PROJECT 1: BRIGHTPATH ACADEMY

Muhammad Toufeeq Parker 600947
Kyle Colin Haynes-Smart 601311
Shivaan Boodhoo 601067
Carmen Walliser 600169

Table of Contents

Introduction 2

GitHub link 2

Dashboard link 2

Problem statement 2

Hypothesis 2

Load Data into the System 3

Importing Libraries and loading data 3

Understanding the Data 4

Handel missing values 4

Categorical Encoding 5

Feature Scaling 5

Outlier Detection 5

Exploratory Data Analysis 6

Univariate Analysis 6

Bivariate Analysis 7

Class Imbalance (SMOTE) 8

Outlier Treatment 10

Model Building 11

Logistic Regression 11

Random Forest 13

XGBoost 15

Evaluation of the metrics 16

Introduction

GitHub link

https://github.com/ToufeeqParker786/GuidedProject_MLG382_2025.git

Dashboard link

https://guidedproject-mlg382-2025-2.onrender.com/

Problem statement

BrightPath academy is a high school that is committed to the academic excellence and holistic development of its students. BrightPath academy wants to help their student succeed but are currently facing numerous issues such as they are identifying at-risk students too late, teachers do not have sufficient personalised tools for struggling students, they are unsure whether extracurricular activities help or harm academic performance and there is plentiful data, but no way to easily use it to assist in making helpful decisions.

To address the issues mentioned, this project aims to develop a machine learning model that will be capable of predicting a student's academic performance and categorise it into grade categories based on various features like study time, tutoring, and parental support. This will be made possible by making use of algorithms such as Logistic Regression, Random Forest, XGBoost and Deep Learning (e.g., neural networks). The model will allow BrightPath academy to identify which students require intervention earlier, enable them to create tailored support strategies and give them an improved understanding of the factors that contribute towards a student success.

Hypothesis

Feature(s)	**Hypothesis Statement**	**Expected Relationship**
Study Time Weekly	Higher weekly study time is associated with better GradeClass.	Positive
Tutoring	Students receiving tutoring are more likely to achieve higher GradeClasses.	Positive
Absences	More absences are associated with lower GradeClasses.	Negative
Parental Support	Higher parental support leads to better GradeClasses.	Positive
Parental Education	Students with more educated parents perform better academically.	Positive
Extracurricular	Participation in extracurricular activities improves GradeClass.	Positive

Music	Music participation is associated with higher GradeClasses.	Positive
Volunteering	Volunteering is linked to higher academic performance.	Positive
Gender	No significant difference in GradeClass based on gender.	Neutral
Ethnicity	Ethnic background is related to GradeClass due to broader socioeconomic influences.	Varies

Load Data into the System

Importing Libraries and loading data

Importing libraries guarantees that all necessary features are accessible. The data's structure and content are verified by loading and previewing it.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import math

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from scipy.stats import f_oneway, chi2_contingency
from imblearn.over_sampling import SMOTE
from collections import Counter
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    confusion_matrix, ConfusionMatrixDisplay,
    roc_curve, RocCurveDisplay,
    accuracy_score, precision_score, recall_score, f1_score

)
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from xgboost import XGBClassifier
```

Libraries:

Data Manipulation & Numerical Operations

- pandas: For data loading, manipulation, and analysis (e.g., DataFrames, filtering, merging).
- numpy: For efficient numerical computations (e.g., arrays, mathematical functions).

Data Visualization

- matplotlib: Core library for creating static, animated, and interactive visualizations.

- seaborn: Built on top of matplotlib for easier and more attractive statistical plots (e.g., feature importance, distributions).

Preprocessing, Modeling, and Evaluation

- sklearn (scikit-learn):
    - Preprocessing: Scaling (StandardScaler, MinMaxScaler), encoding (OneHotEncoder, LabelEncoder).
    - Modeling: LogisticRegression, RandomForestClassifier.
    - Evaluation: Accuracy, precision, recall, F1 score, confusion matrix, classification report, cross-validation.

Advanced Modeling

- xgboost: Gradient boosting library optimized for speed and performance; used for the XGBoost model.

Imbalanced Data Handling

- imblearn (Imbalanced-learn):
    - Tools like SMOTE (Synthetic Minority Over-sampling Technique) to balance class distributions.

Loading Data:

```
Filename="Student_performance_data .csv" #importing .csv file using pandas
df=pd.read_csv(Filename)

df.head()
```

Reads the dataset into df

df.head() shows the dataset structure including the columns

Understanding the Data

Handel missing values

Missing values can bias models or cause errors. Verifying none exist ensures the data is ready for analysis.

```
df.isnull().sum() #check to see if any values are missing
```

```
StudentID          0
Age                0
Gender             0
Ethnicity          0
ParentalEducation  0
StudyTimeWeekly    0
Absences           0
Tutoring           0
ParentalSupport    0
Extracurricular    0
Sports             0
Music              0
Volunteering       0
GPA                0
GradeClass         0
dtype: int64
```

Categorical Encoding

Categorical variables are variables that have a limited number of unique values.

LabelEncoder() convert categorical variables into numerical values

```
#Categorical variables
categorical_cols = ['Gender', 'Ethnicity', 'ParentalEducation', 'Tutoring', 'ParentalSupport', 'Extracurricular', 'Sports', 'Music', 'Volunteering']
le = LabelEncoder()
for columns in categorical_cols:
    df[columns] = le.fit_transform(df[columns])
```

Feature Scaling

Feature scaling ensures numerical features (Age, StudyTimeWeekly, Absences, GPA) are normalized to a range of [0, 1], which helps models converge faster and perform better by eliminating scale differences.

scaler.fit_transform(df[numerical_columns]) computes the minimum and maximum values for each feature and scales the data. The result is assigned to the corresponding columns in df_minmax.

```
#Feature Scaling
numerical_columns = ['Age', 'StudyTimeWeekly', 'Absences', 'GPA']
scaler = MinMaxScaler()
df_minmax = df.copy()
df[numerical_columns] = scaler.fit_transform(df[numerical_columns])

print("Min-Max Scaled Data (First 5 rows):")
display(df_minmax[numerical_columns].head())
print("\nMin-Max Scaled Stats:")
display(df_minmax[numerical_columns].describe().T)
```

Outlier Detection

Identify and remove outliers in numerical features using the Interquartile Range (IQR) method to ensure data quality for model training. Outliers can skew model results, so the IQR method is used to detect and remove extreme values in numerical features

```python
#outlier detection using IQR method
print("Initial number of rows: ", len(df))

def remove_outliers(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    initial_rows = len(df)
    df_filtered = df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]
    print(f"\nRemoving outliers for {column}:")
    print(f"Q1: {Q1:.2f} \nQ3: {Q3:.2f}")
    print(f"IQR: {IQR:.2f}")
    print(f"Lower bound: {lower_bound:.2f}")
    print(f"Upper bound: {upper_bound:.2f}")
    print(f"Rows before: {initial_rows}, Rows after: {len(df_filtered)}")
    return df_filtered

initial_rows = len(df)
for col in numerical_columns:
    df = remove_outliers(df, col)
    if len(df) < 0.9 * initial_rows:
        print(f"Warning: Significant data loss after removing outliers for {col}.")

print('\nFinal number of rows after outlier removal:', len(df))
print("\nSummary of numerical columns after outlier removal:\n", df[numerical_columns].describe().T)
```

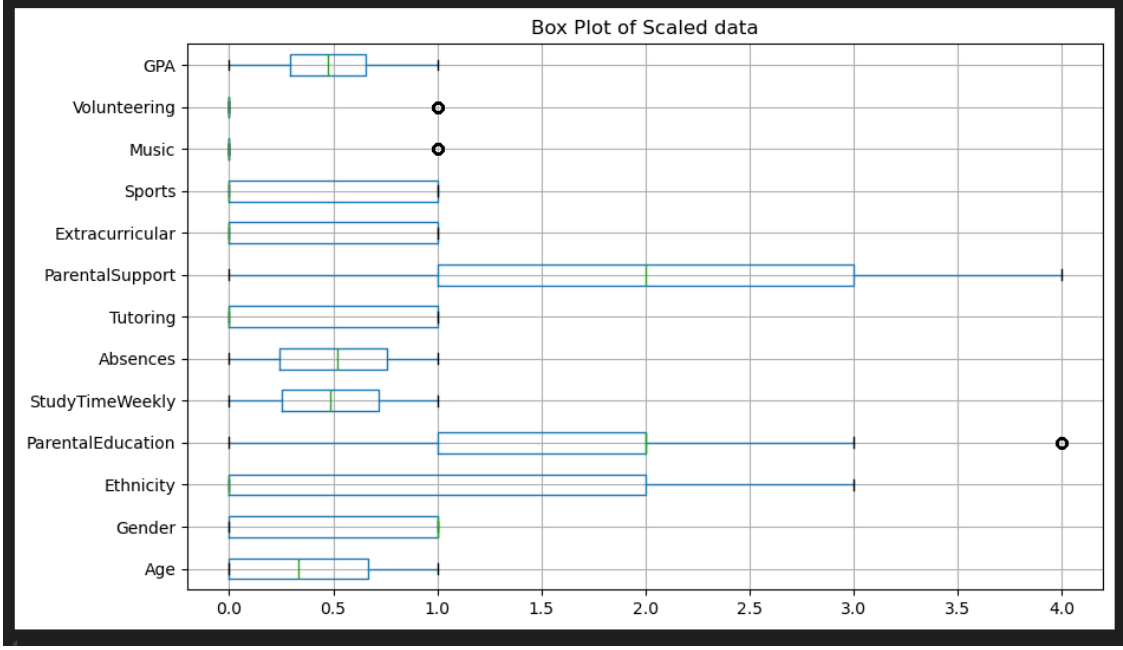
Exploratory Data Analysis

Univariate Analysis

The purpose of the Univariate Analysis is to analyse the distribution of each feature individually to understand its range, shape, and frequency. This help to inform preprocessing, guide model selection, and highlight target imbalance.

```python
scaler = MinMaxScaler()  # or StandardScaler()
df[['Age', 'StudyTimeWeekly', 'Absences', 'GPA']] = scaler.fit_transform(df[['Age', 'StudyTimeWeekly', 'Absences', 'GPA']])

fig, ax=plt.subplots(figsize=(10, 6))
boxplot=df[features].boxplot(vert=0,ax=ax)
_=ax.set_title(f'Box Plot of Scaled data')
```



Numerical features use sns.histplot with a kernel density estimate (kde) to visualize distribution

```
plt.figure(figsize=(12, 10))

for i, col in enumerate(features):

    plt.subplot(math.ceil(len(features) / 3), 3, i + 1)

    sns.histplot(df[col], kde=True)

    plt.title(col)

plt.tight_layout()

plt.show()
```

Bivariate Analysis

The purpose is to examines relationships between features and the target (GradeClass) or among features to identify predictive patterns.

Box plots are ideal for comparing numerical distributions across categories. Informs feature importance

```
df = pd.read_csv('Student_performance_data .csv')

target = 'GradeClass'

# Define numerical and categorical features (excluding target)
numerical_features = ['Age', 'StudyTimeWeekly', 'Absences', 'GPA']
categorical_features = ['Gender', 'Ethnicity', 'ParentalEducation', 'Tutoring',
                        'ParentalSupport', 'Extracurricular', 'Sports', 'Music', 'Volunteering']

# ------------------------------
# NUMERICAL → CATEGORICAL (Box Plots)
# ------------------------------
for feature in numerical_features:
    plt.figure(figsize=(7, 5))
    sns.boxplot(data=df, x=target, y=feature)
    plt.title(f'{feature} distribution across {target}')
    plt.tight_layout()
    plt.show()
```

Visualizes relationships between categorical features and GradeClass and tests their statistical significance using Chi-square tests.

sns.countplot – creates a count plot with the features on the x-axis and GradeClass as the hue

Chi-Square test

- pd.crosstab creates a contingency table of feature vs GradeClass count
- chi2_contingency computes Chi-square statistic, p-value, degree of freedom and expected frequencies

```
# ------------------------------
# CATEGORICAL → CATEGORICAL (Count Plots + Chi-Square)
# ------------------------------
chi_square_results = {}

for feature in categorical_features:
    plt.figure(figsize=(7, 5))
    sns.countplot(data=df, x=feature, hue=target)
    plt.title(f'{feature} vs {target}')
    plt.legend(title=target)
    plt.tight_layout()
    plt.show()

    # Chi-square test
    contingency = pd.crosstab(df[feature], df[target])
    chi2, p, dof, expected = chi2_contingency(contingency)
    chi_square_results[feature] = p
```

Tests if numerical features differ significantly across GradeClass categories using ANOVA. ANOVA is suitable for comparing means across multiple groups

Tests categorical feature significance using Chi-square tests and adds to summary to ensures all features are evaluated consistently

```
# ------------------------------
# Show Chi-Square Test Results
# ------------------------------
summary = []

# Analyze numerical features
for feature in numerical_features:
    groups = [group[feature].dropna() for name, group in df.groupby(target)]
    _, p = f_oneway(*groups)
    significant = p < 0.05
    summary.append((feature, 'Numerical', "Yes" if significant else "No"))

# Analyze categorical features
for feature in categorical_features:
    contingency = pd.crosstab(df[feature], df[target])
    _, p, _, _ = chi2_contingency(contingency)
    significant = p < 0.05
    summary.append((feature, 'Categorical', "Yes" if significant else "No"))

# Create and display final summary
results_df = pd.DataFrame(summary, columns=['Feature', 'Type', 'Significant'])
print(" Summary of Features Significantly Related to GradeClass:\n")
print(results_df.to_string(index=False))
```

Class Imbalance (SMOTE)

Defines a function to convert GPA to letter grades

```python
df = pd.read_csv("Student_performance_data .csv")


def assign_grade_class(gpa):
    if gpa >= 3.5:
        return 'A'
    elif gpa >= 3.0:
        return 'B'
    elif gpa >= 2.5:
        return 'C'
    elif gpa >= 2.0:
        return 'D'
    else:
        return 'F'
```

Applies the assign_grade_class function to create a new column.

Encodes GradeLetter into numerical labels for SMOTE.

Defines features (X) and target (y) for SMOTE.

```python
df['GradeLetter'] = df['GPA'].apply(assign_grade_class)


label_encoder = LabelEncoder()
df['GradeClassEncoded'] = label_encoder.fit_transform(df['GradeLetter'])


X = df.drop(['GradeLetter', 'GradeClassEncoded', 'StudentID', 'GPA'], axis=1)
y = df['GradeClassEncoded']
```

Displays the class distribution of GradeClassEncoded

Counts occurrences of each encoded class.

Dictionary comprehension - Maps letters to counts.

label_encoder.inverse_transform - Converts encoded values back to letters

```python
original_counts = dict(Counter(y))
print("Original class distribution:", {label_encoder.inverse_transform([k])[0]: v for k, v in original_counts.items()})
```

Balances classes using SMOTE

fit_resample - Generates synthetic samples for minority classes to match the majority class count.

Converts encoded y_resampled back to letter grades for interpretation.

Displays the balanced class distribution.

```
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

y_resampled_letters = label_encoder.inverse_transform(y_resampled)


resampled_counts = dict(Counter(y_resampled))
print("After SMOTE class distribution:", {label_encoder.inverse_transform([k])[0]: v for k, v in resampled_counts.items()})
```

Outlier Treatment

Outliers can skew model predictions, especially for linear models like Logistic Regression. Ensures data quality for downstream modeling

Computes the 25th (Q1) and 75th (Q3) percentiles for ParentalEducation

```
# Calculate Q1 (25th percentile) and Q3 (75th percentile)
Q1 = df["ParentalEducation"].quantile(0.25)
Q3 = df["ParentalEducation"].quantile(0.75)
```

Filters out rows where ParentalEducation is outside the bounds.

df["ParentalEducation"] > lower_bound - Keeps values above lower_bound

df["ParentalEducation"] <= upper_bound - Keeps values at or below upper_bound

```
# Calculate IQR
IQR = Q3 - Q1

# Define bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Remove outliers
df_cleaned = df[(df["ParentalEducation"] >= lower_bound) & (df["ParentalEducation"] <= upper_bound)]
df.info()
```

Model Building

Logistic Regression

Defines features (X) and target (y) for modeling

Converts categorical features to dummy variables

```
x = df.drop(['GradeClass','StudentID'], axis=1)
y=df["GradeClass"]
x = pd.get_dummies(x)
```

Standardizes features to mean=0, variance=1.

Logistic Regression is sensitive to feature scales

```
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
```

Splits data into training (80%) and testing (20%) sets for model validation

```
X_train,X_test,y_train,y_test=train_test_split(x_scaled,y,test_size=0.2,random_state=42)
```

Trains a Logistic Regression model.

Generates predictions on training and test sets.

```
model = LogisticRegression(random_state = 0, solver='lbfgs',max_iter=1000)
model.fit(X_train, y_train)

y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)
```

Attempts to plot an ROC curve for binary classification

predict_proba(X_test) - Gets probability scores.

roc_curve - Computes false positive rate (FPR) and true positive rate (TPR).

RocCurveDisplay - Plots ROC curve.

```python
# ROC Curve (Only if binary classification)
if len(model.classes_) == 2:
    y_probs = model.predict_proba(X_test)[:, 1]
    fpr, tpr, _ = roc_curve(y_test, y_probs, pos_label=model.classes_[1])
    RocCurveDisplay(fpr=fpr, tpr=tpr).plot()
    plt.title("ROC Curve")
    plt.show()
else:
    print("ROC Curve skipped (only applicable for binary classification).")
```

Confusion matrices are standard for evaluating classification models, especially in multiclass settings where accuracy alone is insufficient. Helps identify misclassifications

```python
cm = confusion_matrix(y_test, y_test_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.title("Confusion Matrix")
plt.show()
```

Interpretation:

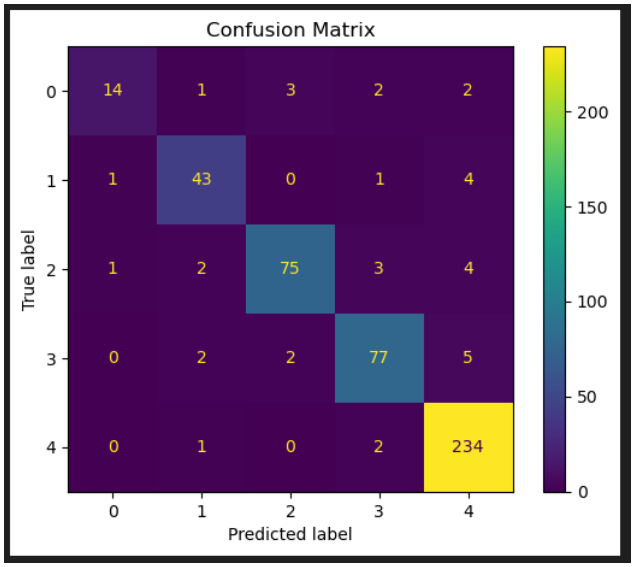
Class 0 (A): 14 correct predictions.

Class 1 (B): 43 correct predictions.

Class 2 (C): 75 correct predictions.

Class 3 (D): 77 correct predictions.

Class 4 (F): 234 correct predictions.



Random Forest

Loads data and creates new features to enhance model performance. Feature engineering enhances model accuracy by summarizing key patterns

```
data = pd.read_csv('Student_performance_data .csv')
data['StudyEfficienty'] = data['StudyTimeWeekly'] / (data['Absences'] + 1)
data['ActivityScore'] = data[['Extracurricular', 'Sports', 'Music', 'Volunteering']].sum(axis=1)
data['ParentalInfluence'] = data['ParentalEducation'] * 0.4 + data['ParentalSupport'] * 0.6
```

```
#To define features (x) and target (y)
X = data.drop(['StudentID', 'GradeClass', 'GPA'], axis=1, errors='ignore')
y = data['GradeClass']

#Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Initialize and train Random Forest model
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
```

Confusion Matrix is critical for multiclass evaluation, comparing Random Forest to Logistic Regression.

```
# Make predictions
y_pred_rf = rf.predict(X_test)

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred_rf)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='summer', xticklabels=['A', 'B', 'C', 'D', 'F'], yticklabels=['A', 'B', 'C', 'D', 'F'])
plt.title('Confusion Matrix - Random Forest')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

Diagonal (Correct Predictions):

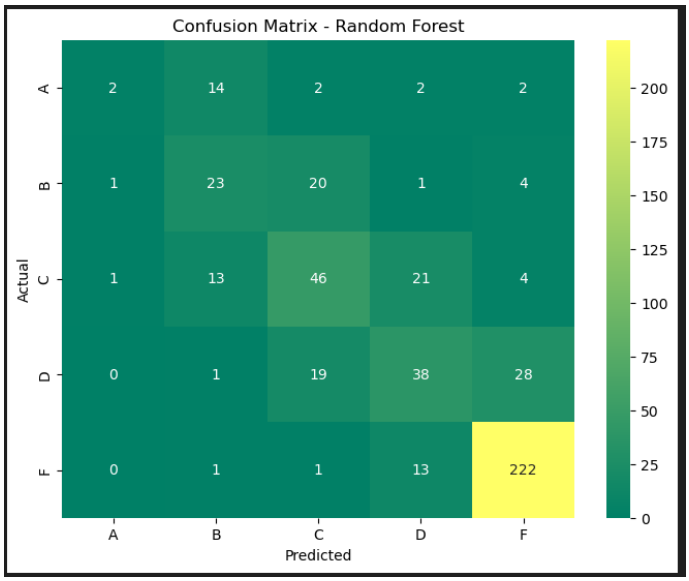
- A (0): 14 correct (out of ~22 true A's, per Logistic Regression matrix).
- B (1): 23 correct (out of ~49 true B's).
- C (2): 46 correct (out of ~85 true C's).
- D (3): 38 correct (out of ~86 true D's).
- F (4): 222 correct (out of ~237 true F's).

Off-Diagonal (Misclassifications):

- A: 2 predicted as B, 2 as C, 2 as D, 2 as F (spread errors).
- B: 20 predicted as C, 4 as F, 1 as A (confusion with C).
- C: 21 predicted as D, 13 as B, 4 as F, 1 as A.
- D: 28 predicted as F, 19 as C, 1 as B.

F: 13 predicted as D, 1 as B, 1 as C (few errors).

Compares Random Forest to Logistic Regression, showing Logistic Regression's edge due to GPA inclusion.



Random Forest Feature Importance

Random Forest's feature importance helps understand model behavior and prioritize features.

```
# Feature Importance
feature_importance = pd.DataFrame({
    'Feature': X.columns,
    'Importance': rf.feature_importances_
}).sort_values(by='Importance', ascending=False)

print("\nFeature Importance:\n", feature_importance)

# Plot feature importance
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feature_importance)
plt.title('Feature Importance - Random Forest')
plt.show()
```

Top Features:

- Absences (0.383574): Most influential (38.36%), likely because high absences strongly correlate with lower grades (F), per Bivariate Analysis.
- StudyEfficiency (0.238006): Second highest (23.80%), capturing the ratio of study time to absences, a key engineered feature.
- StudyTimeWeekly (0.114772): Third (11.48%), reflecting study effort's role in grades.

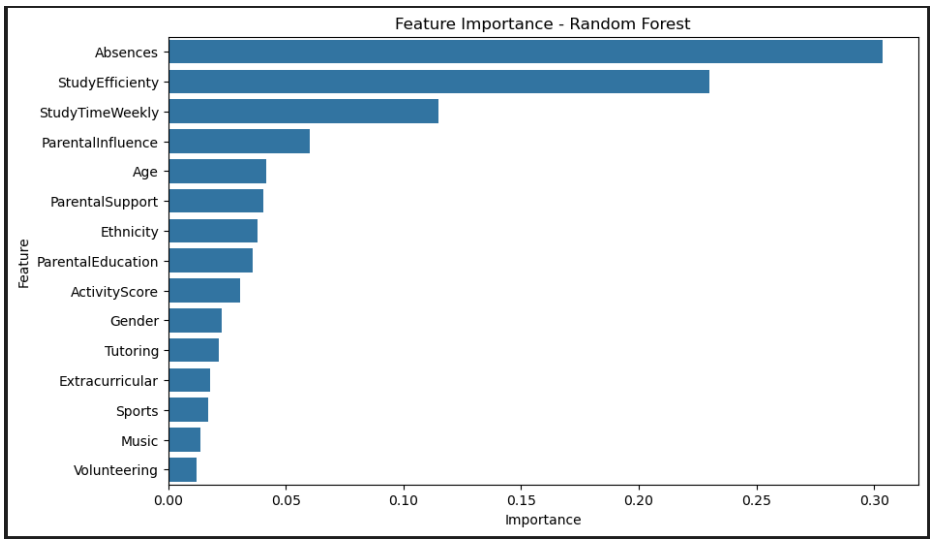
Mid-Tier Features:

- ParentalInfluence (0.060297): Engineered feature (6.03%), combining ParentalEducation and ParentalSupport, moderately impactful.

- Age (0.041684), ParentalSupport (0.040281), Ethnicity (0.037767), ParentalEducation (0.035969), ActivityScore (0.032776): Each contributes 3–4%, suggesting secondary influence.

Low-Impact Features:

- Gender (0.022776), Tutoring (0.021650), Extracurricular (0.017929), Sports (0.017007), Music (0.013883), Volunteering (0.012091): Each below 2.3%, indicating minimal impact on predictions.



XGBoost

XGBoost is a highly effective machine learning algorithm that builds sequential decision trees to correct errors and minimize a loss function for tasks like classification and regression. It enhances performance through gradient boosting, regularization, parallel processing, and robust handling of missing data.

```
file_path = 'Student_performance_data .csv'
df = pd.read_csv(file_path)

# Set target and features
X = df.drop('GradeClass', axis=1)
y = df['GradeClass']

# Identify numeric and categorical columns
numeric_features = X.select_dtypes(include=['int64', 'float64']).columns.tolist()
categorical_features = X.select_dtypes(include=['object', 'category']).columns.tolist()
```

```python
# Preprocessing: scale numeric + encode categoricals
preprocessor = ColumnTransformer(transformers=[
    ('num', StandardScaler(), numeric_features),
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
])

# Full pipeline: preprocessing + model
pipeline = Pipeline(steps=[
    ('preprocessing', preprocessor),
    ('classifier', XGBClassifier(use_label_encoder=False, eval_metric='mlogloss', random_state=42))
])

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Fit the model
pipeline.fit(X_train, y_train)

# Predict and evaluate
y_pred = pipeline.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy on test set:", accuracy)
```

The model achieved **91.65% accuracy** on the test set, meaning 91.65% of the ~479 test samples (~439) were correctly predicted for GradeClass (A to F).

```
Accuracy on test set: 0.916492
```

Evaluation of the metrics

This evaluation step fulfils the project agenda's requirement to evaluate models using accuracy, precision, recall, and F1-score, particularly for imbalanced datasets.

The deep learning model achieves an accuracy of 80.58%, with strong performance for class 4 (F) but lower recall for class 0 (A), indicating potential class imbalance that may require techniques like SMOTE, as suggested in the agenda.

```python
# Evaluate on test set
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy*100:.2f}%")

# Predictions
y_pred_probs = model.predict(X_test)
y_pred_classes = np.argmax(y_pred_probs, axis=1)
y_true_classes = np.argmax(y_test, axis=1)

# Ensure that label_encoder.classes_ contains valid labels and convert to string
class_labels = [str(label) for label in label_encoder.classes_]
```

```
# Proceed with the classification report
print("\nClassification Report:")
print(classification_report(y_true_classes, y_pred_classes, target_names=class_labels))
```

- **Precision**: The ratio of correct predictions for a class to the total predictions for that class
  - e.g. 1.00 for class 0=A means 100% of the predicted A grades were correct
- **Recall**: The ratio of correct predictions for a class to the total actual instances of that class
  - e.g. 0.14 for class 0=A means 14% of the actual A grades were correctly predicted
- **F1-Score**: The harmonic means of precision and recall
  - e.g. 0.24 for class 0=A
- **Support**: The number of test samples for each class
  - e.g. 22 for class 0=A

The overall accuracy is 0.81 (81%), with macro and weighted averages for precision, recall, and F1-score indicating balanced performance across classes.

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7808 - loss: 0.7207
Test Accuracy: 80.58%
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step

Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      0.14      0.24        22
         1.0       0.57      0.78      0.66        49
         2.0       0.76      0.65      0.70        85
         3.0       0.75      0.71      0.73        86
         4.0       0.89      0.97      0.93       237

    accuracy                           0.81       479
   macro avg       0.80      0.65      0.65       479
weighted avg       0.82      0.81      0.79       479
```