Guided Project MLG382: Student Performance Prediction

Abstract:
BrightPath Academy is a high school that has a commitment to academic excellence. BrightPath Academy faces various challenges with their progressive approach; this project serves as a solution to predict students' academic performance by utilising machine learning.

Scan for GitHub Repository

Scan for webapp

2025-04-22

Team Members:
Markus Reblin 578083
Ulrigh Oosthuizen 577952
Vutivi Maswanganyi 577800
Jonathan Joubert 578085

Contents

1. Problem Statement

This system aims to provide real-time data driven insights and information that will empower lecturers to delivery personalised interventions to improve individual academic performance.

Problem Statement:
Will we be able to develop a program that organises students into their respective ‘GradeClass’ based on data that includes students’ demographics details, study habits, parental involvement, etc.

2. Hypothesis generation

Demographic:

H1 – Students with higher parental education have better academic performance.
Parents with higher education levels are better equipped to support the students’ learning habits. They may influence the students’ grades by engaging with students by explaining homework, assignments, etc. (Dubow, Boxer and Huesmann, 2009)

H2 – Gender may influence academic performance.
Traditional gender expectations encourage females to be more diligent and compliant with school norms and expectations. Males tend to be more affected by peer pressure that de-emphasises academic achievement in adolescence. Given that females are encouraged to be more organised and diligent regarding schoolwork, this positively impacts female academic performance. (Duckworth and Seligman, 2006)

H3 – Ethnicity may influence academic performance.
In many South and East Asian cultural backgrounds strong emphasis is placed on educational achievement through forms of upholding family reputation and family honour. (Goyette and Xie, 1999) These educational attainments lead to higher levels of academic motivation which lead to increased academic performance (Sue and Okazaki, 1990)

Study habits and parental involvement:

H4 – Students with higher weekly study times have better academic performance.
Better academic performance is often a direct result of higher weekly study times. Students that study more often engage with the study material and display more effort than students who study less. This is a direct correlation with GPA. (Pei, 2024)

H5 – Students with fewer absences have better academic performance.
Students who attend lectures and lessons more often continually learn and tend not to fall behind in study material opposed to students who are frequently absent. These factors improve academic performance. (Pei, 2024)

H6 – Students who receive parental support have better academic performance.
Students with higher levels of parental involvement receive academic assistance such as explanations of study material, reminders to study, motivation and inspiration, etc. This foundation of support increase chances of academic success. (The Annie E. Casey Foundation, 2022)

H7 – Students that receive tutoring have better academic performance.
Students who receive tutoring receive specific academic assistance. Tutors assist students with their weak areas in their study material. This individual assistance addresses students’ knowledge gaps by

reinforcing course concepts which improves overall academic performance. (Nickow, Oreopoulos and Quan, 2020)

Extracurricular activities:

H8 – Students who participate in sports and/or music have better academic performance.
Student involvement in creative and physical extracurriculars enhance discipline and time management. Studies link these activities to improved academic performance. (Furda and Shuleski, 2019)

H9 – Students who volunteer have better academic performance.
Students that volunteer have tend to have a sense of purpose and increased responsibility which are important factors that positively impact academic performance. (Astin et al., 2000)

Initial data exploration

Before starting with the analysis, the data set was loaded using pandas to read the csv file. Thereafter, the first 10 records of the dataset were printed out to check that the csv file was successfully linked. After verifying that the CSV file could be read, a missing value check was run and revealed that the dataset was clean and contained no null values. The dataset we were working with had over 2300 student records with 15 columns for each record. Each row contained individual student attributes like demographics, study habits, parental involvement, and academic performance. These initial data exploration checks confirmed that the dataset is complete and was ready for further exploration and preprocessing.

Dataset description:

Feature	Description
StudentID	Unique identifier for each student
Age	Student age which ranged from 15-18 years old
Gender	0 is the identifier for male, 1 is the identifier for female
Ethnicity	0=Caucasian, 1=African American,2=Asian,3=Other
ParentalEducation	Education level of the student's parents ranging from none to higher studies (0-4)
StudyTimeWeekly	Hours spent studying per week
Absences	Number of school absences
Tutoring	0=no tutoring,1=participates in tutoring
ParentalSupport	Level of the student's parents support in their academics ranging from none to very high (0-4)
Extracurricular	0=no participation,1=participates in extracurricular activities
Sports	0=no participation,1=participates in sports
Music	0=no participation,1=participates in music
Volunteering	0=no volunteering,1=volunteers
GPA	Grade point average (2.0-4.0)
GradeClass	Classified grade (0=A,1=B,2=C,3=D,4=F)

Student Performance Data Analysis:

1. Handling Missing Data

The dataset was initially checked for missing values. All missing numeric values were replaced with the **median** of their respective columns.
This method was chosen to avoid the influence of outliers while maintaining data integrity.
After imputation, the dataset had **no remaining missing values**.

2. Univariate Analysis

Univariate analysis was conducted to understand the individual behavior of variables.

Numerical Variables:

- Age
- StudyTimeWeekly
- Absences
- GPA
- GradeClass

Each numerical variable was summarized using descriptive statistics and visualized using **histograms with KDE plots**.

Categorical Variables:

- Gender
- Ethnicity
- ParentalEducation
- Tutoring
- ParentalSupport
- Extracurricular
- Sports
- Music
- Volunteering

Value counts and **count plots** were used to visualize the frequency of each category.

3. Outlier Detection and Treatment

Z-score analysis was performed on numerical features to identify outliers.
Data points with Z-scores above +3 or below –3 were considered outliers.
These rows were removed from the dataset to improve the quality of analysis and ensure more accurate modeling.

4. Bivariate Analysis

Bivariate analysis was conducted to examine **relationships between variables**.

Correlation Analysis

A **correlation heatmap** was generated for numerical variables to identify linear relationships. For example, a positive correlation was observed between **StudyTimeWeekly** and **GPA**.

Categorical vs Numerical

Boxplots were used to compare **GPA** across different categories such as:

- Gender
- ParentalSupport
- Tutoring

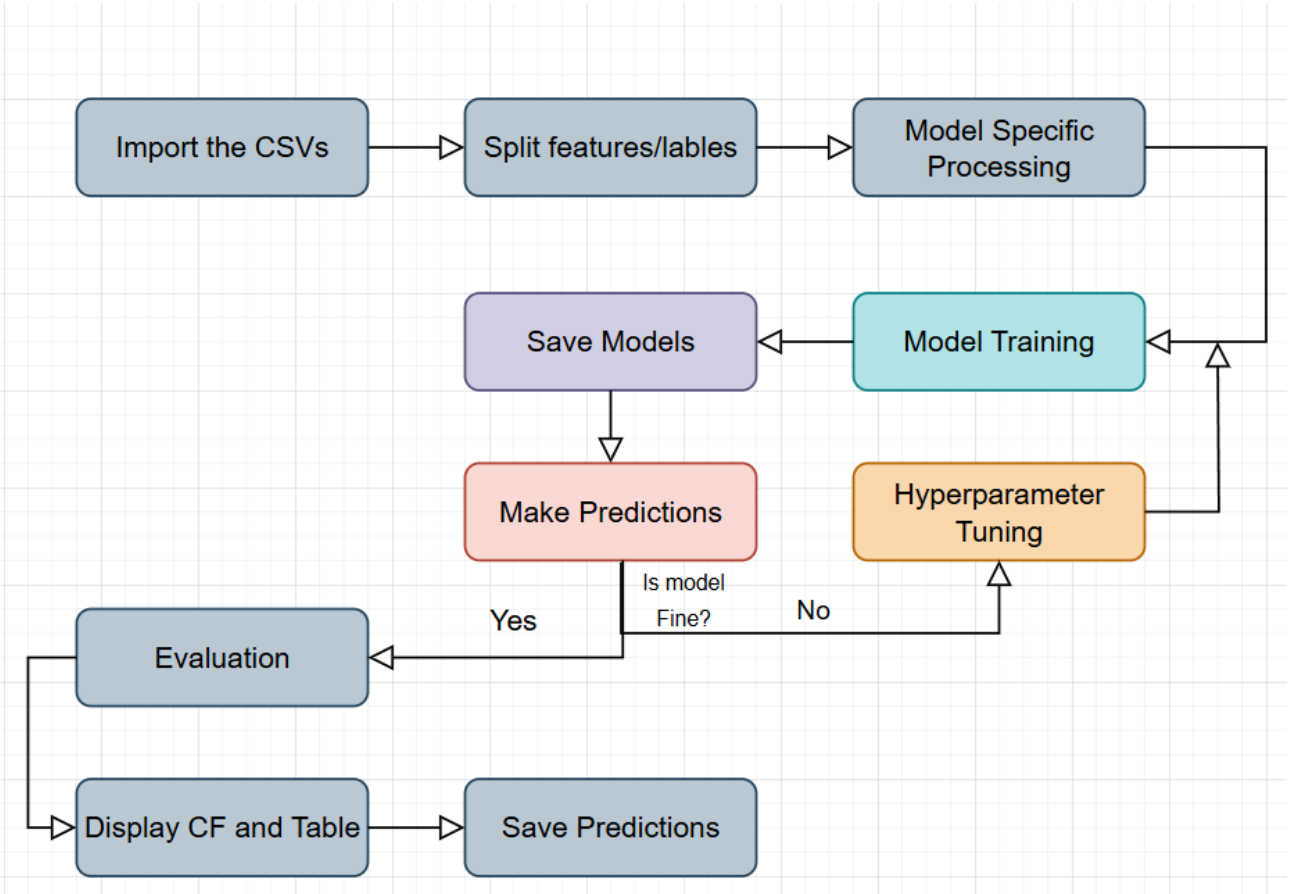
Numerical vs Numerical

Scatter plots were used to visualize the relationships between numerical variables and **GPA** (e.g., GPA vs Absences).

Models

Logistic Regression, Random Forest, XGBoost

This is a basic visual overview of how each model executes:



Logistic Regression

Step 1: Loading the Datasets

Here, two datasets are loaded from CSV files using pandas.read_csv(). The df_train file is used to train the model, while df_test is used to evaluate its performance. The delimiter is set to a comma to correctly parse the CSV structure.

Step 2: Split Features and Target

Unnecessary columns such as "StudentID" is dropped as well as "GPA" from the input features, as they are not needed, or shouldn't be used for prediction. The target variable is "GradeClass", which is stored separately in y_train and y_test.

Step 3: Feature Scaling

Feature scaling is performed using StandardScaler to normalize the data. This ensures that all features contribute equally to the model's learning process. The scaler is fit on the training data and then applied to both training and testing data.

Step 4: Hyperparameter Tuning with GridSearchCV

GridSearchCV is used to find the best hyperparameters for the logistic regression model. It performs a 5-fold cross-validation over a grid of parameter values for regularization strength C and max_iter. The best model is selected based on accuracy, and the optimal parameters are printed.

Step 5: Save the Trained Model and Scaler

The trained model and scaler are saved as .pkl files using pickle. This allows the model to be loaded and used later without needing to retrain it.

Step 6: Make Predictions

Once the best logistic regression model is found, it is used to make predictions on the test dataset. The predictions represent the predicted GradeClass labels.

Step 7: Model Evaluation

The model's performance is evaluated using the accuracy score and classification report. The classification report provides precision, recall, and F1-score for each class, giving detailed insight into how well each grade class is predicted.

The model is then checked to see if it worked well or not, adjusted then steps 4 to 6 is done again. Keep note that the model also does this automatically when executing to find the best mdoel based on the params provided.

Step 8: Display and Save Results

This section displays the first 20 predictions in a styled DataFrame. Incorrect predictions are highlighted in red for quick visual analysis. This helps identify patterns in misclassifications. The

confusion matrix is visualized using a heatmap, making it easier to understand how many predictions were correct or incorrect for each class. The diagonal values represent correct classifications.

Step 9: Export Predictions

Finally, the comparison DataFrame is exported as a CSV file. This provides a record of the model's predictions that can be reviewed or shared for further analysis.

Step 10: Predicting for a Single Student

This function takes a single student's raw data (as a CSV-formatted string), preprocesses it (removing ID, GPA, and grade), scales it using the saved scaler, and predicts their grade using the trained model.

Random Forest

Step 1: Load the Datasets

The training and testing datasets are loaded using pandas. The training set (df_train) is used to train the model, and the test set (df_test) is used to evaluate its performance.

Step 2: Split features and Targets

Unnecessary columns such as "StudentID" is dropped as well as "GPA" from the input features, as they are not needed, or shouldn’t be used for prediction. The target variable is "GradeClass", which is stored separately in y_train and y_test.

Step 3: Hyperparameter tuning with GridSearchCV

To optimize the Random Forest model, GridSearchCV is used to test combinations of hyperparameters such as n_estimators, max_depth, and criterion. This cross-validation approach selects the best parameters based on accuracy. The verbose=1 parameter shows the progress of the search.

Step 4: Model Selection

The best performing model found from the grid search is selected and used to generate predictions on the test dataset.

Step 5: Save the Trained Model

The trained Random Forest model is serialized and saved using pickle. This allows the model to be reused later without retraining.

Step 6: Make Predictions

Once the best Random Forest model is found, it is used to make predictions on the test dataset. The predictions represent the predicted GradeClass labels.

Step 7: Model evaluation

The model's performance is evaluated using three key metrics:

- Best Parameters: Displays the best hyperparameter combination.
- Accuracy: Proportion of correct predictions over total predictions.
- Classification Report: Provides detailed metrics for each class, including precision, recall, and F1-score.

Step 8: Display and Save Results

This section creates a styled output showing the first 20 predictions, with incorrect ones highlighted in red. This allows for quick visual analysis of where the model might be making mistakes. A heatmap is used to visualize the confusion matrix. This matrix highlights how often classes were correctly or incorrectly predicted, with darker cells indicating higher counts.

Step 9: Save Predictions

The final comparison DataFrame, containing actual and predicted grade classifications, is saved to a CSV file for documentation or further analysis.

Step 10: Predicting for a Single Student

A prediction function is defined that takes a raw CSV-like input string, preprocesses it, and uses the trained Random Forest model to predict the student's grade.

XGBoost

Step 1: Load the Datasets

Training and testing data are loaded from CSV files. The train_data is used to train the XGBoost model, while test_data is reserved for evaluating its performance.

Step 2: Separate Features and Target

Unnecessary columns such as "StudentID" is dropped as well as "GPA" from the input features, as they are not needed, or shouldn't be used for prediction. The target variable is "GradeClass", which is stored separately in y_train and y_test.

Step 3: Build the pipeline

A scikit-learn Pipeline is used to encapsulate the XGBoost model. This modular approach makes it easy to integrate preprocessing steps in the future and streamlines hyperparameter optimization.

Step 4: Set up Bayesian Hyperparameter Search

Bayesian Optimization is set up using BayesSearchCV from the scikit-optimize library. A range of hyperparameters for XGBoost (such as learning rate, max depth, regularization terms, and column subsampling) are defined to search the best combination efficiently.

Step 5: Train the Model

The model is trained using BayesSearchCV, performing 3-fold cross-validation over 10 iterations. This helps to efficiently identify the best-performing hyperparameter configuration based on accuracy.

Step 6: Save the Trained Model

The best XGBoost model found by BayesSearchCV is saved to a .pkl file, ensuring that it can be reused for inference or deployment without retraining.

Step 7: Make Prediction

After the optimal model is found, it is used to predict the grade class of the test dataset.

Step 8: Model evaluation

The accuracy and classification report of the XGBoost model are generated. Metrics include precision, recall, and F1-score for each grade class. The zero_division=0 flag avoids errors for any class with zero predictions.

Step 9: Display and Save Results

This section displays the first 20 predictions, using custom styling to highlight incorrect predictions in red. It provides an intuitive view of model accuracy immediately. The confusion matrix is plotted to visualize model performance across all grade classes. It highlights correct predictions on the diagonal and misclassifications elsewhere.

Step 10: Save Predictions

A copy of the test data is created with added columns for actual and predicted grade classes, preparing the dataset for export or visualization. The comparison DataFrame is saved as a CSV file, allowing the results to be archived, shared, or further analysed outside the notebook environment.

Step 11: Predicting for a Single Student

This function preprocesses a CSV-style student input, predicts their grade using the trained XGBoost model, and maps the numeric label back to a letter grade.

Deep Learning Model

Data is split into training and testing data, where 80% of the data is used for training and the other 20% is used for testing. A fixed random state variable is set to ensure reproducibility of results. Since multi-class classification is involved, we need to convert the target variable into one-hot encoded vectors using the “to categorical” function. This is needed to meet the format expected from the softmax activation function in the final layer of the neural network.
The model is defined using Keras’ Sequential API, which enables the creation of a linear stack of layers. The model consists of three connected (Dense) layers, with the first layer containing 128 neurons and using the ReLU activation function. The second layer contains 64 neurons and also uses the ReLU function. Lastly the third layer only contains 5 neurons, which corresponds to the number of target classes.

The model is compiled using the Adam Optimizer, which is an adaptive learning rate optimization algorithm that shows high efficiency and performance when training deep neural networks. The loss function is set to “categorical crossentropy”, which is suited for tasks where the target variable is one-hot encoded. Lastly, the model is configured to evaluate its performance using accuracy.

The compiled model is trained using the training data for 50 epochs and with a batch size of 32. This means that the model processes 32 samples at a time before updating its weight. The validation split indicates that 20% of the training data is further set aside internally for validation during training, which helps monitor the model’s performance and detect overfitting. The validation parameter provides an explicit validation set using the separate test data.

The evaluate() function calculates the model’s loss and accuracy on the test set. The model then generates predictions using the predict() function, which outputs probability distributions for each class. To convert these into the actual class labels, the np.argmax() function is used to select the index of the highest probability for each class. The classification report() function provides a comprehensive summary of performance metrics including precision, recall and F1-score for each class, along with macro and weighted averages. Lasty, the confusion matrix() function outputs a matrix that visualizes the number of correct and incorrect predictions for each class. The model achieved an overall test accuracy of 72%.

Evaluation Results:

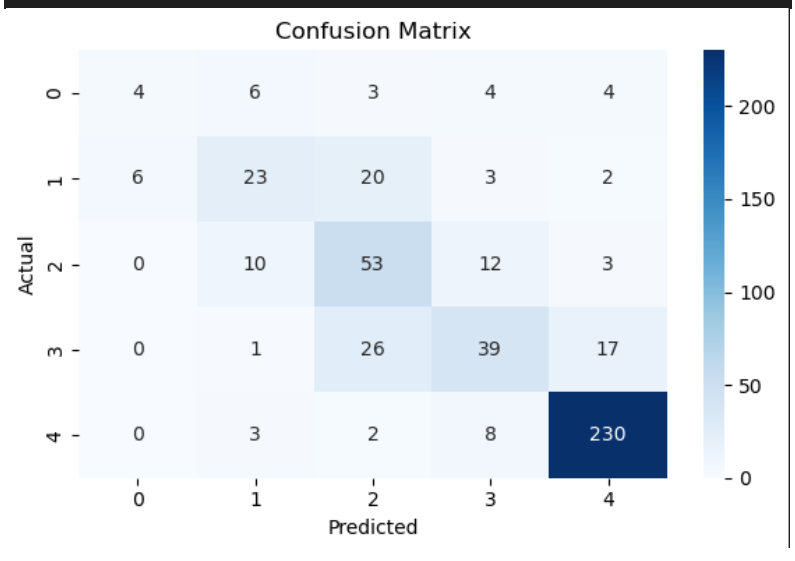
Logistic Regression

```
Accuracy: 0.7286012526096033

Classification Report:
              precision    recall  f1-score   support

           A       0.40      0.19      0.26        21
           B       0.53      0.43      0.47        54
           C       0.51      0.68      0.58        78
           D       0.59      0.47      0.52        83
           F       0.90      0.95      0.92       243

    accuracy                           0.73       479
   macro avg       0.59      0.54      0.55       479
weighted avg       0.72      0.73      0.72       479
```



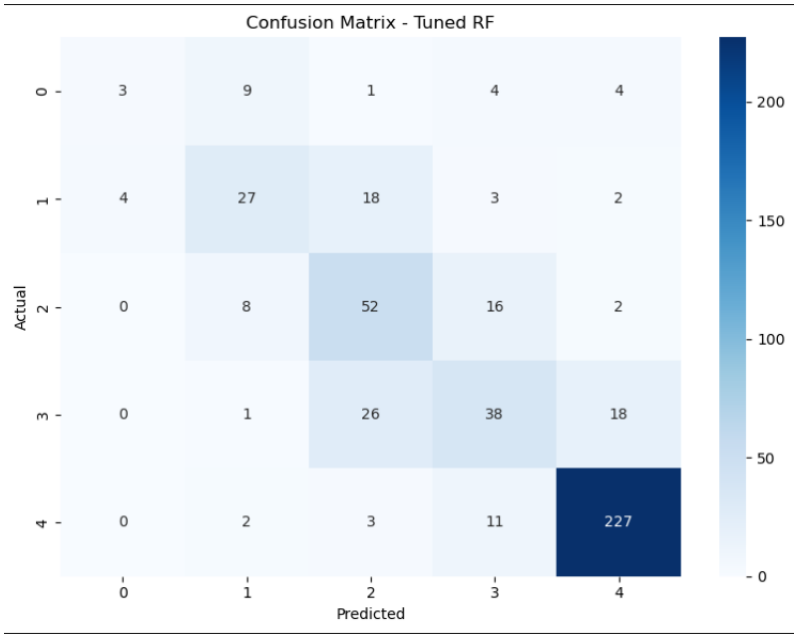
Random Forest

```
Accuracy: 0.7244258872651357

Classification Report:
              precision    recall  f1-score   support

           A       0.43      0.14      0.21        21
           B       0.57      0.50      0.53        54
           C       0.52      0.67      0.58        78
           D       0.53      0.46      0.49        83
           F       0.90      0.93      0.92       243

    accuracy                           0.72       479
   macro avg       0.59      0.54      0.55       479
weighted avg       0.71      0.72      0.71       479
```

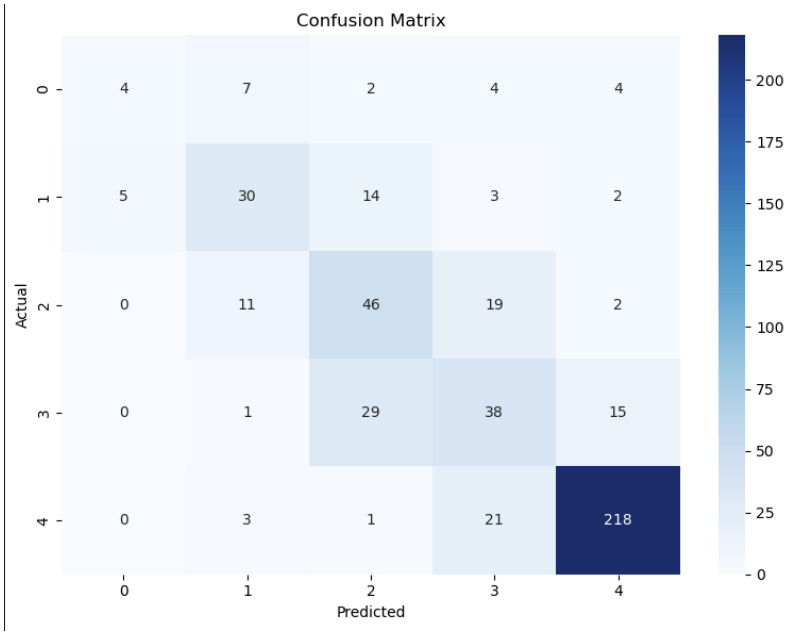


XGBoost

```
XGBoost Accuracy: 0.7014613778705637
Classification Report:
              precision    recall  f1-score   support

           A       0.44      0.19      0.27        21
           B       0.58      0.56      0.57        54
           C       0.50      0.59      0.54        78
           D       0.45      0.46      0.45        83
           F       0.90      0.90      0.90       243

    accuracy                           0.70       479
   macro avg       0.57      0.54      0.55       479
weighted avg       0.70      0.70      0.70       479
```



Deep Learning Model

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7251 - loss: 0.9590
Test Accuracy: 0.72
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
              precision    recall  f1-score   support

         0.0       0.73      0.36      0.48        22
         1.0       0.58      0.61      0.59        49
         2.0       0.60      0.54      0.57        85
         3.0       0.53      0.50      0.51        86
         4.0       0.85      0.92      0.88       237

    accuracy                           0.72       479
   macro avg       0.66      0.59      0.61       479
weighted avg       0.71      0.72      0.71       479

[[  8   5   5   2   2]
 [  2  30  13   0   4]
 [  0  13  46  20   6]
 [  0   3  13  43  27]
 [  1   1   0  16 219]]
```

Expected vs. Obtained Results

During testing, it was observed that while Random Forest was not the most accurate model according to overall metrics, but it produced the most consistent predictions in line with expected outcomes when evaluated on specific inputs. In contrast, both Logistic Regression and XGBoost occasionally misclassified inputs, typically predicting a grade either one level above or below the expected class.

An example for this was with the following inputs across all 3 models:

Example Input:
“1015,18,1,0,1,11.197810636915708,9,1,2,0,0,0,0,2.396788117124796,3.0”

Expected Grade: D

Predicted Results:

Model	Predicted Grade
Deep Learning	D
Logistic Regression	C
Random Forest	D
XGBoost	C

This demonstrates that although Logistic Regression and XGBoost had strong performance in general accuracy, Random Forest and the Deep Learning models may be more reliable in edge cases or scenarios where accurate identification of at-risk students is critical.

Web Application Development

Dash, a Python framework for building was used for the development of the web application. Components from dash-core-components and dash-bootstrap-components were used to provide a clean, responsive user interface. The application is used as a student performance prediction tool and uses multiple machine learning models to predict a final grade based on user input.

User Interface and Features

The app features a single-page interface organized using a Bootstrap container and responsive row-column layout. Key UI elements include:

- Numeric inputs: For Age, Study Time Weekly, and Absences.
- Dropdown Menus: For selecting categorical features like Gender, Ethnicity, Parental Education, and Parental Support.
- Checklists: For binary activities like Tutoring, Extracurricular, Sports, Music, and Volunteering.
- Predict Button: Triggers model prediction.
- Prediction Output Area: Displays results from all four models in a table.

Input Validation

Input validation is implemented to ensure:

- Age falls within 15-20
- Study Time Weekly falls within 0-80 hours
- Absences is not negative

Model Integration and Prediction Logic

A Pandas dataframe is constructed with the given input values after validation is done. Numeric fields are then standardized using a pre-fitted StandardScaler. Predictions are then made using the four loaded models. The predicted grade is then mapped from numeric form to letter grades (A to F) using a custom mapping dictionary.

Local Deployment and Testing

The application was deployed locally for deployment and testing using the built-in Dash server. To prevent GPU conflicts, Tensorflow was restricted to CPU execution.

Deployment to Render

To make the student grade prediction application publicly available, the project was deployed using Render. Render was selected due to its ease of integration with GitHub, support for Python environments, and compatibility with Dash applications.

Deployment Steps:

1. **Project Structure Organization:**

   The project was organized into a clean directory structure to separate the source code, models, and data files.

2. **Preparing the Requirements File:**

   All dependencies needed by the application were listed in a requirements.txt file. This ensures that Render could recreate the environment properly during deployment. Key libraries included:
   - Dash
   - Dahs-boostrap-components
   - Pandas
   - Numpy
   - Scikit-learn
   - Tensorflow

3. **Creating the Web Service on Render:**

   A new Web Service was created in Render with the GitHub repository containing the application connected. Python was then selected as the runtime environment. Configuration settings were set up as follows:
   - Buid Command: pip install -r requirements.txt
   - Start Command: gunicorn source.app:server --timeout 120

4. **Handling File Paths and Model Loading:**

   To ensure that models and artifacts loaded correctly in both local and cloud environments, all file paths were handled dynamically using Python's os.path methods. Lazy loading was implemented when loading the ML models and artifacts to optimize the performance and efficiency of the application. Lazy loading delays the initialization of an object until it is actually needed. This was achieved by using the @lru cache decorator from Python's functools module.

Reference List:

Astin, A., Vogelgesang, L., Ikeda, E. and Yee, J. (2000). *How Service Learning Affects Students*. [online] Available at: https://www.heri.ucla.edu/PDFs/HSLAS/HSLAS.PDF.

Dubow, E.F., Boxer, P. and Huesmann, L.R. (2009). Long-term Effects of Parents' Education on Children's Educational and Occupational Success: Mediation by Family Interactions, Child Aggression, and Teenage Aspirations. *Merrill-Palmer Quarterly*, 55(3), pp.224–249. doi:https://doi.org/10.1353/mpq.0.0030.

Duckworth, A.L. and Seligman, M.E.P. (2006). *APA PsycNet*. [online] psycnet.apa.org. Available at: https://psycnet.apa.org/doiLanding?doi=10.1037%2F0022-0663.98.1.198.

Furda, M. and Shuleski, M. (2019). The Impact of Extracurriculurs on Academic Performance and School Perception. *The Excellence in Education Journal*, [online] 8(1). Available at: https://files.eric.ed.gov/fulltext/EJ1208711.pdf.

Goyette, K. and Xie, Y. (1999). Educational Expectations of Asian American Youths: Determinants and Ethnic Differences. *Sociology of Education*, [online] 72(1), p.22. doi:https://doi.org/10.2307/2673184.

Nickow, A., Oreopoulos, P. and Quan, V. (2020). The Impressive Effects of Tutoring on Prek-12 Learning: A Systematic Review and Meta-Analysis of the Experimental Evidence. *SSRN Electronic Journal*. doi:https://doi.org/10.2139/ssrn.3644077.

Pei, C. (2024). Research on the influencing factors of student performance . *Theoretical and Natural Science*, 51(1), pp.26–33. doi:https://doi.org/10.54254/2753-8818/51/2024ch0131.

Sue, S. and Okazaki, S. (1990). Asian-American educational achievements: A phenomenon in search of an explanation. *American Psychologist*, 45(8), pp.913–920. doi:https://doi.org/10.1037/0003-066x.45.8.913.

The Annie E. Casey Foundation (2022). *Parental Involvement in Your Child's Education*. [online] The Annie E. Casey Foundation. Available at: https://www.aecf.org/blog/parental-involvement-is-key-to-student-success-research-shows.