4/22/2025

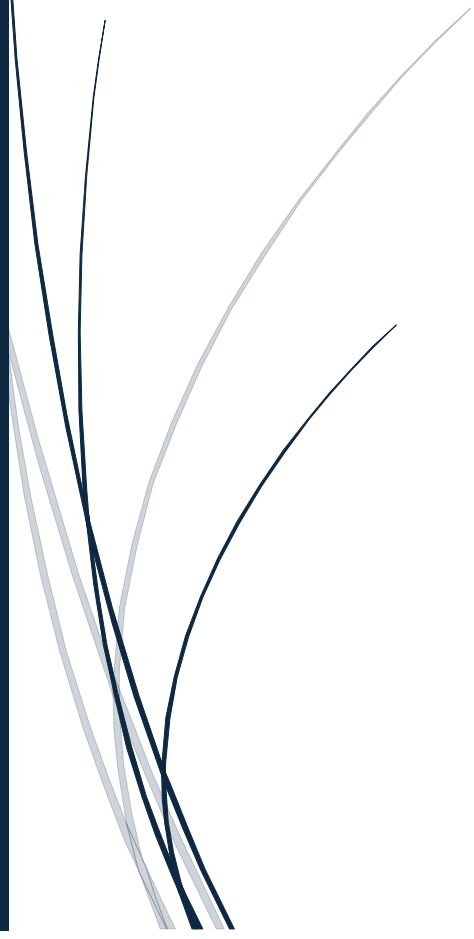
MLG382 Guided Project

Waldo Blom (578068)

Erin David Cullen (600531)

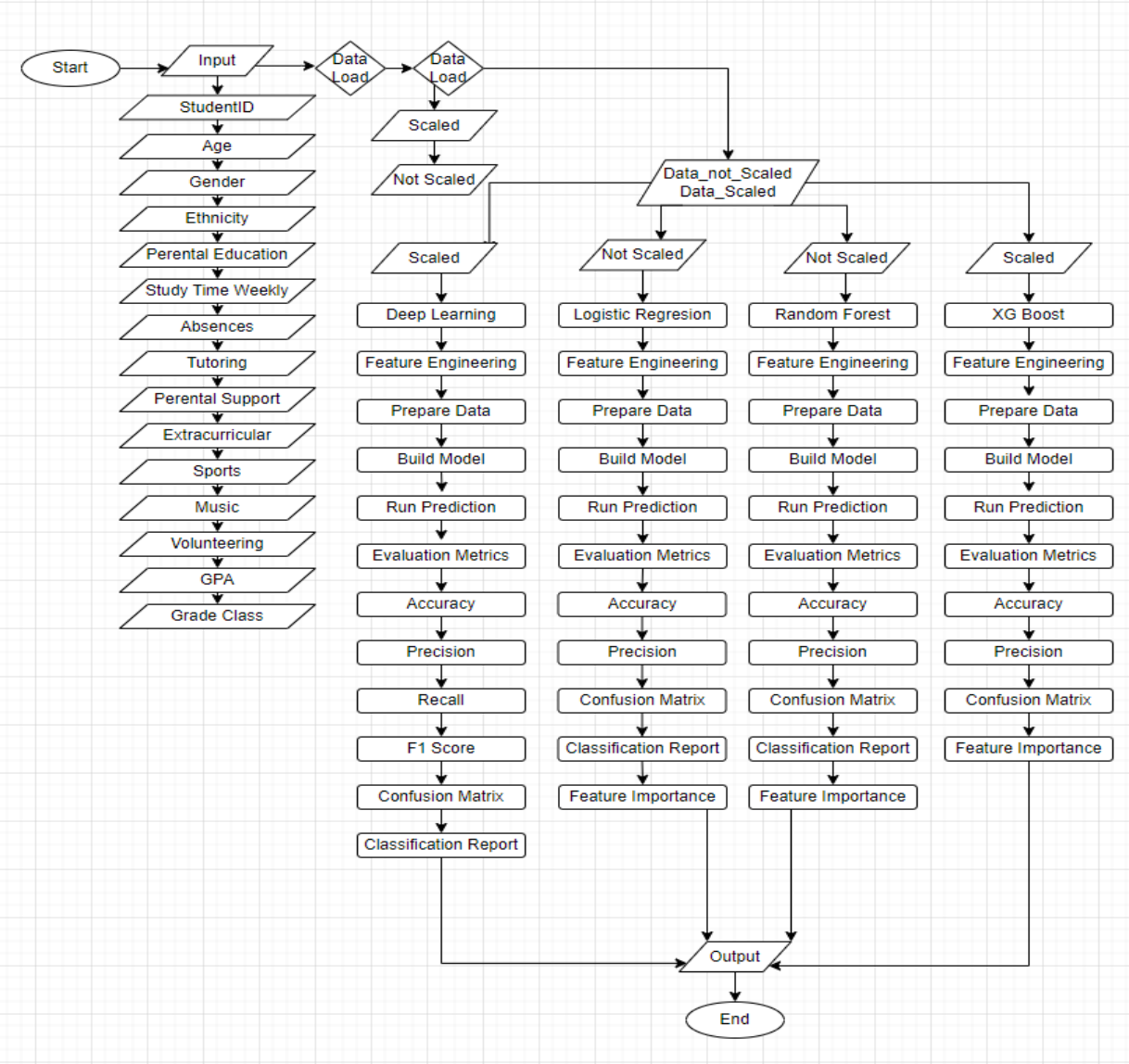
Brandon Alfonso Lemmer (578062)

Tristan James Ball (601541)



Contents:

Flow chart to visualize our Machine learning prosses:



Problem statement

BrightPath Academy faces some challenges that cause them to not be able to fully support each student such as:

- **Delayed identification of At-Risk Students:** BrightPath does not have access to a system that gives them real-time insights to the students who are struggling academically so they are not able to intervene in time.
- **No individual support strategies**: Educators don't have access to a tool that are able to identify the reason for why exactly a specific student is struggling.
- **Unknown effect of Extracurricular activities:** At BrightPath extracurricular involvement is encourage but educators are now unsure if extracurricular activities are the cause of poor academic performance. They want to identify whether this is the cause for poor academic performance.
- **No centralized system for actionable insights:** BrightPath has access to a lot of student data but no system that has the ability to analyse and understand the data.

Hypotheses Generation

The following hypotheses will guide our data exploration and modelling:

1. **Study Time and GradeClass:** There is a negative correlation between weekly study time (**StudyTimeWeekly**) and **GradeClass**, for example students who study more hours per week tend to have better grade classifications (lower GradeClass values, e.g., 'A' or 'B').

2. **Absences and GradeClass:** There is a positive correlation between the number of absences (**Absences**) and **GradeClass**, indicating that students with more absences are likely to have weaker grade classifications (higher GradeClass values, e.g., 'D' or 'F').

3. **Tutoring and GradeClass:** Tutoring status (**Tutoring**) is associated with **GradeClass** though it is difficult to know to what extent as the level of quality of the tutoring is not known, thus this will be explored further in the data analysis phase.

4. **Parental Support and GradeClass**: There is a negative correlation between the level of parental support **(ParentalSupport**) and **GradeClass**, suggesting that higher parental support is linked to better grade classifications.

5. **Extracurricular Activities and GradeClass**: Participation in extracurricular activities (**Extracurricular**) is associated with better **GradeClass**, thus showing a positive impact of these activities on academic performance.

6. **Parental Education and GradeClass**: There is a negative correlation between parental education level (**ParentalEducation**) and **GradeClass**, indicating that students with parents who have higher education levels are more likely to have better grade classifications.

These hypotheses will be tested and refined during the data analysis phase to uncover the factors influencing student performance, with a focus on the role of extracurricular activities as mentioned in the problem statement

Data Understanding

Data Inspection: We already know the target feature is GradeClass.

1. **Preview:** Gives an understanding of the dataset structure (names and general characteristics).

	StudentID	Age	Gender	Ethnicity	ParentalEducation	StudyTimeWeekly	Absences	Tutoring	ParentalSupport	Extracurricular	Sports	Music	Volunteering	GPA	GradeClass
0	1001	17	1	0	2	19.833723	7	1	2	0	0	1	0	2.929196	2.0
1	1002	18	0	0	1	15.408756	0	0	1	0	0	0	0	3.042915	1.0
2	1003	15	0	2	3	4.210570	26	0	2	0	0	0	0	0.112602	4.0
3	1004	17	1	0	3	10.028829	14	0	3	1	0	0	0	2.054218	3.0
4	1005	17	1	0	2	4.672495	17	1	3	0	0	0	0	1.288061	4.0

2. **Information:** Provides overview of data types as well as total entries and columns

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2392 entries, 0 to 2391
Data columns (total 15 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   StudentID         2392 non-null   int64  
 1   Age               2392 non-null   int64  
 2   Gender            2392 non-null   int64  
 3   Ethnicity         2392 non-null   int64  
 4   ParentalEducation 2392 non-null   int64  
 5   StudyTimeWeekly   2392 non-null   float64
 6   Absences          2392 non-null   int64  
 7   Tutoring          2392 non-null   int64  
 8   ParentalSupport   2392 non-null   int64  
 9   Extracurricular   2392 non-null   int64  
 10  Sports            2392 non-null   int64  
 11  Music             2392 non-null   int64  
 12  Volunteering      2392 non-null   int64  
 13  GPA               2392 non-null   float64
 14  GradeClass        2392 non-null   float64
dtypes: float64(3), int64(12)
memory usage: 280.4 KB
```

3. **Describe:** Shows a more detailed summary of each features statistics (mean, standard deviation, frequency counts, etc.).

	count	mean	std	min	25%	50%	75%	max
StudentID	2392.0	2196.500000	690.655244	1001.000000	1598.750000	2196.500000	2794.250000	3392.000000
Age	2392.0	16.468645	1.123798	15.000000	15.000000	16.000000	17.000000	18.000000
Gender	2392.0	0.510870	0.499986	0.000000	0.000000	1.000000	1.000000	1.000000
Ethnicity	2392.0	0.877508	1.028476	0.000000	0.000000	0.000000	2.000000	3.000000
ParentalEducation	2392.0	1.746237	1.000411	0.000000	1.000000	2.000000	2.000000	4.000000
StudyTimeWeekly	2392.0	9.771992	5.652774	0.001057	5.043079	9.705363	14.408410	19.978094
Absences	2392.0	14.541388	8.467417	0.000000	7.000000	15.000000	22.000000	29.000000
Tutoring	2392.0	0.301421	0.458971	0.000000	0.000000	0.000000	1.000000	1.000000
ParentalSupport	2392.0	2.122074	1.122813	0.000000	1.000000	2.000000	3.000000	4.000000
Extracurricular	2392.0	0.383361	0.486307	0.000000	0.000000	0.000000	1.000000	1.000000
Sports	2392.0	0.303512	0.459870	0.000000	0.000000	0.000000	1.000000	1.000000
Music	2392.0	0.196906	0.397744	0.000000	0.000000	0.000000	0.000000	1.000000
Volunteering	2392.0	0.157191	0.364057	0.000000	0.000000	0.000000	0.000000	1.000000
GPA	2392.0	1.906186	0.915156	0.000000	1.174803	1.893393	2.622216	4.000000
GradeClass	2392.0	2.983696	1.233908	0.000000	2.000000	4.000000	4.000000	4.000000

From this information we can gather:

1. StudentID can be exluded as it has no effect on other data (it is an identifier).
2. Data is split up into Categorical (Ordinal/Binary) and Numerical (Discrete/Continuous):

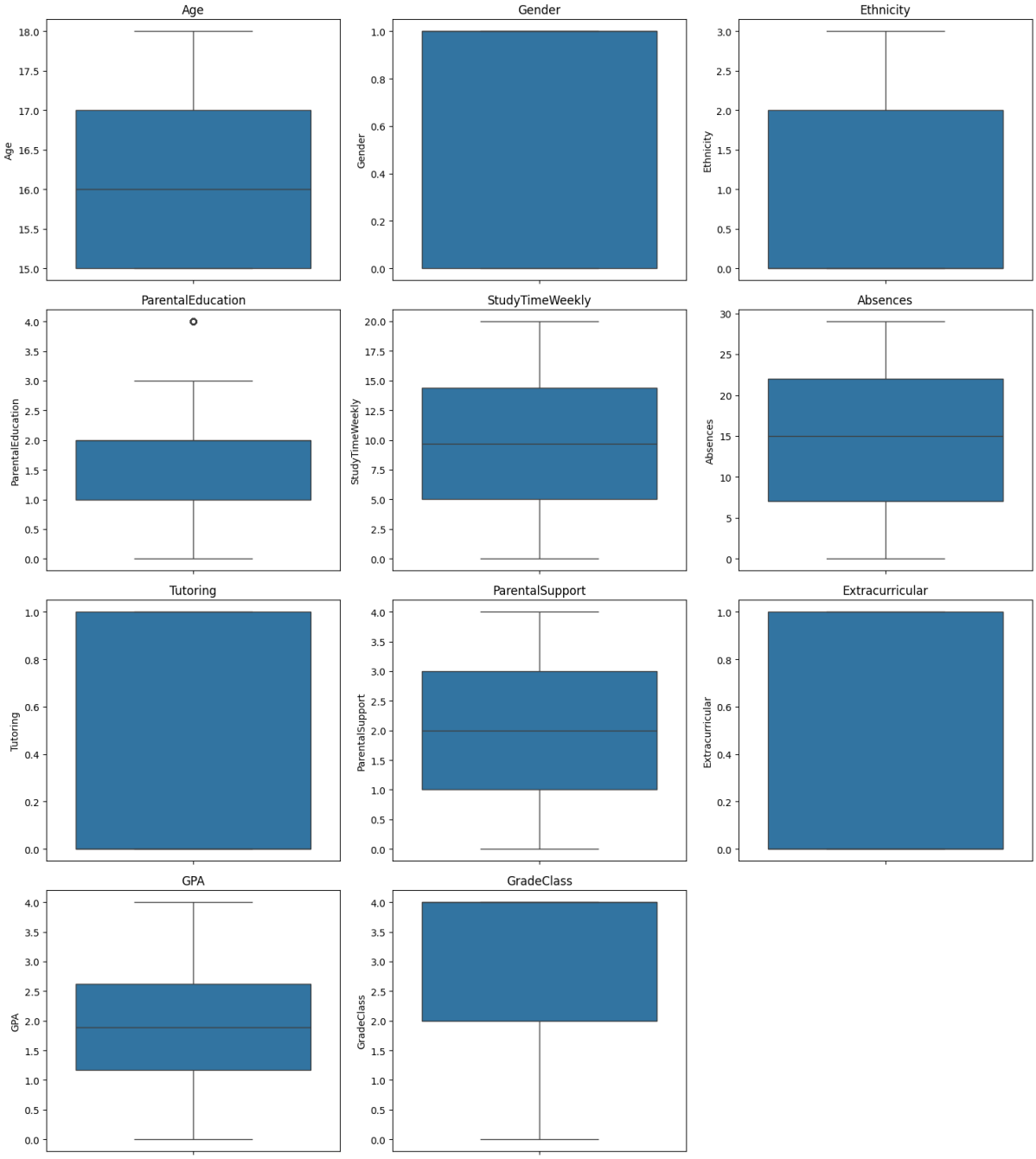
Categorical Variables	Numerical Variables
Ordinal:	**Discrete:**
- Ethnicity - ParentalEducation - ParentalSupport - GradeClass	- Age - Absences
Binary:	**Continuous:**
- Gender - Tutoring	- StudyTimeWeekly - GPA

- Extracurricular - Sports - Music - Volunteering	

Exploratory Data analysis

Univariate Analysis

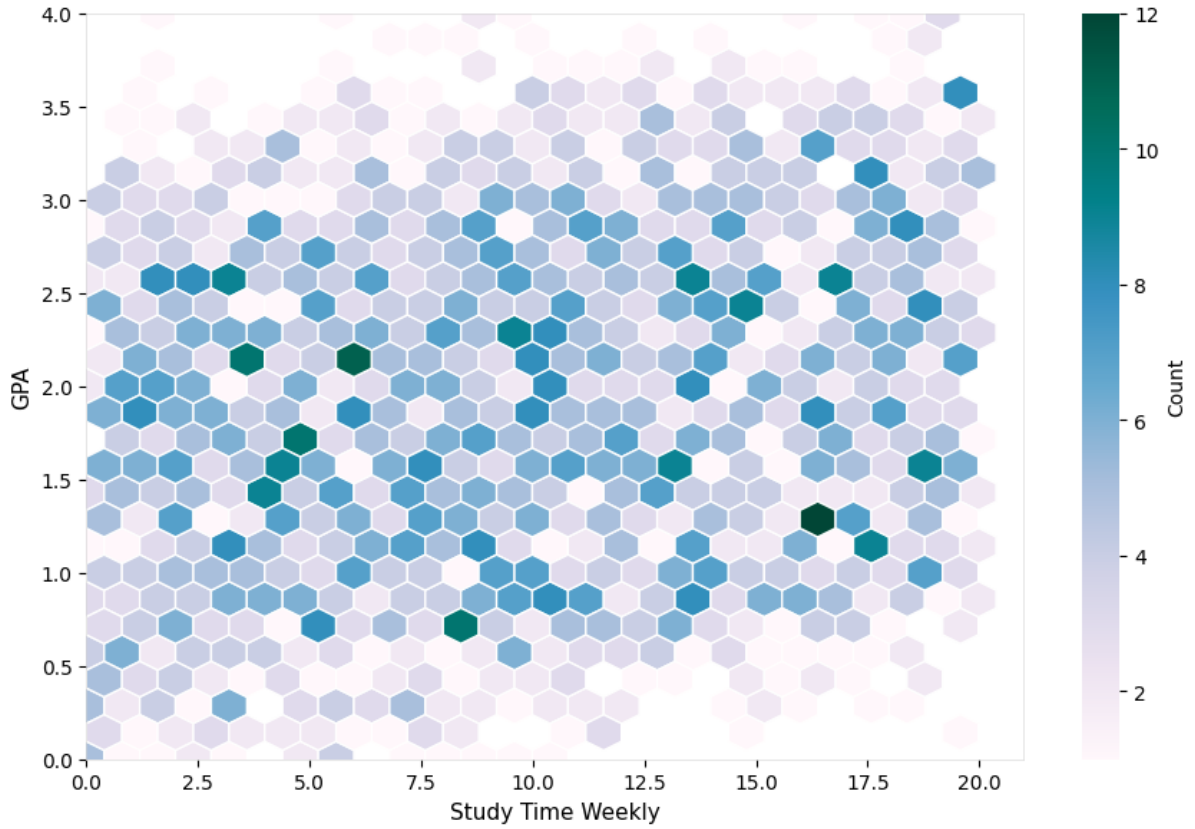
- Is the simplest form of statistical analysis. Comparing one variable at a time to its distribution, mean or outliers.
- We will conduct an analysis on the mean of (Age, Gender, Ethnicity, ParentalEducation, StudyTimeWeekly, Absences, Tutoring, ParentalSupport, Extracurricular, GPA, GradeClass)



Bivariate Analysis

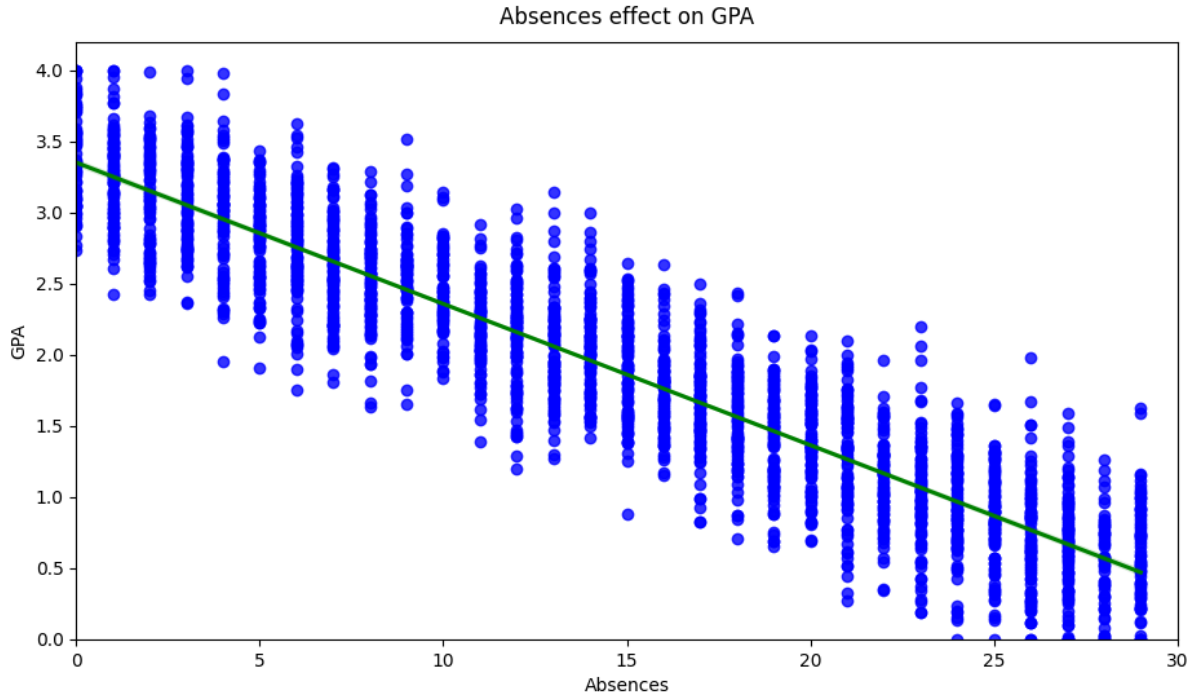
- Is used to examine the relationship between two variables to determine if there is association, correlation or causation between them.
- **We will conduct an analysis on the relationships of:** (Gender, Ethnicity, ParentalEducation, StudyTimeWeekly, Absences, Tutoring, ParentalSupport, Extracurricular, Volunteering) with (GPA)

1. Study time weekly does increase GPA but by a small amount.



The darker green shows us that if you spend less time studying your GPA will be lower.

2. Absences had a drastic effect on GPA, the less you stay home the better you do.



This graph shows us that the more you stay home the lower you GPA will be.

Missing value and Outlier treatment

Clean the mess before modelling.

```
df.info()
```

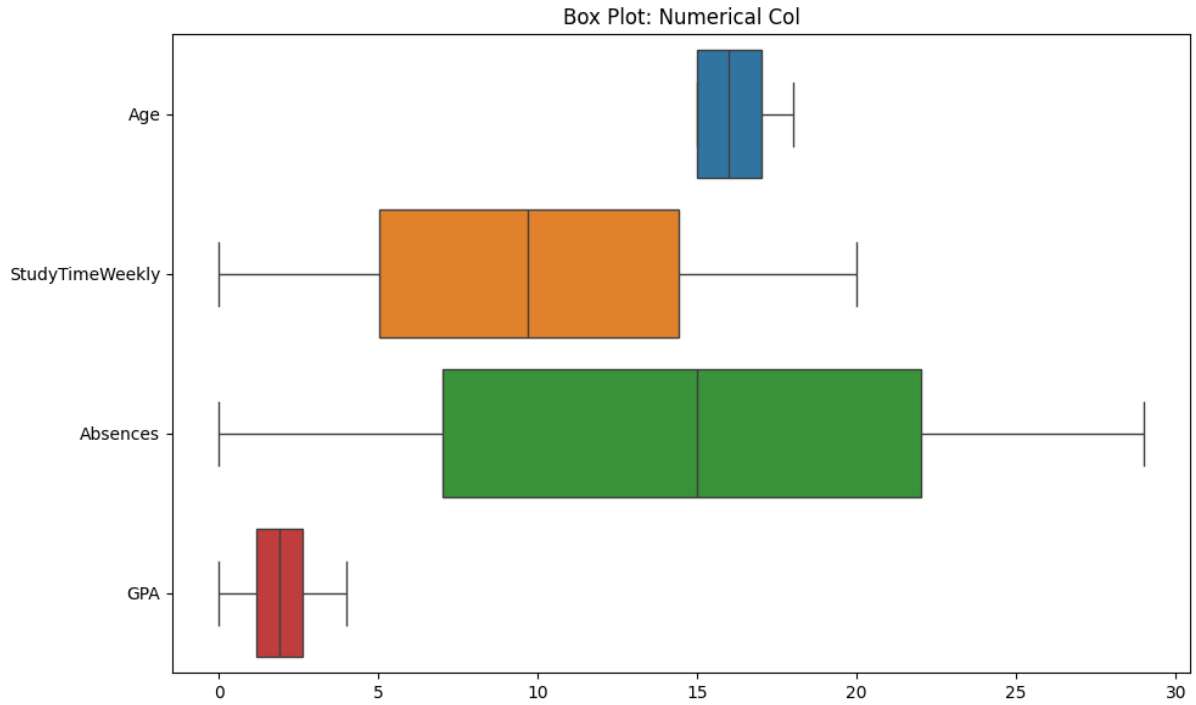
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2392 entries, 0 to 2391
Data columns (total 15 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   StudentID          2392 non-null   int64  
 1   Age                2392 non-null   int64  
 2   Gender             2392 non-null   int64  
 3   Ethnicity          2392 non-null   int64  
 4   ParentalEducation  2392 non-null   int64  
 5   StudyTimeWeekly    2392 non-null   float64
 6   Absences           2392 non-null   int64  
 7   Tutoring           2392 non-null   int64  
 8   ParentalSupport    2392 non-null   int64  
 9   Extracurricular    2392 non-null   int64  
 10  Sports             2392 non-null   int64  
 11  Music              2392 non-null   int64  
 12  Volunteering       2392 non-null   int64  
 13  GPA                2392 non-null   float64
 14  GradeClass         2392 non-null   float64
dtypes: float64(3), int64(12)
memory usage: 280.4 KB
```

Missing value treatment:

From the above we can see all the above columns have the same non-null count of 2392 meaning that there are no missing values.

Outlier Treatment:

We are only doing outlier treatment on the numerical variables not the categorical varia:




```
Column Age has 0 outliers.
Column StudyTimeWeekly has 0 outliers.
Column Absences has 0 outliers.
Column GPA has 0 outliers.
```

From the above we can see there are no outliers present in the numerical data.

Model evaluation metrics

Once each model has been trained, we need set ways to evaluate its performance, since the classification problem is multi-class (GradeClass = 0->4), we use metrics that account for multi-class.

We need to generate a model before testing it, example Random Forest Classifier:

- Split the data into training and testing sets

  `X = df.drop(['GradeClass', 'GPA', 'StudentID', 'Ethnicity'], axis=1)`

  `y = df['GradeClass']`

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

- Train a Random Forest Classifier:

  `rf_model = RandomForestClassifier(random_state=42)`

  `rf_model.fit(X_train, y_train)`

- Make predictions:

  `y_pred = rf_model.predict(X_test)`

Accuracy:

- Provides the ratio of correctly predicted observations to total observations, good for balanced datasets but struggles if there are too many of one class.
- Calculate accuracy score:

  `accuracy = accuracy_score(y_test, y_pred)`
  `print(f"Accuracy Score: {accuracy:.2f}")`

Precision:

- Compares the level of predicated population in a class vs how many were actually in that class, provides an understanding on false positives and high precision = low false positives
- Calculate precision score:

  `precision = precision_score(y_test, y_pred, average='weighted')`

Recall:

- Returns how many real students were correctly identified in a specific class, provides an understanding on false negatives and high recall = low false negatives
- Calculate recall score:

  `recall = recall_score(y_true, y_pred, average='macro')`

F1-Score:

- Gives a balance between Precision and Recall, useful for imbalanced class problems

- Calculate F1-score:

  `f1 = f1_score(y_true, y_pred, average='macro')`

Confusion Matrix:

- Gives a full breakdown of how many entries were correctly and incorrectly classified in each class, useful for showing which classes each model is struggling with
- Calculate confusion matrix:

  `cm = confusion_matrix(y_true, y_pred)`

  `sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')`

  `plt.xlabel('Predicted')`

  `plt.ylabel('Actual')`

  `plt.title('Confusion Matrix')`

  `plt.show()`

Feature engineering

Make the data smarter

Compatibility of Feature Types with Selected ML Models

	Feature Engineering Type	Logistic Regression	Random Forest	XGBoost	Deep Learning
0	Encoding Categorical Variables	✅ Required	✅ Optional	✅ Optional	✅ Required
1	Scaling Numerical Features	✅ Yes	❌ Not needed	❌ Not needed	✅ Yes
2	Interaction Features	✅ Improves linearity	✅ Fine	✅ Great	✅ Useful
3	Ratio & Aggregate Features	✅ Often helps	✅ Great	✅ Great	✅ Very good

Why and Where Each Feature Engineering Technique Is Useful

	Feature Type	Why It's Useful	Needed For
0	Encoding Categoricals	Converts labels to usable format	LogReg, DL
1	Interaction Features	Captures nonlinear relationships	All models
2	Ratio/Aggregate	Creates informative summaries	All models
3	Scaling	Prevents skew in distance-based methods	LogReg, DL
4	Binning/Flags (optional)	Increases interpretability, simplifies complexity	All models (optional)

Encoding Categorical Variables:

- Label Encoding (for ordinal data)

- One-Hot Encoding (for nominal data)

Scaling Numerical Features:

- StandardScaler, MinMaxScaler, or RobustScaler (Important for Logistic Regression & Deep Learning)

Binning:

- Multiplying or combining features to capture interactions E.g., StudyTimeWeekly * ParentalSupport

Ratio and Aggregate Features:

- Creating ratios like StudyTimeWeekly / Absences
- Aggregates like TotalExtracurricular = Sports + Music + Volunteering

Polynomial Features:

- Squared/cubed versions of features (mostly for linear models).

Feature Selection / Dimensionality Reduction:

- Using techniques like PCA, SelectKBest, or feature importance to reduce inputs.

Domain-Specific Features:

- Custom logic based on domain knowledge, e.g., a “high risk” flag for absences > 20

Models

Logistic regression

Create and train model:

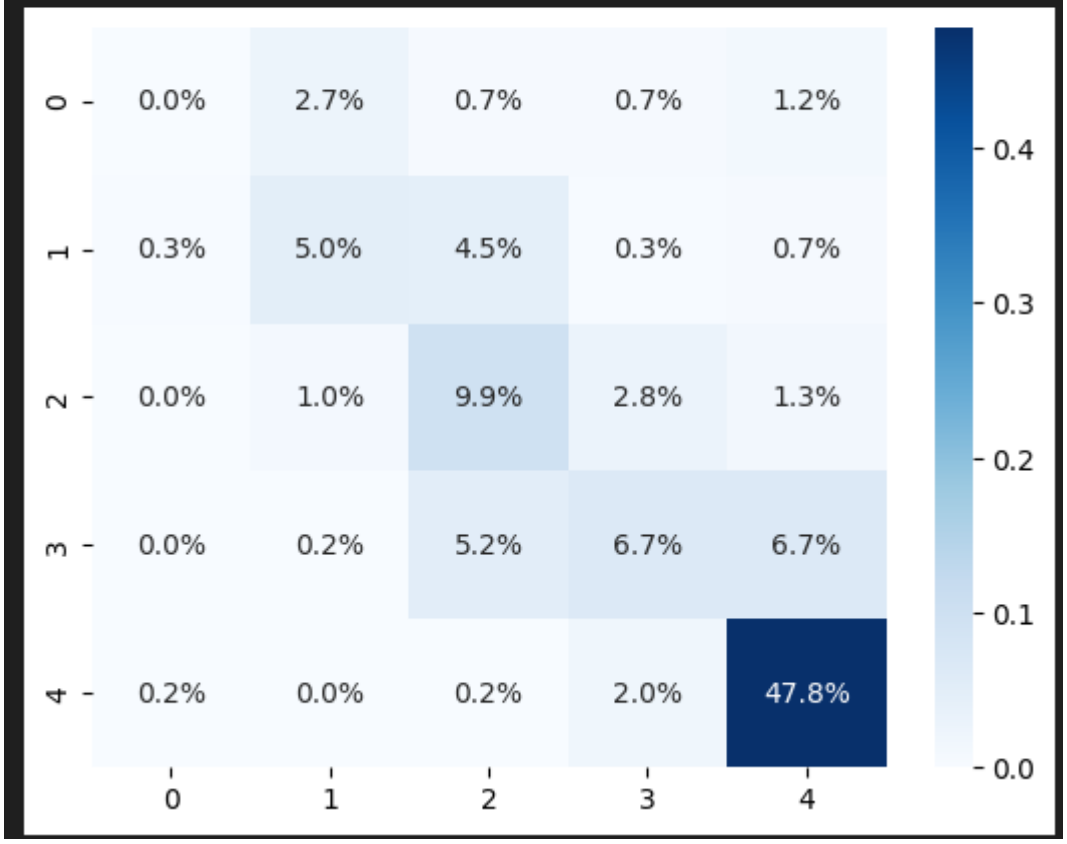
```
LogisticRegression
LogisticRegression(max_iter=2000)
```

Max_iter increases the maximum of iterations the model runs trough be for it end:

- To high and you waste resources to low and you don get a accurate prediction while the standard is 1000 iterations going for 2000 is more reasonable and it has a minimal effect on recourses.

Test Data and accuracy:

```
[[  0  16   4   4   7]
 [  2  30  27   2   4]
 [  0   6  59  17   8]
 [  0   1  31  40  40]
 [  1   0   1  12 286]]

Test Accuracy is: 69.398 %
```



Here we can see the model has an accuracy of 69% in predicting future outcomes.

Feature importance:

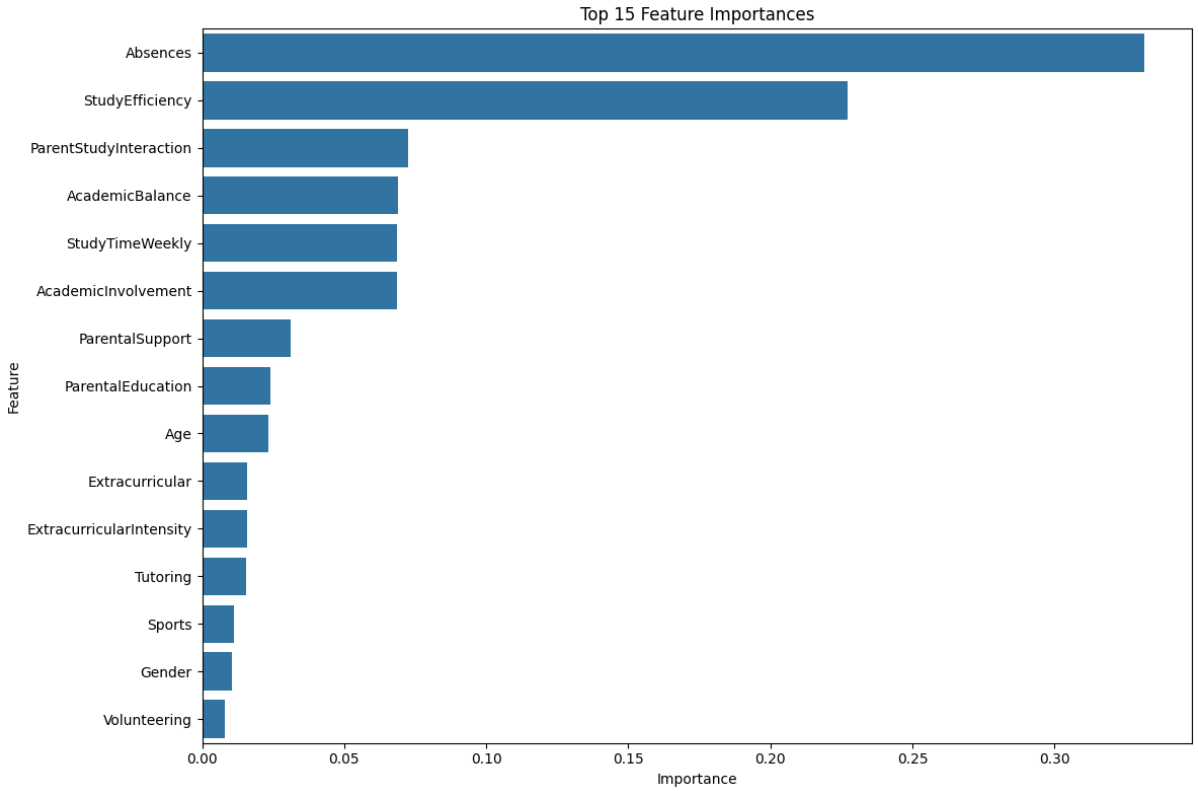
	Feature	Importance
4	Absences	0.333779
3	StudyTimeWeekly	0.036789
6	ParentalSupport	0.029766
5	Tutoring	0.027759
7	Extracurricular	0.010702
8	Sports	0.005853
2	ParentalEducation	0.004181
9	Music	0.004013

For this we can see Absences and Study Time Weekly have the strongest relationship.

Random forest

Analysing the model:

```
Best parameters: {'max_depth': 10, 'n_estimators': 200}

Accuracy: 0.7157190635451505

Classification Report:
              precision    recall  f1-score   support

         0.0       0.60      0.22      0.32        27
         1.0       0.57      0.52      0.55        67
         2.0       0.51      0.58      0.55        98
         3.0       0.50      0.56      0.53       103
         4.0       0.91      0.90      0.90       303

    accuracy                           0.72       598
   macro avg       0.62      0.56      0.57       598
weighted avg       0.72      0.72      0.71       598
```



Explanation of the results:

Best parameters explanation:

- `max_depth: 10`: Limits tree depth of each tree to 10 branches to prevent overfitting trees.
- `n_estimators: 200`: The number of trees, although this increases the execution time of the model, it does allow for a more accurate prediction.

Accuracy explanation:

- Overall accuracy is 71.57%, however the accuracy is further explored in the classification report.

Classification report explanation:

The classification report provides precision, recall, f1-score, and support for each class.

Class 0.0:

- Explanation: The model struggles to identify class 0.0 instances (indicated by the recall value of 22% and the f1-score of 32%), likely due to the low amount of data points that are present to train and test the model on (only 27 data points are available as indicated by the support). This is not a huge issue as this class is the top performance students thus a high accuracy here is not the biggest concern.

Class 1.0, 2.0 AND 3.0:

- Explanation: The model preforms relatively the same for these classes, with high recall (52% for class 1.0, 0.58 for class 2.0 and 0.56 for class 3.0) suggesting it's not reliable for accurate predictions, this is also indicated by the f1 score of 55% in class 1.0, 55% in class 2.0 and 53% in class 3.0.

Class 4.0:

- Explanation: The model preforms the best here, likely due to the large number of samples. This is also the most important class to be accurate in as this class indicates the problem students. Having a high accuracy here is very beneficial. The model has an accuracy of 90% as indicated by the f1 and recall value of 0.90

Deep learning

Build model and set up tuner:

- Using the sequential keras model
- Model: https://keras.io/api/models/sequential/
- Explained: https://www.geeksforgeeks.org/keras-sequential-class/

keras tuner:

https://keras.io/keras_tuner/api/tuners/random/

Automatically searches for the best hyperparameters for the deep learning model instead of using a grid or manual methods.

It works by:

- Randomly picks different combinations of settings.
- Trains a model with each.
- Picks the best based on a metric specified (`objective` = `'val_accuracy'`).

Settings it tries in the script:

- `units1`, `units2`: Neurons in 1st and 2nd layers (32 to 256).
- `dropout1`, `dropout2`: Dropout rates (0 to 0.5).

```
Trial 10 Complete [00h 00m 04s]
val_accuracy: 0.7467362880706787

Best val_accuracy So Far: 0.7467362880706787
Total elapsed time: 00h 00m 37s
```

Choose the best model and run predictions:

- Best model = the one that scored highest on validation accuracy during tuner search.
- Fit the best model again on full training data (20 epochs).
- Use it to predict the classes for X_test.

```
Epoch 1/20
c:\Program Files\Python311\Lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
c:\Program Files\Python311\Lib\site-packages\keras\src\saving\saving_lib.py:757: UserWarning: Skipping variable loading for optimizer 'adam', because it has 2 variables whereas the saved optimizer
  saveable.load_own_variables(weights_store.get(inner_path))
48/48 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7336 - loss: 0.8605 - val_accuracy: 0.7285 - val_loss: 0.8514
Epoch 2/20
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7555 - loss: 0.7363 - val_accuracy: 0.7337 - val_loss: 0.8003
Epoch 3/20
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7616 - loss: 0.7472 - val_accuracy: 0.7311 - val_loss: 0.7987
Epoch 4/20
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7668 - loss: 0.6612 - val_accuracy: 0.7337 - val_loss: 0.7846
Epoch 5/20
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7616 - loss: 0.7059 - val_accuracy: 0.7311 - val_loss: 0.7850
Epoch 6/20
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7908 - loss: 0.6596 - val_accuracy: 0.7337 - val_loss: 0.7789
Epoch 7/20
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7764 - loss: 0.6379 - val_accuracy: 0.7258 - val_loss: 0.7909
Epoch 8/20
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7733 - loss: 0.6396 - val_accuracy: 0.7311 - val_loss: 0.7784
Epoch 9/20
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7682 - loss: 0.7056 - val_accuracy: 0.7493 - val_loss: 0.7819
Epoch 10/20
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7881 - loss: 0.6224 - val_accuracy: 0.7285 - val_loss: 0.7739
Epoch 11/20
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7965 - loss: 0.5892 - val_accuracy: 0.7337 - val_loss: 0.7739
Epoch 12/20
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7824 - loss: 0.6197 - val_accuracy: 0.7285 - val_loss: 0.7707
Epoch 13/20
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8040 - loss: 0.5877 - val_accuracy: 0.7258 - val_loss: 0.7681
...
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7881 - loss: 0.5805 - val_accuracy: 0.7180 - val_loss: 0.8100
Epoch 20/20
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8075 - loss: 0.5519 - val_accuracy: 0.7154 - val_loss: 0.8245
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
```

Run Evaluation Metrics:

Accuracy:

- simple measure of correctness
- `correct predictions` / `total predictions`

```
Accuracy: 0.7077244258872651
```

Precision (weighted):

- How many predictions were correct. Weighted adjusts for class imbalance.

```
Precision: 0.7210215956962963
```

Recall (weighted):

- How many labels were correctly predicted? Weighted adjusts for class imbalance

```
Recall: 0.7077244258872651
```

F1 Score (weighted):

- Harmonic mean of precision and recall. Weighted adjusts for class imbalance.

```
F1 Score: 0.7087159825807737
```

Confusion Matrix:

- Shows real vs predicted class counts.

```
Confusion Matrix:
 [[  4   5   3   5   4]
 [ 12  20  16   4   2]
 [  1   5  52  19   1]
 [  0   0  23  50  10]
 [  1   3   2  24 213]]
```

Classification Report:

- Breakdown of precision, recall, F1-Score per class.

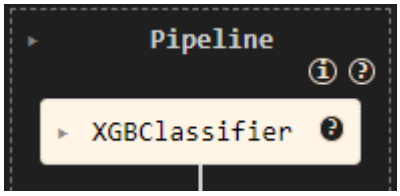
```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.22      0.19      0.21        21
         1.0       0.61      0.37      0.46        54
         2.0       0.54      0.67      0.60        78
         3.0       0.49      0.60      0.54        83
         4.0       0.93      0.88      0.90       243

    accuracy                           0.71       479
   macro avg       0.56      0.54      0.54       479
weighted avg       0.72      0.71      0.71       479
```

XG Boost

- XG Boost (eXtreme Gradient Boost) is an enhanced form of the gradient boosting machine learning model that combines weaker models (usually decision trees) together and subsequently trained in succession with each iteration focusing on correcting the errors of the previous.
- Unlike regular gradient boosting, XG Boost integrates L1 and L2 regularization and more sophisticated tree pruning (over and underfitting prevention) as well as parallel processing making this model perform well with larger datasets (highly scalable).

Training Pipeline:

- Pipelines streamline the preprocessing, model training, and evaluation.

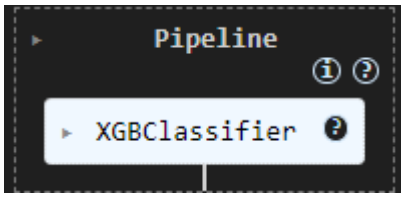


XG Boost Model Training:

```
OrderedDict([('clf__colsample_bytree', 1.0),
             ('clf__gamma', 1.168291298946482),
             ('clf__learning_rate', 0.17429148910524114),
             ('clf__max_depth', 95),
             ('clf__min_child_weight', 15),
             ('clf__subsample', 0.7340484670123454)])
```

Evaluate Model for Predictions:

- The training and test scores (based on roc_auc_ovr metrics) are promising as both are close to the value of 1 and are close to each other.
- The convergence plot shows gradual improvement through each evaluation until a stable solution was found (plateau).



Shows the pipeline instance with best hyper Param combinations.

Shows best CV (cross-validation) score:

```
np.float64(0.8652953895649971)
```

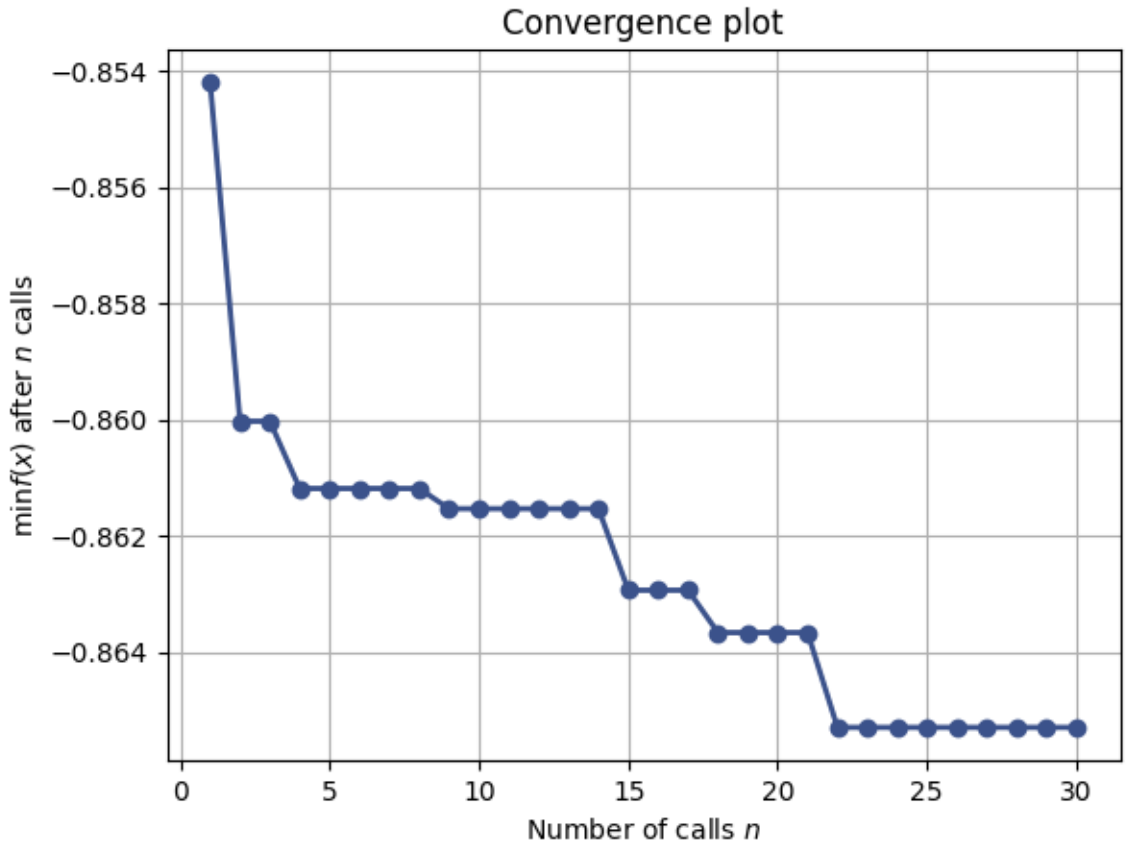
Shows best CV (cross-validation) score from roc_auc_ovr during training.

Shows final models score on test data:

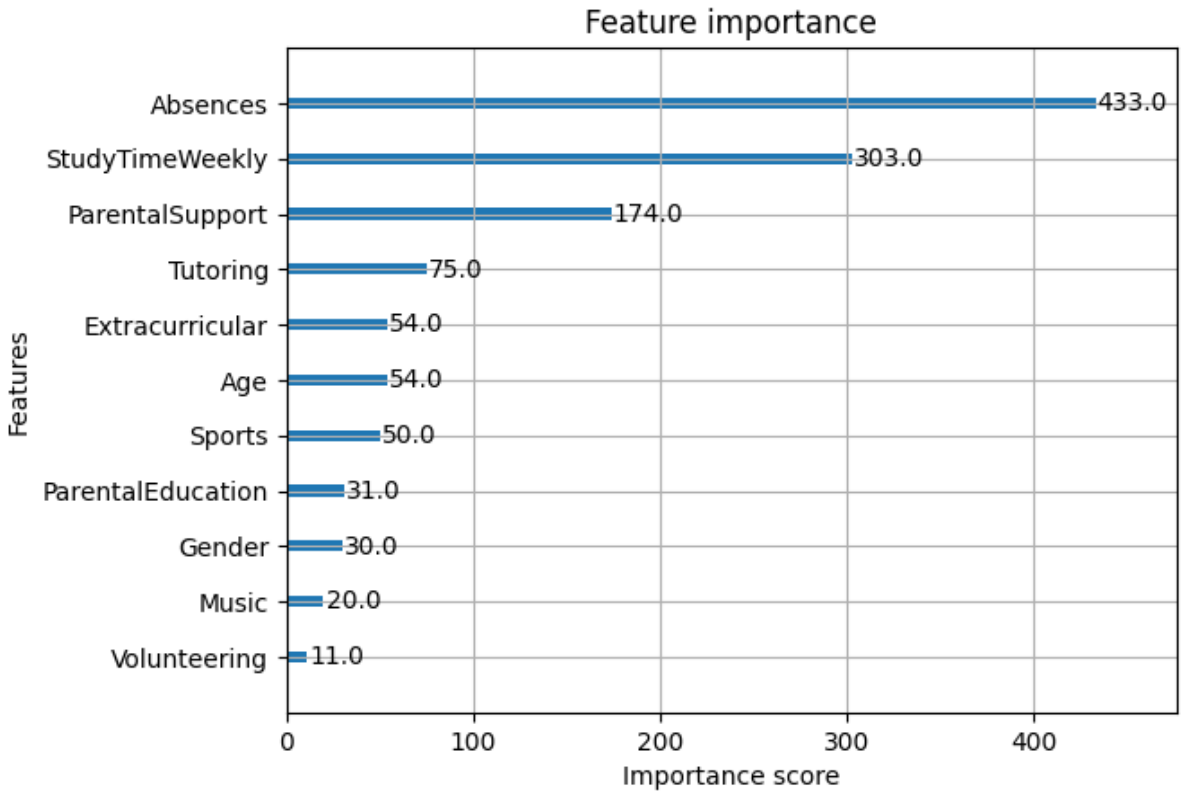
```
np.float64(0.900316781235748)
```

Shows final models score on test data.

Convergence plot based on Bayes Search optimum results:

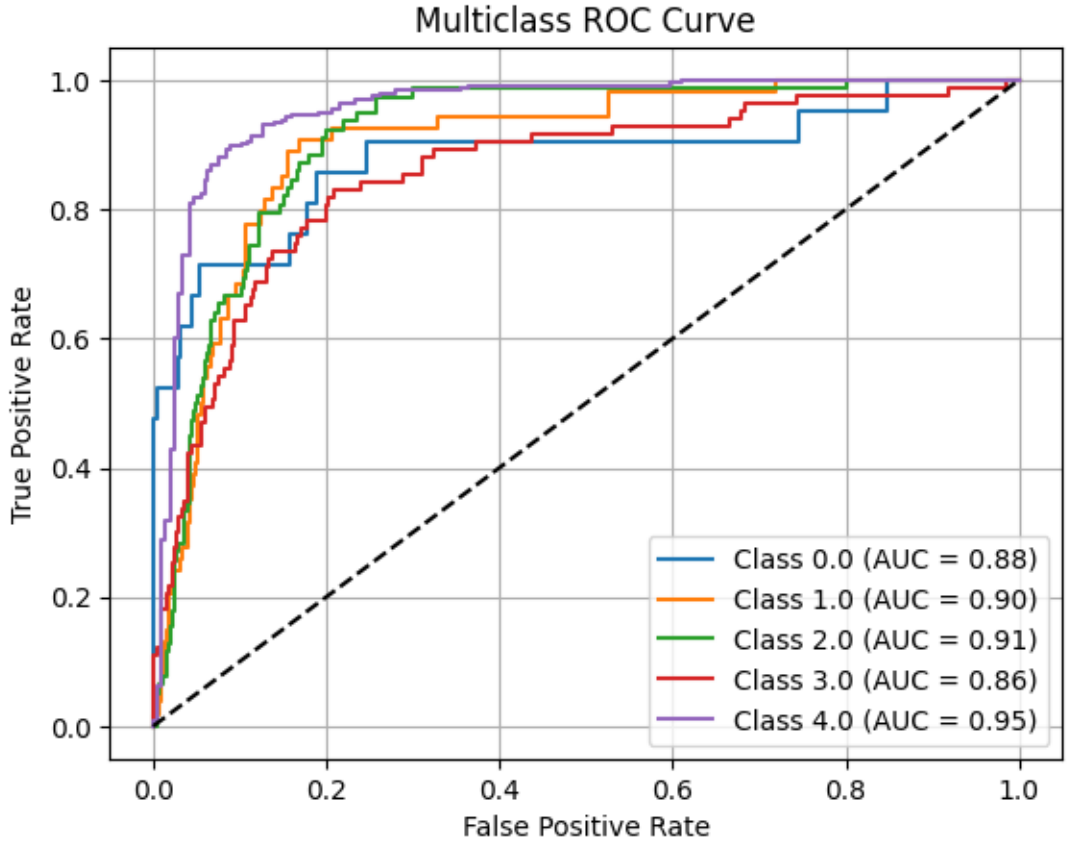


Measuring Feature Performance:



ROC Curves:

- A good model is represented by values in the top left portion (True Positive Rate (TPR)) with each AUC (Area Under Curve) being close to 1.0.



Link to GitHub repository

https://github.com/BeardedSeal77/MLG382_Guided_Project

Link to Dash app

https://mlg382-guided-project-1.onrender.com/

Challenges faced

Dash app was difficult to launch (All). Pushing to GitHub for VS code (Brandon).

Lessons Learned

Team building, the importance of communicating effectively and asking for help if needed.