Assignment Report

Student Dropout and Academic Success Prediction using Machine Learning

Module Code:
Convenor Name:
Student Name:
Student Number:
Date: 20 Feb, 2024

Actual Hours Spent: 40+ hrs (I work in 2 hour sessions, and take 5 min break every 30 mins. I needed 19 such sessions, and some extra work in between.)

Abstract

This report presents a predictive comprehensive analysis of student dropouts and academic success in higher education institutions via machine learning (ML). Utilizing a dataset from the UCI Machine Learning Repository by *Realinho et al. (2021)*, I applied three distinct machine learning models: Logistic Regression, Random Forest Classifier, and a Deep Learning model implemented with TensorFlow. The study aims to proactively identify students at risk of dropping out, allowing for timely intervention strategies.

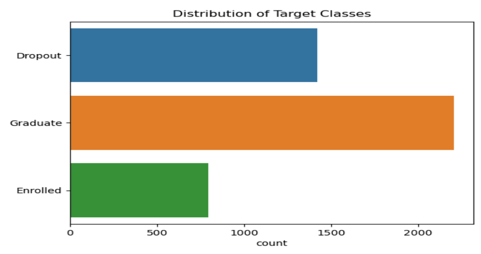
Background and Problem Addressed

The challenge in the educational sector to predict student dropouts and successes is critical. Proactive identification of at-risk students can lead to timely interventions, improving educational outcomes. This insight is not just a matter of statistical interest but a tool for informed decision-making, policy formulation, and resource allocation for the university authorities and student counselors. By identifying students at risk of dropping out, institutions can come up with strategies, allocate support resources more effectively, and ultimately foster a more inclusive and successful educational environment.

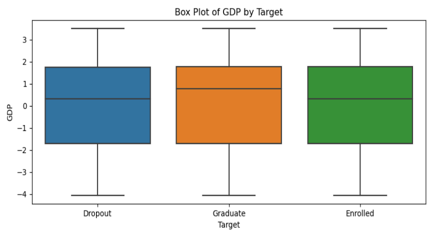
This study leverages machine learning technologies to analyze multidimensional factors (demographic, academic, and socio-economic) to address these challenges, contributing to the burgeoning field of educational data mining. I hope that my model can help universities make better decisions so students can have the best possible chances of getting an education.

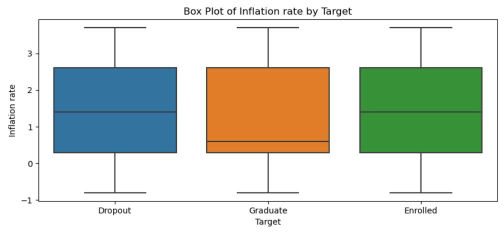
Exploratory Data Analysis

The dataset comprises **4424 instances with 36 features**, including demographic, academic, and socio-economic factors. Initial explorations reveal a balanced representation of features like Marital Status, Gender, and others, with insights gained from distribution plots and correlation analyses.

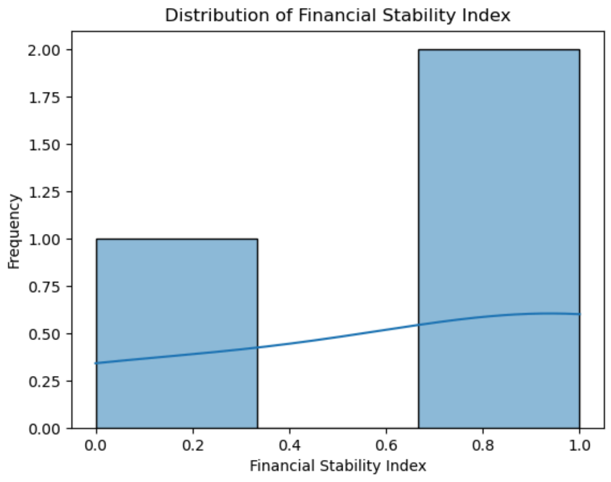


This figure shows the imbalanced nature of our dataset, and as we can see, the Enrolled class has the least example, and our models have a harder time correctly finding this class.



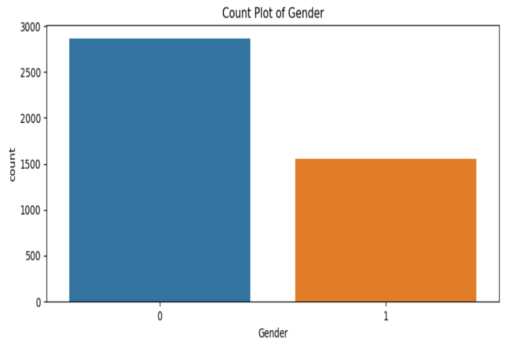


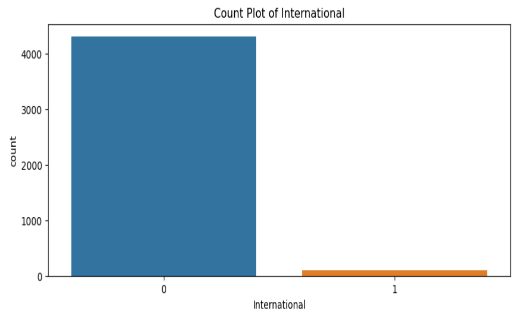
These figures show that current economic factors (GDP, Inflation) affect students' decisions. Then, we developed a feature that condenses these factors.We called it the *Financial*

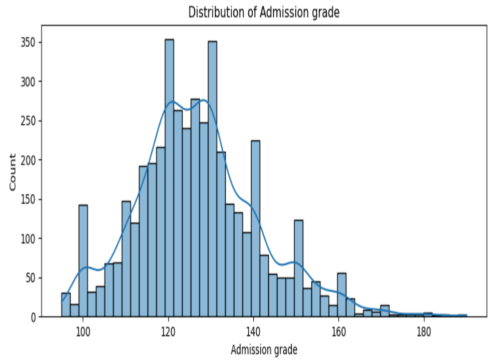


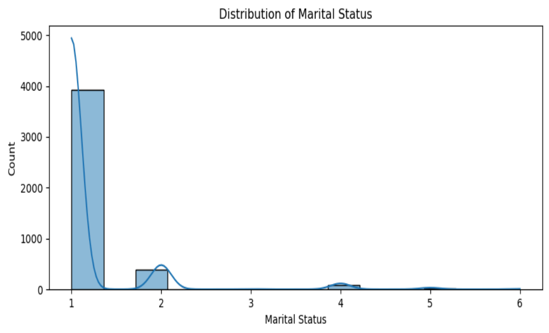
Stability Index.

These figures show the imbalance of gender, international/national students, marital status and the admission grades. ***Lenis, Wong. (2023), and Ioanna et al. (2009)***

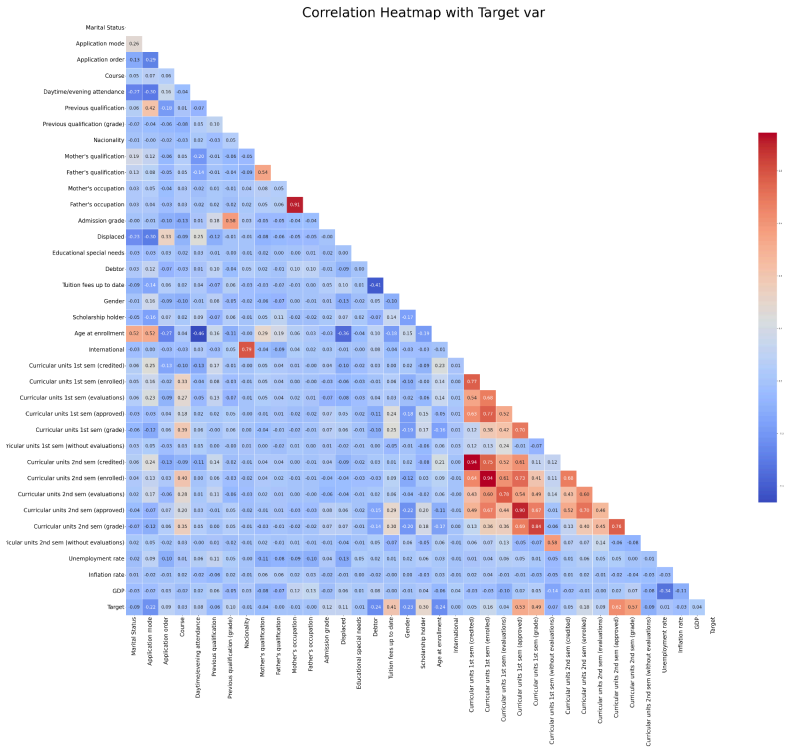








The Heatmap Below shows which factors are Correlated to the target.



Data Pre-processing and Feature Selection

- The first step was Imputation. Although our data has no missing values, I applied a mean and a knn imputer from sklearn.impute. Since we're fetching the data from *ucimlrepo*, if they update the data, and if missing features come up - our models would still be able to work with it.

- Our initial EDA hinted at the presence of outliers, indicated by the dots lying outside the box plots and unusual statistics. To address this, for the second step in our data preprocessing, I utilized Z-scores to detect and handle these outliers. I highlighted data points where the Z-scores were above 3 (corresponds to the 99.7th percentile in a standard normal distribution). After identifying these outliers, I implemented a capping method to mitigate their impact. The extreme values were replaced by a threshold (Q1 - 1.5 * IQR, and Q1 + 1.5 * IQR). The capping ensures that outliers are not removed entirely from the dataset, but brought to a standard level to minimize their potentially distorting effects on the analysis, while still retaining the overall data distribution and structure.

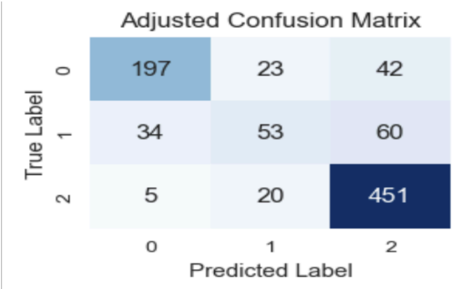
- Then, I transformed all categorical variables using one-hot encoding to convert them into a format that machine learning models can interpret more effectively. For continuous variables, I normalized to bring all variables to a common scale, eliminating any bias that might arise from varying scales.

- Feature Engineering : There were 36 features, and some were correlated to each other. So, instead of letting the model figure out the overlapping pattern, I introduced some new features such as 'Approval Rate' and 'Financial Stability Index'. The details are in my jupyter notebook. They were improved over time as I built the models.

Machine learning models

Model 1: Logistic Regression

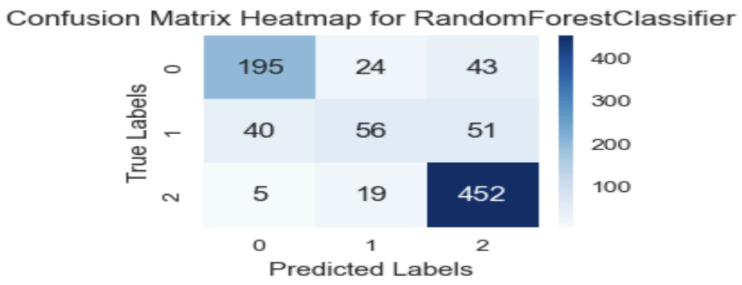
- **Summary:** It was the easiest model to build, so it was my first choice as a baseline model. To my surprise, it did pretty well compared to some other complex models.
- **Parameters:** max_iter=500.
- **Rationale for using the Model:** Simplicity and Interpretability.
- **Training and Evaluation:** Achieved an accuracy of 79.02%, with a weighted precision of 77.2%.
- **Analysis:** Overall balanced performance but slightly lower compared to other 2 models.



```
Adjusted Accuracy :  0.792090395480226
Precision :  0.7774713801989583
Recall :  0.792090395480226
F1-Score :  0.7781472294154175
ROC-AUC Score: 0.8948133272631641
Classification Report :
              precision    recall  f1-score   support

     Dropout       0.83      0.75      0.79       262
    Enrolled       0.55      0.36      0.44       147
    Graduate       0.82      0.95      0.88       476

    accuracy                           0.79       885
   macro avg       0.73      0.69      0.70       885
weighted avg       0.78      0.79      0.78       885
```

Model 2: Random Forest Classifier

- **Summary:** I made a Decision Trees model, and although very easy to build, it showed promising results, so I finally did a RFC model for its robustness against overfitting and ability to capture nonlinear relationships. Results similar to Logistic Regression indicate that the data aren't nonlinear in nature. *Leo, Breiman. (2001).*
- **Parameters:** Used default parameters.
- **Rationale for using the Model:** Robustness against overfitting, nonlinear pattern capturing capabilities.
- **Training and Evaluation:** Achieved the highest ROC-AUC score of 0.8964, indicating superior class differentiation. Accuracy 80%, precision 77.9%
- **Analysis:** The best overall performance. Some details here -

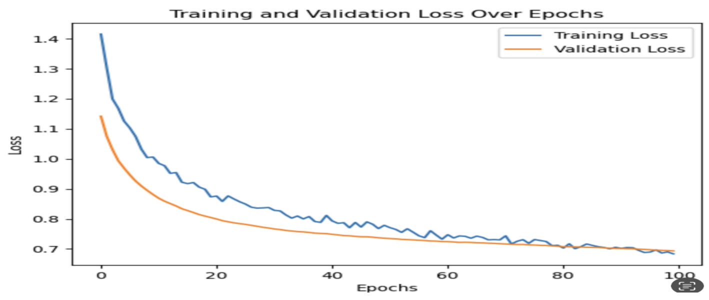


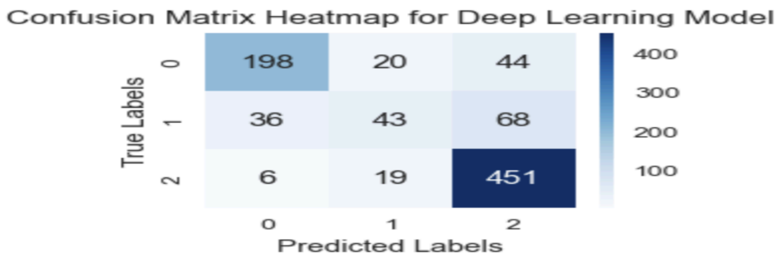
```
Random Forest Accuracy: 0.7943502824858757
Random Forest Precision: 0.7797489233929912
Random Forest Recall: 0.7943502824858757
Random Forest F1-Score: 0.7813714597647134
Random Forest ROC-AUC Score: 0.8964698222285884
Random Forest Classification Report:
              precision    recall  f1-score   support

     Dropout       0.81      0.74      0.78       262
    Enrolled       0.57      0.38      0.46       147
    Graduate       0.83      0.95      0.88       476

    accuracy                           0.79       885
   macro avg       0.74      0.69      0.71       885
weighted avg       0.78      0.79      0.78       885
```

Model 3: Deep Learning with TensorFlow

- **Summary:** The model's architecture, focused on scalability, incorporates a sequential layout with multiple layers, each designed to progressively refine the learning from the data.
  - Structure : A Sequential model is employed, a common choice for a stack of layers where each layer has exactly one input tensor and one output tensor.
  - Layers : Dense layers form the core of this model. These layers are fully connected, meaning each neuron in a layer receives input from all neurons of the previous layer, enhancing the model's ability to learn complex patterns.

    - Activation Function : I've used the very common 'relu' (rectified linear unit) activation function. It is known for its efficiency and effectiveness and helps in mitigating the vanishing gradient problem, which is crucial for DL.
    - Regularization : I've applied L2 regularization, aiming to prevent overfitting by penalizing large weights. This approach helps the model to learn smaller weights, leading to simpler models that generalize better.
    - Dropout Rate This is to stop overfitting. While training, half of the neurons in a layer are randomly deactivated (dropout rate of 0.5 implies that)
- **Parameters:** Sequential model with Dense layers, 'relu' activation, L2 regularization, and dropout rate of 0.3, Learning rate = 0.0001
- **Rationale for using the Model:** Adopted for its capability to model complex relationships.
- **Training and Evaluation:** The model has shown performance comparable to a Random Forest model, an encouraging sign given Random Forest's reputation for robustness. Accuracy 78.2%, and a ROC-AUC of 0.897 is notable. This metric indicates good discriminatory ability.
- **Analysis:** I believe that with some proper tuning, maybe a different architecture, or hyperparameter readjustment could help DL models outperform the other ones. Here are the current results that I could push the model to (so far) -





```
Deep Learning Model Accuracy: 0.7819209039548023
Deep Learning Model Precision: 0.7621950827901571
Deep Learning Model Recall: 0.7819209039548023
Deep Learning Model F1-Score: 0.7628455758968234
Deep Learning Model ROC-AUC Score: 0.8960685484000672
Deep Learning Model Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.76      0.79       262
           1       0.52      0.29      0.38       147
           2       0.80      0.95      0.87       476

    accuracy                           0.78       885
   macro avg       0.72      0.67      0.68       885
weighted avg       0.76      0.78      0.76       885
```

Performance Measures and Evaluation Strategies

I've embraced a multifaceted approach to evaluate their performance. One of my go-to tools is the **ROC-AUC Score**, which brilliantly showcases a model's knack for distinguishing between various classes. To get a well-rounded view, I also lean on **Accuracy**, **Precision**, **Recall**, and the **F1-Score**. They provide a comprehensive and balanced view of how well my models are performing. Then there's the **Confusion Matrix** which offers insights into the classification accuracy across different categories. I've used lime and yellowbook, you can see it in the notebook or in the appendix section below.

Model Limitations

- Potential for Bias : If the data reflects historical biases, the models may inadvertently perpetuate these. For instance, if certain demographic groups historically had higher dropout rates, the model might unfairly flag students from these groups as at-risk.

- Data Quality : Any missing nuances or unrecorded variables that significantly impact student success could skew predictions.

- While the models excel in identifying patterns within the dataset, they cannot ascertain causation, an important consideration when interpreting the results.

Results Comparison and Analysis

For this classification problem at hand, I crafted 8 models, each with its own pros and cons. I zeroed in on the top 3 for a more detailed study, finding that these models had a special knack for handling unseen test data. It was a close call, but the **Random Forest Classification** stood out, not just in ROC-AUC and accuracy but across various metrics.

Deep Learning, with its complex neural networks, hinted at a vast potential yet to be fully tapped. **Logistic Regression**, though a bit more modest in performance, still held its ground with admirable effectiveness. These 3 models were very close to each other. Check the confusion matrices.

The other models I haven't discussed here - **Gradient Boosting Machines (GBM)** and **Support Vector Machines (SVM)** were powerful and accurate. **The Naive Bayes Classifier**, with its simplicity, was surprisingly quick and accurate. **K-Nearest Neighbors (KNN)** and **Decision Tree**s were reliable and straightforward. All of these models show accuracy near **65-78%**, and ROC-AUC scores between **0.78** to **0.87**.

However, every single model lacks in identifying 'Enrolled' classes. I knew this would be the case given the imbalanced nature of the dataset, but, even after working and building so many different models and different feature engineering, it couldn't be improved much. I tried applying ADASYN to oversample this minority class with synthetic samples to balance the dataset and to improve classification accuracy. Also tried SMOTE, which helps a bit but comes with other costs. Ultimately, those parts of the code were commented out.

I tried to grid search for the best hyperparameters for better metrics, and tuned the model accordingly. These parts were added to the last portion of the notebook.

Conclusion, Recommendations, and Future Work

Throughout this assignment, I've worked with lots of ML models for multiclass classification techniques and it's clear that ML is a powerful tool for predicting student outcomes in educational settings. Each model brought something unique to the table, with the Random Forest Classifier emerging as the most balanced one. For future works -

- **Further Feature Engineering:** Looking ahead, I'm excited to do an in-depth Analysis of feature impacts and model behaviors. We need better Feature Engineering to give the models a sharper edge.
- **Over/Undersampling for Class Imbalance:** All of our models suffered in predicting the 'Enrolled' category. As expected from an imbalance dataset *Tahira et al. (2019)*. I have tried weighted variations, but that didn't work very well for me. I also tried sampling techniques ADASYN (Adaptive Synthetic Sampling) and SMOTE (Synthetic Minority Over-sampling Technique) He et al. (2008), but I'd like to experiment more with Sampling.
- **Model Tuning and Ensemble Methods:** A mix of hyperparameter tuning and Ensemble Methods could be the secret for even better performance.
- **Advanced Model:** Something like Long Short-Term Memory (LSTM) networks could adeptly handle the sequential nature of academic data. *Hochreiter et al. (1997).* Also plan to implement a one-vs-rest model to address the problems.
- **Cross-Validation Strategy:** To strengthen the robustness and interpretability of the models.

Ethical Considerations: It's always important to consider the ethical implications of predictive modeling. I have to ensure fairness and make sure that my model avoids biases that could adversely affect certain student groups before they are put to use in the real world.

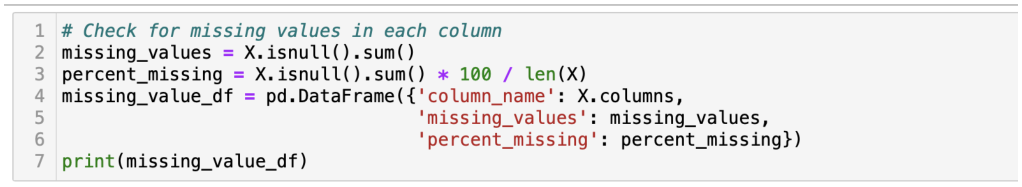
Practical Implications : We can develop targeted support systems for students at risk, optimize our resources, and cater our educational policies to enhance student retention and success. With insights from these models, institutions can improve their academic performances and also foster a more supportive and equitable learning system.

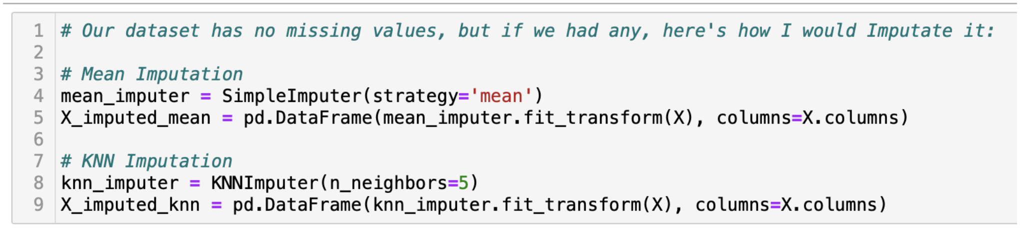
References

1. Realinho,Valentim, Vieira Martins,Mónica, Machado,Jorge, and Baptista,Luís. (2021). Predict students' dropout and academic success. UCI Machine Learning Repository. https://doi.org/10.24432/C5MC89.
2. Leo, Breiman. (2001). Random Forests. 45(1):5-32. doi: 10.1023/A:1010933404324
3. Lenis, Wong. (2023). Model for the Prediction of Dropout in Higher Education in Peru applying Machine Learning Algorithms: Random Forest, Decision Tree, Neural Network and Support Vector Machine. 116-124. doi: 10.23919/FRUCT58615.2023.10143068
4. Ioanna, Lykourentzou., Ioannis, Giannoukos., Vassilis, Nikolopoulos., George, Mpardis., Vassili, Loumos. (2009). Dropout prediction in e-learning courses through the combination of machine learning techniques. Computer Education, 53(3):950-965. doi: 10.1016/J.COMPEDU.2009.05.010
5. Hochreiter, Sepp & Schmidhuber, Jürgen. (1997). Long Short-term Memory. Neural computation. 9. 1735-80. 10.1162/neco.1997.9.8.1735.
6. Tahira, Alam., Chowdhury, Farhan, Ahmed., Sabit, Anwar, Zahin., Muhammad, Asif, Hossain, Khan., Maliha, Tashfia, Islam. (2019). An Effective Recursive Technique for Multi-Class Classification and Regression for Imbalanced Data. IEEE Access, 7:127615-127630. doi: 10.1109/ACCESS.2019.2939755
7. He, Haibo & Bai, Yang & Garcia, Edwardo & Li, Shutao. (2008). ADASYN: Adaptive Synthetic Sampling Approach for Imbalanced Learning. Proceedings of the International Joint Conference on Neural Networks. 1322 - 1328. 10.1109/IJCNN.2008.4633969.

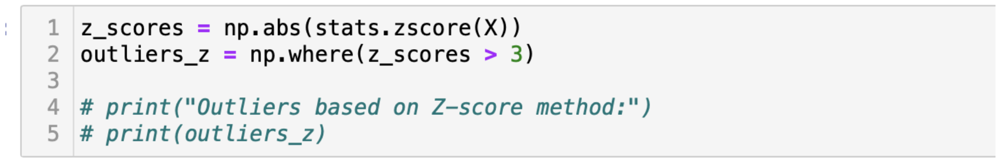
Appendix

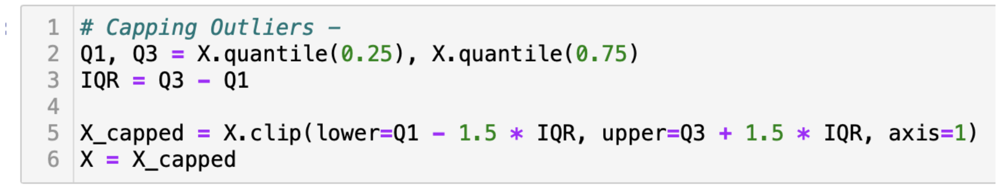
Imputation Strategy Code :


```
# Check for missing values in each column
missing_values = X.isnull().sum()
percent_missing = X.isnull().sum() * 100 / len(X)
missing_value_df = pd.DataFrame({'column_name': X.columns,
                                 'missing_values': missing_values,
                                 'percent_missing': percent_missing})
print(missing_value_df)
```

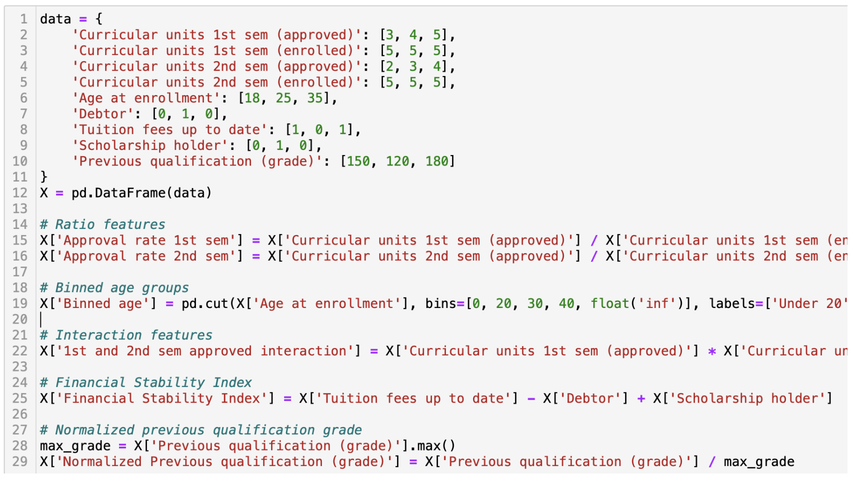

```
# Our dataset has no missing values, but if we had any, here's how I would Imputate it:

# Mean Imputation
mean_imputer = SimpleImputer(strategy='mean')
X_imputed_mean = pd.DataFrame(mean_imputer.fit_transform(X), columns=X.columns)

# KNN Imputation
knn_imputer = KNNImputer(n_neighbors=5)
X_imputed_knn = pd.DataFrame(knn_imputer.fit_transform(X), columns=X.columns)
```

Outlier Capping Code:

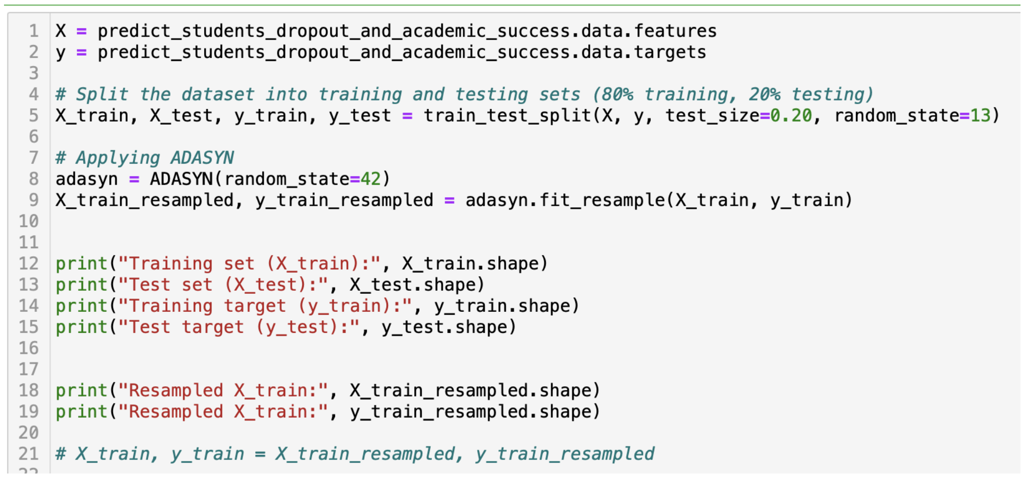

```
z_scores = np.abs(stats.zscore(X))
outliers_z = np.where(z_scores > 3)

# print("Outliers based on Z-score method:")
# print(outliers_z)
```


```
# Capping Outliers -
Q1, Q3 = X.quantile(0.25), X.quantile(0.75)
IQR = Q3 - Q1

X_capped = X.clip(lower=Q1 - 1.5 * IQR, upper=Q3 + 1.5 * IQR, axis=1)
X = X_capped
```

Feature Engineering Code Overview :

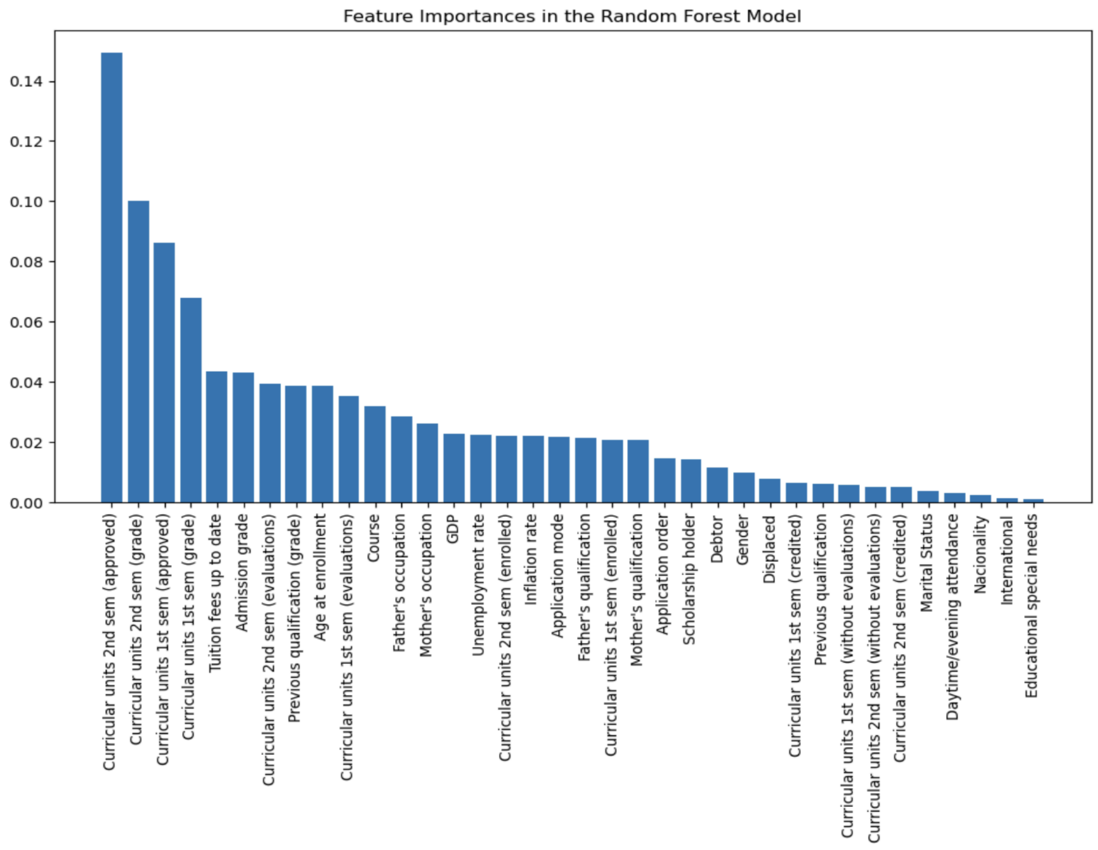

```
data = {
    'Curricular units 1st sem (approved)': [3, 4, 5],
    'Curricular units 1st sem (enrolled)': [5, 5, 5],
    'Curricular units 2nd sem (approved)': [2, 3, 4],
    'Curricular units 2nd sem (enrolled)': [5, 5, 5],
    'Age at enrollment': [18, 25, 35],
    'Debtor': [0, 1, 0],
    'Tuition fees up to date': [1, 0, 1],
    'Scholarship holder': [0, 1, 0],
    'Previous qualification (grade)': [150, 120, 180]
}
X = pd.DataFrame(data)

# Ratio features
X['Approval rate 1st sem'] = X['Curricular units 1st sem (approved)'] / X['Curricular units 1st sem (er
X['Approval rate 2nd sem'] = X['Curricular units 2nd sem (approved)'] / X['Curricular units 2nd sem (er

# Binned age groups
X['Binned age'] = pd.cut(X['Age at enrollment'], bins=[0, 20, 30, 40, float('inf')], labels=['Under 20'

# Interaction features
X['1st and 2nd sem approved interaction'] = X['Curricular units 1st sem (approved)'] * X['Curricular ur

# Financial Stability Index
X['Financial Stability Index'] = X['Tuition fees up to date'] - X['Debtor'] + X['Scholarship holder']

# Normalized previous qualification grade
max_grade = X['Previous qualification (grade)'].max()
X['Normalized Previous qualification (grade)'] = X['Previous qualification (grade)'] / max_grade
```

Over-Sampling Technique :

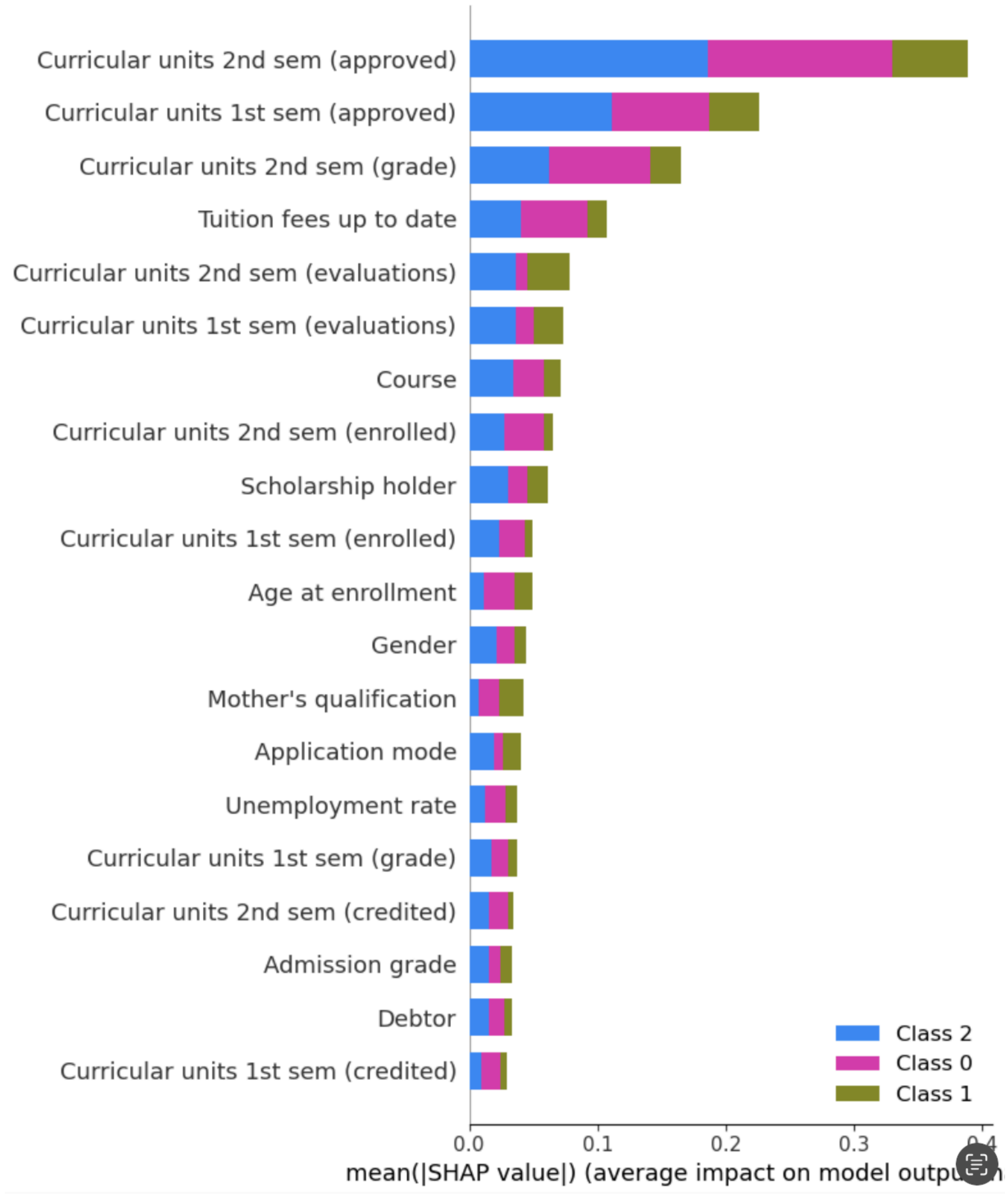


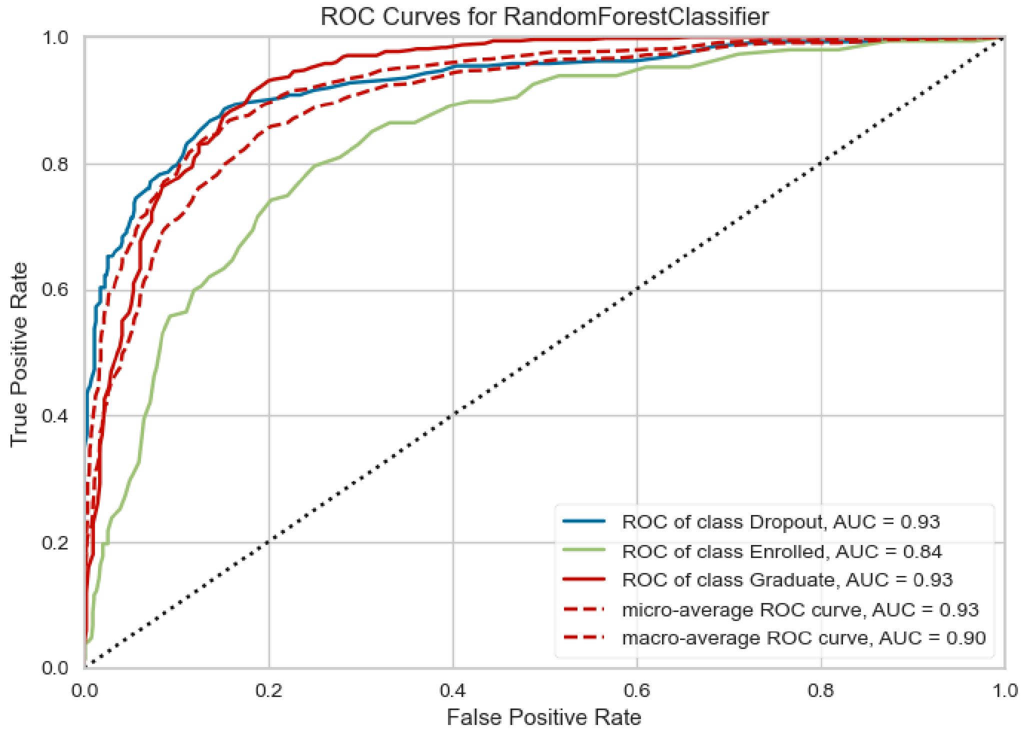
```python
X = predict_students_dropout_and_academic_success.data.features
y = predict_students_dropout_and_academic_success.data.targets

# Split the dataset into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=13)

# Applying ADASYN
adasyn = ADASYN(random_state=42)
X_train_resampled, y_train_resampled = adasyn.fit_resample(X_train, y_train)


print("Training set (X_train):", X_train.shape)
print("Test set (X_test):", X_test.shape)
print("Training target (y_train):", y_train.shape)
print("Test target (y_test):", y_test.shape)


print("Resampled X_train:", X_train_resampled.shape)
print("Resampled X_train:", y_train_resampled.shape)

# X_train, y_train = X_train_resampled, y_train_resampled
```

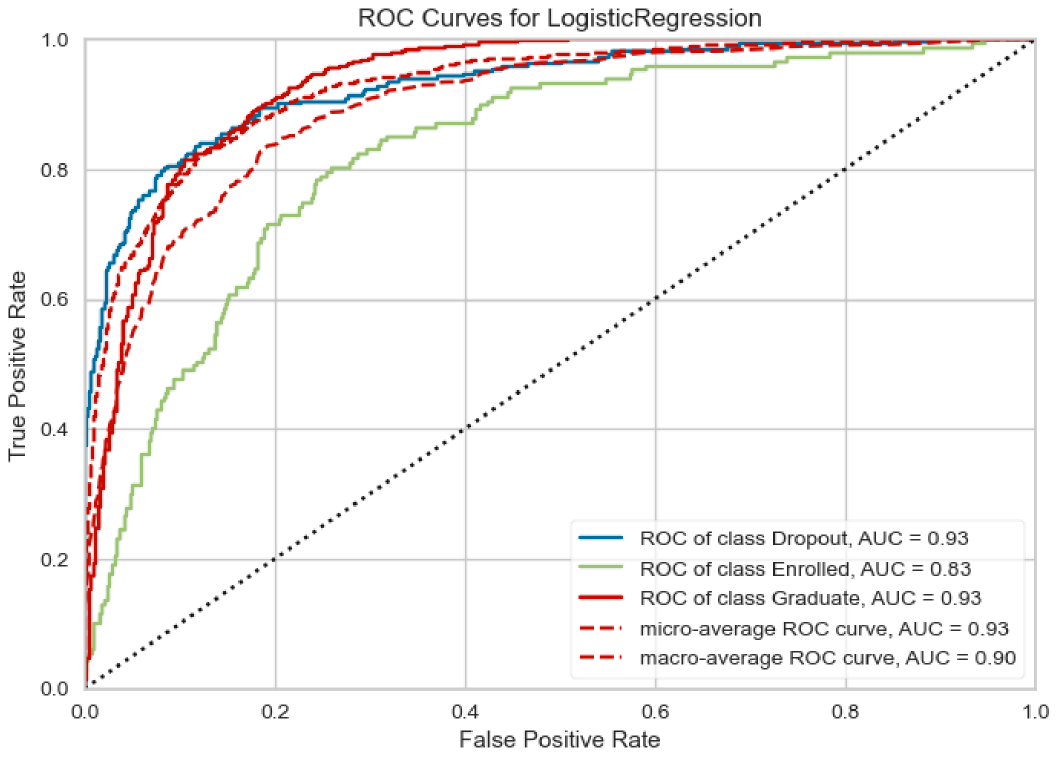
Feature Importance Analysis :

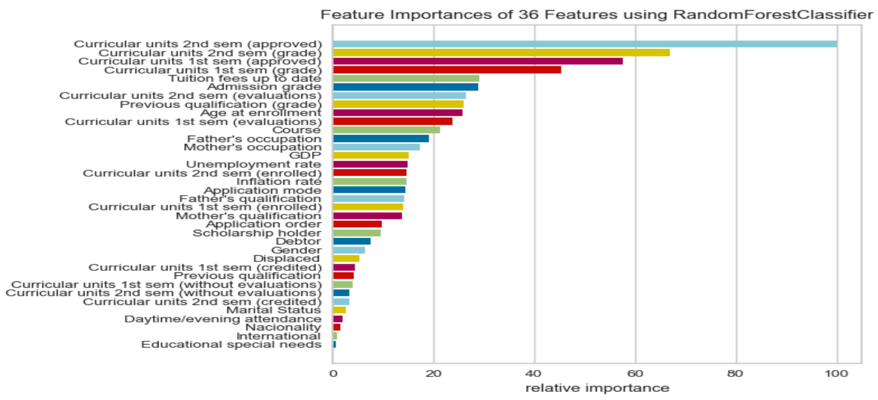


SHapley Additive exPlanations (SHAP) to interpret the DL model's predictions :




ROC Curves for RandomForestClassifier
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC of class Dropout, AUC = 0.93
ROC of class Enrolled, AUC = 0.84
ROC of class Graduate, AUC = 0.93
micro-average ROC curve, AUC = 0.93
macro-average ROC curve, AUC = 0.90


ROC Curves for LogisticRegression
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC of class Dropout, AUC = 0.93
ROC of class Enrolled, AUC = 0.83
ROC of class Graduate, AUC = 0.93
micro-average ROC curve, AUC = 0.93
macro-average ROC curve, AUC = 0.90



Grid Search to Improve DL model's Hyperparameters

```python
def build_model(hp):
    model = keras.Sequential()
    model.add(keras.layers.InputLayer(input_shape=(X_train_scaled.shape[1],)))

    # Tuning the number of layers and units in each layer
    for i in range(hp.Int('num_layers', 2, 5)):
        model.add(keras.layers.Dense(units=hp.Int('units_' + str(i), min_value=32, max_value=512, step=32),
                                     activation=hp.Choice('activation_' + str(i), ['relu', 'tanh', 'sigmoid'])))
        model.add(keras.layers.Dropout(rate=hp.Float('dropout_' + str(i), min_value=0.0, max_value=0.5, step=0.1

    model.add(keras.layers.Dense(y_train_categorical.shape[1], activation='softmax'))

    # Tuning the learning rate
    model.compile(optimizer=keras.optimizers.legacy.Adam(hp.Float('learning_rate', 1e-4, 1e-2, sampling='log')),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model

tuner = RandomSearch(
    build_model,
    objective='val_accuracy',
    max_trials=10,  # Number of trials to run
    executions_per_trial=1,  # Number of models to build and fit for each trial
    directory='model_tuning',
    project_name='deep_learning_tuning'
)

tuner.search(X_train_scaled, y_train_categorical, epochs=10, validation_split=0.2)
best_model = tuner.get_best_models(num_models=1)[0]
best_hyperparameters = tuner.get_best_hyperparameters()[0]
print("Best hyperparameters:")
for hp, value in best_hyperparameters.values.items():
    print(f"{hp}: {value}")


print("Best model summary:")
best_model.summary()

history = best_model.fit(X_train_scaled, y_train_categorical, epochs=100, validation_split=0.2)
```

```
Reloading Tuner from model_tuning/deep_learning_tuning/tuner0.json
Best hyperparameters:
num_layers: 5
units_0: 288
activation_0: tanh
dropout_0: 0.30000000000000004
```