for improving grades.

Getting the system ready and loading the data:

Global Environment Configuration:

To prepare our system for development, we installed the following essential software components in our global environment:

- **Visual Studio Code (VS Code):** A lightweight, open-source code editor that serves as our primary development environment. It offers robust support for Python and seamless integration with Git for version control.
- **Git:** A distributed version control system (DVCS) used to track all changes made to our codebase, ensuring efficient collaboration and code management throughout the project lifecycle.
- **Python:** Installing Python provides access to a wide range of libraries and packages essential for building machine learning models and developing web applications using the DASH framework.
- **VS Code Jupyter Extension:** This extension enables native support for Jupyter Notebooks within VS Code, allowing us to write and execute Python code alongside markdown documentation in an interactive and unified interface.

Importing Python Libraries:

We will utilize the following Python libraries to support data analysis, machine learning model development, and visualization:

- **Joblib:** A utility for efficiently saving and loading trained machine learning models, enabling model reuse without retraining.
- **Matplotlib:** A powerful visualization library that allows us to create a variety of static, interactive, and animated plots—including line graphs, bar charts, histograms, and scatter plots—to explore trends and patterns within the data.
- **NumPy:** A fundamental library for numerical computing in Python, providing support for working with arrays (such as vectors and matrices) and performing complex mathematical operations.
- **Pandas:** A versatile data manipulation library used for loading, cleaning, filtering, and transforming structured data, making it easier to prepare datasets for analysis and modelling.
- **Scikit-learn:** A comprehensive machine learning library that we will use to develop, train, and evaluate various models, including those for classification, regression, and clustering tasks.
- **Seaborn**: A visualization library built on top of Matplotlib which provides a high-level interface for creating attractive and informative statistical graphics.
- **Mord:** Enables the manipulation of ordinal regression problems**.**
- **Tensorflow:** Enables numerical computation and large-scale machine learning for the creation of deeplearning models.
- **Xgboost:** Enables the scalable implementation of gradient boosting for decision trees.
- **Skopt:** Enables the tuning of ML model hyperparameters using bayesian optimization.

Uploading data:

We will upload the data using the following method:

Step 1: We will create a dataframe variable called “df”.

- Step 2: We will then utilized pandas to read the data stored in the provided csv file “Student_performance_data” and store the data in the dataframe variable.
- CODE: df = pd.read_csv(“Student_performance_data”)

Understanding the data:

Initial Data Exploration:

We explored the given dataset "Student_Performance_Data.csv" to understand its structure and content using the following inbuilt functions:

- **head ():** First, we displayed the datasets records using the inbuilt head () function.
- **info ():** Next, we gathered information about the columns such as number of records, data type and if it can contain null values using the inbuilt info () function.
- **describe ():** Finally, we gathered information about the dataset's records such as count, mean, standard deviation, min and max values using the inbuilt describe () function.

Dataset Overview

Column	Data Type	Statistical Type	Description	Values	Review
StudentID	int64	Nominal (Categorical)	Unique identifier for each student	Unique integers	Unique identifier, insignificant for analysis
Age	int64	Discrete Ordinal	Age of students	15, 16, 17, 18	Age distribution, correlation with performance metrics
Gender	int64	Binary Categorical	Gender of students	0 (Male), 1 (Female)	Gender distribution, performance differences by gender
Ethnicity	int64	Nominal (Categorical)	Ethnicity of students	0 to 3	Demographic analysis, check for performance gaps
ParentalEducation	int64	Ordinal	Parental education level	0 to 4	Impact on student performance, socioeconomic indicator
StudyTimeWeekly	float64	Continuous	Hours of study per week	Positive real numbers	Distribution, correlation with GPA, optimal study time
Absences	int64	Discrete Count	Number of absences during the school year	0 to 30	Impact on performance, identify at-risk students
Tutoring	int64	Binary Categorical	Whether student receives tutoring	0 (No), 1 (Yes)	Effectiveness of tutoring on performance
ParentalSupport	int64	Ordinal	Parental support level	0 to 4	Impact on student outcomes, interaction with other factors
Extracurricular	int64	Binary Categorical	Student involvement in extracurricular activities	0 (No), 1 (Yes)	Impact on academic performance

Sports	int64	Binary Categorical	Student involvement in sports	0 (No), 1 (Yes)	Relationship with GPA, time management
Music	int64	Binary Categorical	Student involvement in music	0 (No), 1 (Yes)	Relationship with GPA, cognitive benefits
Volunteering	int64	Binary Categorical	Student involvement in volunteering	0 (No), 1 (Yes)	Impact on personal development and academics
GPA	float64	Continuous	GPA of the student	Typically 0.0-4.0	Key outcome variable, distribution analysis
GradeClass	float64	Ordinal	Grade class based on GPA	0 to 4	Alternative categorical outcome variable

Exploratory Data Analysis:

Perform Univariate Analysis:

We have performed a univariate analysis to better understand the distribution and characteristics of our target variable. Focusing primarily on the grade class attribute, this process involves both descriptive statistics and visualization techniques. Below are the steps we took:

- **Overview:** we made use of the info() function to inspect the dataset's internal structure. This involved checking for null values and data types.
- **Descriptions:** the describe() function was used to provide statistical values including count, mean, and quartiles. This allowed for a quick overview of central tendency and potential outliers.
- **Outlier detection:** we made use of boxplots to visually identify potential outliers and values that were not within the interquartile range. We supplemented this visualisation with programmed calculations to explicitly find and label outliers.
- **Distribution:** our second visualisation was a histogram to illustrate the distribution within the grade class attribute. This helped identify the frequency of grade class ranges.

This analysis provided a comprehensive overview of students' performance. It helped identify skewness in our data and highlighted the most populous grade class.

Perform Bivariate Analysis:

Continuous Variables vs. Ordinal Target (GradeClass):

The relationship between continuous columns (StudyTimeWeekly and Absences) and the target variable "GradeClass":

- **Visualizing with boxplots**:
  These plots visualized that the distribution of study time and absences varied across the different grade categories. For example, students with higher grades (like A or B) tend to study more or have fewer absences.

- **Statistical Correlation (Spearman's rank correlation)**:
  Spearman's correlation is ideal for measuring relationships between ranked or ordered variables. This analysis quantified the strength and direction of the relationship, while the p-value indicated whether the correlation was statistically significant.

Binary Variables vs. Ordinal Target (GradeClass):

The relationship between the binary columns (Tutoring, Extracurricular, Sports, Music, Volunteering) and the target variable "GradeClass":

- **Visualized with boxplots**:
  Boxplots allowed us to compare grade performance between students who did and did not participate in specific activities. For instance, students in extracurricular activities may show a different distribution of grades than those who are not.
- **Point-biserial correlation**:
  Since the variables are binary (yes/no), and our target variable is ordinal, we calculated the point-biserial correlation to measure how much the presence or absence of each activity is associated with grade performance.

Ordinal Categorical Variable vs. Ordinal Target:

The relationship between Parental Support (ordinal) and GradeClass:

- **Creating a heatmap of a cross-tabulation**:
  Visualizes how grade categories are distributed across different levels of parental support. It visually highlighted any trends, such as whether higher parental support corresponded to better grades.
- **Spearman's correlation**:
  We used the spearman's rank correlation here to quantify the strength of the monotonic relationship between the levels of parental support and the students' grade class.

Missing value and outlier treatment:

Checking for Data Quality Issues:

We have also checked if our dataset contains any missing values or duplicate entries using the duplicate () and isnull() functions. This cleans our data and ensures it remains consistent bu avoiding error or bias in our ML models.

Outlier Detection and Removal:

We used the Interquartile Range (IQR) method to detect and remove any outliers that are higher or lower than most data points in our dataset.

Evaluation Metrics for classification problem:

We have utilized the following two Evaluation Metrics for our classification problem:

Classification report:

The classification report serves as a detailed gradebook which summarizes the metrices that evaluates the performance of the ML models on each class. It summarizes the following matrices:

- **Precision:** Determines the total number of times the ML model predicted the actual class.
- **Recall:** Determines the number of times the ML model could find a class to use for predictions
- **F1 Score:** Calculates a balanced score from using precision and recall.
- **Accuracy:** Provides an overall score on the performance of the ML model.

The classification report helps by showing which grades the ML model struggles with and how reliable its predictions are.

Confusion Matrix:

The confusion matrix works like a performance scoreboard that visually evaluates the performance of our ML models. It displays how many predictions our ML models got right, got wrong and where it started to get confused. This helps us spot patterns in the mistakes that our ML models made and see how close those mistakes were.

Feature engineering:

Target Variable Transformation (Creating Grade Categories):

For the target column "GradeClass" to have more accurate records we mapped the values in the "GPA" column to the target column where we converted the mapped data into ordinal data. We used a function which transformed the data into a new categorical variable called "GradeClass". This function grouped students into five categories based on their GPA:

- **A (Excellent)**: GPA of 3.5 and above
- **B (Good)**: GPA between 3.0 and 3.49
- **C (Average)**: GPA between 2.5 and 2.99
- **D (Below Average)**: GPA between 2.0 and 2.49
- **F (Failing)**: GPA below 2.0

Removing Irrelevant or Redundant Features:

We removed all the columns that did not influence the target variable "GradeClass" such as identifiers (like Student ID), demographic data (like age, gender, and ethnicity). We also removed the "GPA" columns because it can also be perceived as output data which can't be used for training our ML models. These changes simplify our dataset and focuses on the important features.

Model Building:

Logistic Regression:

Logistic regression is a statistical method used for binary classification. It predicts the probability that a given inputs belongs to a particular class this can 0 or 1. Unlike linear regression, which predicts continuous values, logistic regression uses the logistic(sigmoid) function to map predictions to probabilities between 1 and 0.

Ordinal Logistic Regression

The target is GradeClass has 5 ordered categories. Standard logistic regression is for binary outcomes, but for ordinal logistic regression extends it to ordinal data. With the use of cumulative probabilities this will predict in which ordinal class to go into based on the data.

The model introduces thresholds between cumulative probabilities. The model predicts the class by finding the first cumulative probability that exceeds a threshold or by comparing the linear predictor to the thresholds.

LogistacAt from the mord library, which implements ordinal logistic regression using the All-Thresholds method, LogistacAt fits a single set of coefficients and thresholds to model the ordinal relationship between features and GradeClass.

Step by step Process:

- **Step 1**:

  Logistic regression is sensitive to the scale of featured because it optimizes coefficients to minimize the loss function. Features with larger ranges could dominate smaller ones. SdandardScaler standardizes StudyTimeWeekly and Absences to have a mean of 0 and a standard deviation of 1.

  Parental Support is treated as an ordinal variable. OrdinalEncoder ensured its treated as a single numerical feature, preserving its ordered nature. This is a appropriate for ordinal logistic regression which could interpret the numerical values as ordered levels

  Categorical features are binary, so we use 'passthrough' to keep them as is. Alternatively, we could one hot encode them but since they are binary this is not necessary.

  Logistic regression assumes a linear relationship between the features and the log-odds. Proper scaling and encoding ensures that the model can fairly weigh each features contribution.

- **Step 2**:

  (Stratify = y) ensures that the proportions of each gradeclass is the same in the training and testing data sets. This is crucial for ordinal regression because imbalance classes can bias the model.

  The train test split, 80% of the data is used for training and 20% for testing which is a standard split for evaluating model performance.

- **Step 3**:

  GradeClass likely has an imbalanced distribution. This can cause the model to favour majority classes leading to poor predictions for minority classes.

  SMOTE generates synthetic samples for minority classes by interpolating between existing samples this balances the training set, helping the model to learn better decision boundaries for all classes

Logistical regression optimizes a loss function that can be skewed by imbalance of data. Balancing the classes ensured that the model doesn't overfit to the majority class

- **Step 4**:

  LogisticAT() initializes the ordinal logistic regression model. It assumes the proportional odds model, meaning the effect of features is the same across all thresholds.

  The alpha parameter controls L1 regularization (Lasso penalty). A higher alpha increases regularization, shrinking coefficients toward zero, which can prevent overfitting and perform feature selection by setting some coefficients to exactly zero grid_search.best_estimator_ gives the LogisticAT model with the optimal alpha, which minimizes the MAE on the cross-validated training set.

- **Step 5**:

  LogisticAT minimizes a loss function that combines the log-likelihood (how well the model fits the data) with the L1 penalty (controlled by alpha). The log-likelihood for ordinal logistic regression is based on the cumulative probabilities

  The model assumes that the relationship between the features and the log-odds of being in a higher class is the same across all thresholds. This means a single set of coefficients is learned, but multiple thresholds are used to separate classes

- **Step 6**

  Provides precision, recall, and F1-score for each class (0 to 4).
  Precision: Proportion of correct predictions for a given class (e.g., GradeClass 2 predicted correctly).
  Recall: Proportion of actual instances of a class that were correctly predicted.
  F1-score: Harmonic mean of precision and recall, useful for imbalanced data.
  This treats the problem as a multi-class classification task, which is standard but doesn't fully capture the ordinal nature of the target.

  Confusion Matrix:

  The confusion matrix shows the number of predictions for each actual vs. predicted class pair.

  Normalizing by row (cm.sum(axis=1)) converts counts to proportions, making it easier to see the model's performance for each class.

  The plot visualizes where the model is making errors (e.g., if it often predicts GradeClass 3 instead of 2).

- **Step 7**

  The coefficients represent the change in the log-odds of being in a higher GradeClass for a one unit increase in the feature, holding other features constant.

  A positive coefficient means that increasing the feature value increases the likelihood of being in a higher GradeClass. For example, a positive coefficient for Absences suggests that more absences lead to a higher GradeClass.

A negative coefficient means the opposite: increasing the feature decreases the likelihood of being in a higher GradeClass. For example, a negative coefficient for StudyTimeWeekly suggests that more study time leads to a lower GradeClass.

Random Forest:

This is a supervised machine learning algorithm which we will use for the classification of students regarding our target attribute, grade class. We tested for accuracy and applied measures to test the effect on accuracy of both changing the number of trees and the number of attributes. We train, test, and rebuild the model to find the highest achievable accuracy using the steps below.

- **Preparation**: this is where we import our dataset and get a quick glance over it. We also pick out our target attribute as it will be isolated from the other attributes as seen in the steps to follow
- **Training, testing, and fitting**: this is where we introduced the random forest classifier function. We split the data into thirds with the training set taking up two thirds and the test set only taking a third of our data. We have also made use of a 42 random state to randomise the executions of the model in each iteration of the algorithm. This allows for maximum achievable accuracy with lowered chance of the model underperforming. Finally, we fit the model to the training set to start the categorisation.
- **Evaluation**: we have made use of confusion matrix and classification report to assess the model's accuracy and performance, all with precision, recall f-1 score and others all bundled in.

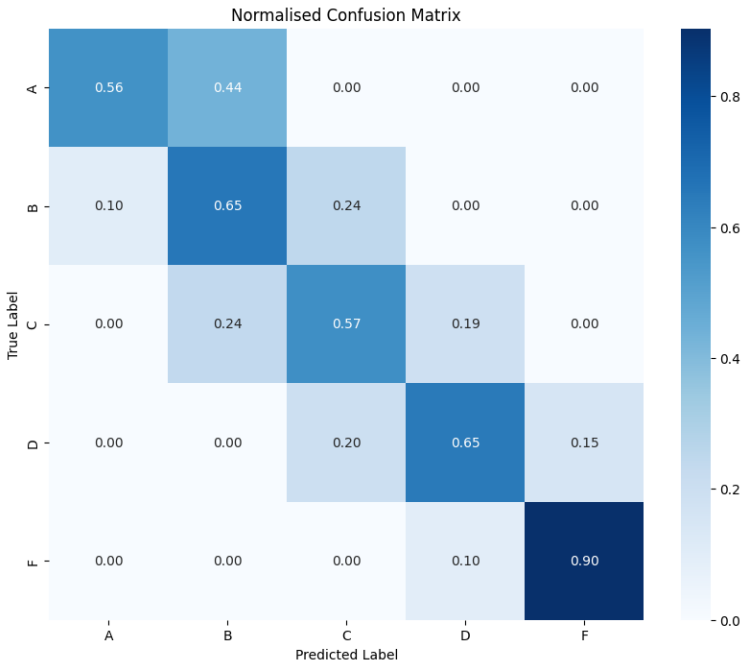
This is all because the model handles categorical targets well. It excels at identifying the most influential attribute all while greatly reducing overfitting due to its ensemble and averaging nature. All while handling various data types well.

XGBoost:

XGBoost is a performant and flexible gradient boosting implementation, which can be utilised for various tasks ranging from tabular data to large-scale datasets. It allows for faster iteration and experimentation with more features and hyperparameters.

To setup XGBoost for multi-class classifications problems, such as the one expressed in this project, we use the multi:softprob objective, which outputs a vector of class probabilities. Additionally, we configure our num_class parameter to specify the number of classes in our target variable (which is 5). This dataset also required assigning sample weights to adjust for the imbalanced data, by setting the sample_weight parameter when training to balance the importance of underrepresented classes.

Model parameters are then further tuned with Bayesian optimisation within the defined search space to determine the optimal parameters, which ultimately yields an accuracy of 77%. This normalised confusion matrix provides a visual overview of the model's performance in predicting different classes.



Deep Learning classification algorithm:

- **Chosen classification algorithm:**

An **A**rtificial **N**eural **N**etwork (ANN) deep learning model that utilizes the **C**umulative **O**rdinal **R**egression with **L**ogistics (CORAL) technique.

- **ANN algorithm with CORAL:**

This guided project provided us with a dataset that has the following characteristics:

- Contains an ordinal target variable called: “GradeClass”, which represents a student’s performance on a scale of meaningful order.
- Contains a mix of Continues(numeric) and Categorical (Binary\Ordinal) variables.

The ANN model allows us to take multiple inputs and predict how well they’re likely to perform in terms of the ‘GradeClass” or ordinal values. This model focuses on the natural ranking between the “GradeClass” values because it can capture **non-linear patterns and interactions** between variables more effectively than simpler models. With the help of the CORAL technique the ANN model can be guided to treat predictions as ordered rather than flat categories by transforming the ordinal target variable values into a series of **binary threshold values.** This leads to more realistic and informed predictions.

- **Step-by-Step Explanation:**
  - **Step 1: Preparing the Data:**

    We prepared the data by uploading the “Cleaned_Student_Performance_Data.csv” and storing the data in a data frame variable.

- **Step 2: Splitting and Scaling:**

  In this step we split the data frame into 80% training data and 20% testing data. All the independent variables are stored in the x_train and x_test variables which are then scaled to be in the same range of values. The target variables are stored in y_train and y_test variables.

  Then we covert the ordinal data stored in y-train into a binary format using the CORAL method which allows the ANN model to make better predictions.

- **Step 3: Feeding the Data into a Neural Network:**

  In this step we build and compile the ANN model. We initialize the model using the TensorFlow library which is then followed by the input, hidden and output layers of the ANN model which is given a number of units/nodes to train the data and the 'RELU' activation quality that introduces non-linearity which means that if x is positive it will return x but if it is negative it will return 0.

- **Step 4: Output as a Set of Binary Decisions:**

  Now that the has been build we then fit it with the x_train data and the binary converted y_train data in order to train the ANN model. We then use the trained ANN model to predict the correct target variable in binary format by fitting the model with the x_test data.

- **Step 5: Converting to Final Grade Prediction:**

- In this step we just created a function that coverts the predicted binary outputs into ordinal data.

- **Step 6: Evaluating the Predictions:**

- We then evaluated the performance of the model using a confusion matrix and a classification report.

Model deployment:

For this project, we employed the use of the Plotly Dash web framework and deployed to the Render platform. The Dash framework uses python directly which allows for straightforward interoperability with the existing model building and training codebase. We determined the web app would consist of two pages, namely the home and analysis page.

- **Home Page:** the landing page of our web app, which also serves as our primary outcome: a tool for evaluating the potential academic performance of a student based on the information provided.

- **Analysis Page:** visually explanatory page in which we highlight some of the major aspects of the data processing process and model building/evaluation.

We created a GitHub repository to house all the major aspects of the project. Including the web app, data processing and model building notebooks, and our refined datasets. Utilising the Render platform, then allows for seamless integration with Git providers and the created repository. Any new

changes to the repository automatically trigger a new deployment and uploads the latest build to the Render services.

Although Render deployments can work without configuration, it does require appropriately setting up a python virtual environment to work with dependencies specific to our web app. To setup a virtual environment, we used the rye and poetry python toolchains, respectively to manage our python environment and dependencies. Any version mismatches were configured through the Render environment variables to ensure deployments use the appropriate versions.

Finally, after determining the most effective model for our solution, to deploy with our web app, we exported the model to a H5 file, which can then be loaded on demand to make predictions on new data.