Introduction to Data Science Project

Predict Student's dropout and academic success

Project report submitted By

Group-9

Tushar Sharma – 21UCC104

Rajdeep Tiwari – 21UCS166

Sahil Sidana – 21UCS177

Shreyansh Jain – 21UCS198

Course Instructors

Dr. Lal Upendra Pratap Singh

Dr. Subrat Kumar Dash

Dr. Aloke Datta

Contents

Google colab Link: IDS_Project

1. Objective

Our problem statement is to use the given Academic Background of a student and to predict the academic dropout and failure in higher education, by using machine learning techniques to identify students at risk at an early stage of their academic path, so that strategies to support them can be put into place.

2. Dataset Description

Source of Dataset: Predict students' dropout and academic success

A dataset created from a higher education institution (acquired from several disjoint databases) related to students enrolled in different undergraduate degrees, such as agronomy, design, education, nursing, journalism, management, social service, and technologies. The dataset includes information known at the time of student enrollment (academic path, demographics, and social-economic factors) and the students' academic performance at the end of the first and second semesters. The data is used to build classification models to predict students' dropout and academic success. The problem is formulated as a three category classification task (dropout, enrolled, and graduate) at the end of the normal duration of the course.

```
# Show first five rows.
data.head()
```

	Marital Status	Application mode	Application order	Course	Daytime/evening attendance	Previous qualification	Previous qualification (grade)	Nacionality	Mother's qualification	Father's qualification	Mother's occupation	Father's occupation	Admiss gr
0	1	17	5	171	1	1	122.0	1	19	12	5	9	1
1	1	15	1	9254	1	1	160.0	1	1	3	3	3	1
2	1	1	5	9070	1	1	122.0	1	37	37	9	9	1
3	1	17	2	9773	1	1	122.0	1	38	37	5	3	1
4	2	39	1	8014	0	1	100.0	1	37	38	9	9	1

```
[ ] # Show last five rows.
data.tail()
```

	Marital Status	Application mode	Application order	Course	Daytime/evening attendance	Previous qualification	Previous qualification (grade)	Nacionality	Mother's qualification	Father's qualification	Mother's occupation	Father's occupation	Adm
4419	1	1	6	9773	1	1	125.0	1	1	1	5	4	
4420	1	1	2	9773	1	1	120.0	105	1	1	9	9	
4421	1	1	1	9500	1	1	154.0	1	37	37	9	9	
4422	1	1	1	9147	1	1	180.0	1	37	37	7	4	
4423	1	10	1	9773	1	1	152.0	22	38	37	5	9	

```
data.describe()
```

	Marital Status	Application mode	Application order	Course	Daytime/evening attendance	Previous qualification	Previous qualification (grade)	Nacionality	Mother's qualification	Father's qualification	Mother's occupation	Fa occu
count	4424.000000	4424.000000	4424.000000	4424.000000	4424.000000	4424.000000	4424.000000	4424.000000	4424.000000	4424.000000	4424.000000	4424.
mean	1.178571	18.669078	1.727848	8856.642631	0.890823	4.577758	132.613314	1.873192	19.561935	22.275316	10.960895	11.
std	0.605747	17.484682	1.313793	2063.566416	0.311897	10.216592	13.188332	6.914514	15.603186	15.343108	26.418253	25.
min	1.000000	1.000000	0.000000	33.000000	0.000000	1.000000	95.000000	1.000000	1.000000	1.000000	0.000000	0.
25%	1.000000	1.000000	1.000000	9085.000000	1.000000	1.000000	125.000000	1.000000	2.000000	3.000000	4.000000	4.
50%	1.000000	17.000000	1.000000	9238.000000	1.000000	1.000000	133.100000	1.000000	19.000000	19.000000	5.000000	7.
75%	1.000000	39.000000	2.000000	9556.000000	1.000000	1.000000	140.000000	1.000000	37.000000	37.000000	9.000000	9.
max	6.000000	57.000000	9.000000	9991.000000	1.000000	43.000000	190.000000	109.000000	44.000000	44.000000	194.000000	195.

It contains 4424 rows and 37 columns/features

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4424 entries, 0 to 4423
Data columns (total 37 columns):
 #   Column                                          Non-Null Count  Dtype
---  ------                                          --------------  -----
 0   Marital Status                                  4424 non-null   int64
 1   Application mode                                4424 non-null   int64
 2   Application order                               4424 non-null   int64
 3   Course                                          4424 non-null   int64
 4   Daytime/evening attendance                      4424 non-null   int64
 5   Previous qualification                          4424 non-null   int64
 6   Previous qualification (grade)                  4424 non-null   float64
 7   Nacionality                                     4424 non-null   int64
 8   Mother's qualification                          4424 non-null   int64
 9   Father's qualification                          4424 non-null   int64
 10  Mother's occupation                             4424 non-null   int64
 11  Father's occupation                             4424 non-null   int64
 12  Admission grade                                 4424 non-null   float64
 13  Displaced                                       4424 non-null   int64
 14  Educational special needs                       4424 non-null   int64
 15  Debtor                                          4424 non-null   int64
 16  Tuition fees up to date                         4424 non-null   int64
 17  Gender                                          4424 non-null   int64
 18  Scholarship holder                              4424 non-null   int64
 19  Age at enrollment                               4424 non-null   int64
 20  International                                   4424 non-null   int64
 21  Curricular units 1st sem (credited)             4424 non-null   int64
 22  Curricular units 1st sem (enrolled)             4424 non-null   int64
 23  Curricular units 1st sem (evaluations)          4424 non-null   int64
 24  Curricular units 1st sem (approved)             4424 non-null   int64
 25  Curricular units 1st sem (grade)                4424 non-null   float64
 26  Curricular units 1st sem (without evaluations)  4424 non-null   int64
 27  Curricular units 2nd sem (credited)             4424 non-null   int64
 28  Curricular units 2nd sem (enrolled)             4424 non-null   int64
 29  Curricular units 2nd sem (evaluations)          4424 non-null   int64
 30  Curricular units 2nd sem (approved)             4424 non-null   int64
 31  Curricular units 2nd sem (grade)                4424 non-null   float64
 32  Curricular units 2nd sem (without evaluations)  4424 non-null   int64
 33  Unemployment rate                               4424 non-null   float64
 34  Inflation rate                                  4424 non-null   float64
 35  GDP                                             4424 non-null   float64
 36  Target                                          4424 non-null   object
dtypes: float64(7), int64(29), object(1)
memory usage: 1.2+ MB
```

3. Imported Libraries

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sns

# Important imports for modeling and evaluation
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score

import sklearn.metrics as metrics

!pip install dython
from dython.nominal import associations
```

3.1 pandas: Data manipulation and analysis library in Python, providing easy-to-use data structures like DataFrames for working with structured data.

3.2 numpy: Numerical computing library in Python, offering support for large, multi-dimensional arrays and matrices, along with mathematical functions to operate on them.

3.3 seaborn: Statistical data visualization library built on top of matplotlib, providing an aesthetically pleasing interface for creating informative and attractive statistical graphics.

3.4 scikit-learn (sklearn): Machine learning library in Python, offering simple and efficient tools for data analysis and modeling, including classification, regression, clustering, and dimensionality reduction algorithms.

3.5 dython: A library for data analysis and feature engineering, specifically designed to work seamlessly with pandas, providing additional tools for handling missing data, encoding categorical features, and exploring data relationships.

3.6 matplotlib: 2D plotting library in Python, producing static, animated, and interactive visualizations in a variety of formats, and serving as the foundation for other visualization libraries like seaborn.

4. Data Preprocessing:

- Correcting column name from Nacionality to Nationality

```
[7] #DATA PREPROCESSING.

    # Rename column Nacionality.
    data.rename(columns={'Nacionality': 'Nationality'}, inplace=True)

    data.columns
```

```
Index(['Marital Status', 'Application mode', 'Application order', 'Course',
       'Daytime/evening attendance', 'Previous qualification',
       'Previous qualification (grade)', 'Nationality',
       'Mother's qualification', 'Father's qualification',
       'Mother's occupation', 'Father's occupation', 'Admission grade',
       'Displaced', 'Educational special needs', 'Debtor',
       'Tuition fees up to date', 'Gender', 'Scholarship holder',
       'Age at enrollment', 'International',
       'Curricular units 1st sem (credited)',
       'Curricular units 1st sem (enrolled)',
       'Curricular units 1st sem (evaluations)',
       'Curricular units 1st sem (approved)',
       'Curricular units 1st sem (grade)',
       'Curricular units 1st sem (without evaluations)',
       'Curricular units 2nd sem (credited)',
       'Curricular units 2nd sem (enrolled)',
       'Curricular units 2nd sem (evaluations)',
       'Curricular units 2nd sem (approved)',
       'Curricular units 2nd sem (grade)',
       'Curricular units 2nd sem (without evaluations)', 'Unemployment rate',
       'Inflation rate', 'GDP', 'Target'],
      dtype='object')
```

- Changing categorical columns to category datatype.

```
cont_cols = ['Age at enrollment', 'Curricular units 1st sem (credited)', 'Curricular units 1st sem (enrolled)',
             'Curricular units 1st sem (evaluations)', 'Curricular units 1st sem (approved)', 'Curricular units 1st sem (grade)',
             'Curricular units 1st sem (without evaluations)', 'Curricular units 2nd sem (credited)',
             'Curricular units 2nd sem (enrolled)', 'Curricular units 2nd sem (evaluations)',
             'Curricular units 2nd sem (approved)', 'Curricular units 2nd sem (grade)',
             'Curricular units 2nd sem (without evaluations)', 'Unemployment rate', 'Inflation rate', 'GDP', 'Target']

# Get all categorical variables except target.
cat_cols = ['Marital Status', 'Application mode', 'Application order', 'Course', 'Daytime/evening attendance',
            'Previous qualification', 'Nationality', "Mother's qualification", "Father's qualification",
            "Mother's occupation", "Father's occupation", 'Displaced', 'Educational special needs', 'Debtor',
            'Tuition fees up to date', 'Gender', 'Scholarship holder', 'International']

# Change categorical columns to category datatype.
data[cat_cols] = data[cat_cols].astype('category')

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4424 entries, 0 to 4423
Data columns (total 37 columns):
 #   Column                                          Non-Null Count  Dtype
---  ------                                          --------------  -----
 0   Marital Status                                  4424 non-null   category
 1   Application mode                                4424 non-null   category
 2   Application order                               4424 non-null   category
 3   Course                                          4424 non-null   category
 4   Daytime/evening attendance                      4424 non-null   category
 5   Previous qualification                          4424 non-null   category
 6   Previous qualification (grade)                  4424 non-null   float64
 7   Nationality                                     4424 non-null   category
 8   Mother's qualification                          4424 non-null   category
 9   Father's qualification                          4424 non-null   category
 10  Mother's occupation                             4424 non-null   category
 11  Father's occupation                             4424 non-null   category
 12  Admission grade                                 4424 non-null   float64
 13  Displaced                                       4424 non-null   category
 14  Educational special needs                       4424 non-null   category
 15  Debtor                                          4424 non-null   category
 16  Tuition fees up to date                         4424 non-null   category
 17  Gender                                          4424 non-null   category
 18  Scholarship holder                              4424 non-null   category
 19  Age at enrollment                               4424 non-null   int64
 20  International                                   4424 non-null   category
 21  Curricular units 1st sem (credited)             4424 non-null   int64
 22  Curricular units 1st sem (enrolled)             4424 non-null   int64
 23  Curricular units 1st sem (evaluations)          4424 non-null   int64
 24  Curricular units 1st sem (approved)             4424 non-null   int64
 25  Curricular units 1st sem (grade)                4424 non-null   float64
 26  Curricular units 1st sem (without evaluations)  4424 non-null   int64
 27  Curricular units 2nd sem (credited)             4424 non-null   int64
 28  Curricular units 2nd sem (enrolled)             4424 non-null   int64
 29  Curricular units 2nd sem (evaluations)          4424 non-null   int64
 30  Curricular units 2nd sem (approved)             4424 non-null   int64
 31  Curricular units 2nd sem (grade)                4424 non-null   float64
 32  Curricular units 2nd sem (without evaluations)  4424 non-null   int64
 33  Unemployment rate                               4424 non-null   float64
 34  Inflation rate                                  4424 non-null   float64
 35  GDP                                             4424 non-null   float64
 36  Target                                          4424 non-null   object
dtypes: category(18), float64(7), int64(11), object(1)
memory usage: 744.1+ KB
```

- Check for missing values in all columns

```
# Check for missing values.
data.isna().sum()
```

```
Marital Status                                  0
Application mode                                0
Application order                               0
Course                                          0
Daytime/evening attendance                      0
Previous qualification                          0
Previous qualification (grade)                  0
Nationality                                     0
Mother's qualification                          0
Father's qualification                          0
Mother's occupation                             0
Father's occupation                             0
Admission grade                                 0
Displaced                                       0
Educational special needs                       0
Debtor                                          0
Tuition fees up to date                         0
Gender                                          0
Scholarship holder                              0
Age at enrollment                               0
International                                   0
Curricular units 1st sem (credited)             0
Curricular units 1st sem (enrolled)             0
Curricular units 1st sem (evaluations)          0
Curricular units 1st sem (approved)             0
Curricular units 1st sem (grade)                0
Curricular units 1st sem (without evaluations)  0
Curricular units 2nd sem (credited)             0
Curricular units 2nd sem (enrolled)             0
Curricular units 2nd sem (evaluations)          0
Curricular units 2nd sem (approved)             0
Curricular units 2nd sem (grade)                0
Curricular units 2nd sem (without evaluations)  0
Unemployment rate                               0
Inflation rate                                  0
GDP                                             0
Target                                          0
dtype: int64
```

Note: There are no missing values

5. Exploratory Data Analysis

For Categorical data

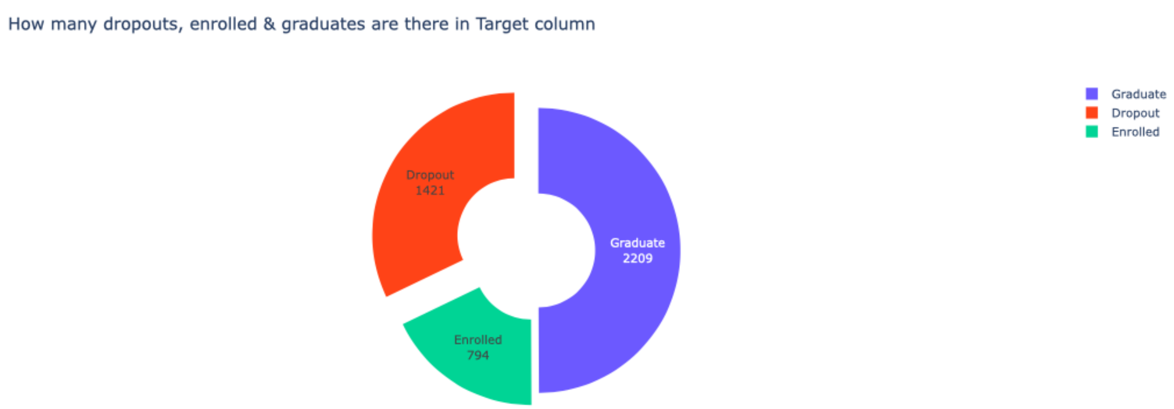
```
[10] #Exploratory data analysis
     new_data = data.copy()

     # Show all unique values of target variable.
     new_data['Target'].unique()
```

```
array(['Dropout', 'Graduate', 'Enrolled'], dtype=object)
```

5.1 Target

Creating a pie chart that depicts the proportions of total number of dropouts, total number of graduated, and total number of enrolled students.

```
[11] x = new_data['Target'].value_counts().index
     y = new_data['Target'].value_counts().values

     df = pd.DataFrame({
         'Target': x,
         'Count_T' : y
     })

     fig = px.pie(df,
                  names ='Target',
                  values ='Count_T',
                 title='How many dropouts, enrolled & graduates are there in Target column')

     fig.update_traces(labels=['Graduate','Dropout','Enrolled'], hole=0.4,textinfo='value+label', pull=[0,0.2,0.1])
     fig.show()
```

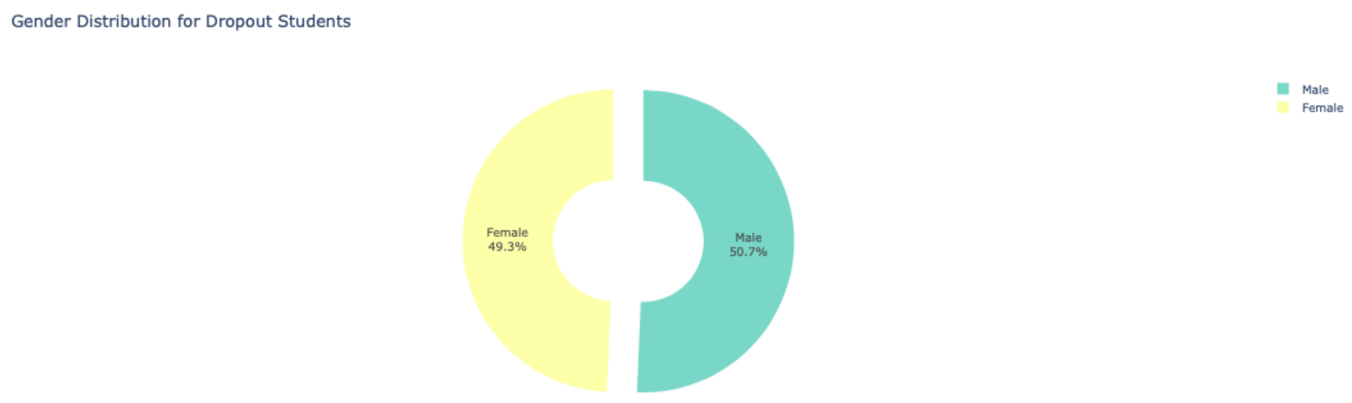


The number of graduated students is more than the number of dropout students.
The total number of graduated and dropout students is 2209 + 1421 = 3630, which is the number of observations for building our model.

5.2 Gender

Creating a pie chart that depicts the proportions of total number of female, and total number of male students who dropped out.

```
[12] # Gender

dropout_data = new_data.loc[new_data['Target'] == "Dropout"]

# Count the occurrences of each marital status
gender_counts = dropout_data['Gender'].value_counts().reset_index()

# Rename the columns for clarity in the plot
gender_counts.columns = ['Gender', 'Count']

# Calculate percentages
gender_counts['Percentage'] = (gender_counts['Count'] / gender_counts['Count'].sum()) * 100

# Plotting the pie chart using plotly.express
fig = px.pie(gender_counts, names='Gender', values='Percentage', title='Gender Distribution for Dropout Students', color_discrete_sequence=px.colors.qualitative.Set3)
fig.update_traces(labels=['Male','Female'], hole=0.4, textinfo='percent+label', pull=[0.1,0.1,0.1,0.1])

fig.show()

# 0 : Male
# 1 : Female

/usr/local/lib/python3.10/dist-packages/numpy/core/numeric.py:2463: FutureWarning:

elementwise comparison failed; returning scalar instead, but in the future will perform elementwise comparison
```



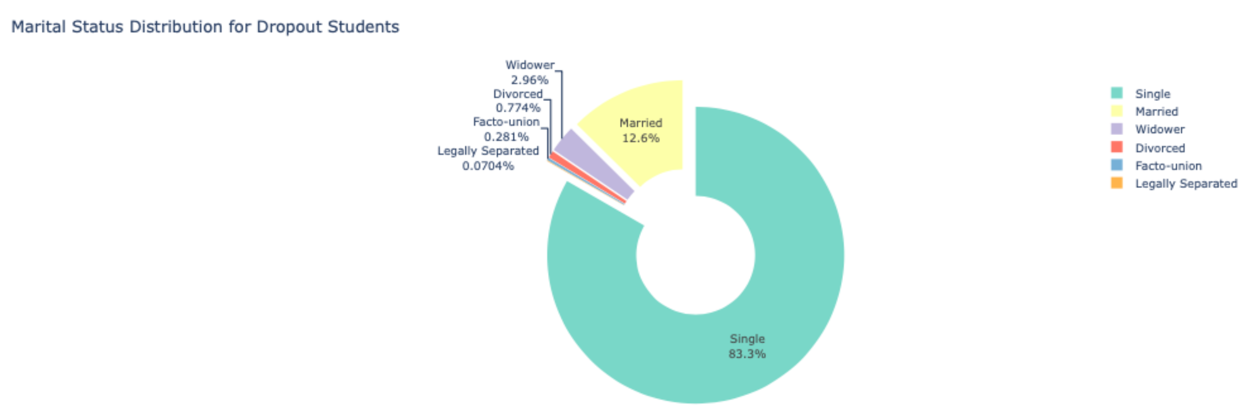
The number of dropout female and male students is almost equal.
There are more female students than male students in the dataset.
The dataset is slightly imbalanced, however it should not significantly affect the future model.

5.3 Marital Status

Creating a pie chart that shows the number of students in the following marital status groups: single, married, divorced, facto union, widower, and legally who dropped out.

```
# Marital Status

# Filter data for 'Target' equal to 0 (dropped out)
dropout_data = new_data.loc[new_data['Target'] == "Dropout"]

# Count the occurrences of each marital status
marital_status_counts = dropout_data['Marital Status'].value_counts().reset_index()

# Rename the columns for clarity in the plot
marital_status_counts.columns = ['MaritalStatus', 'Count']

# Calculate percentages
marital_status_counts['Percentage'] = (marital_status_counts['Count'] / marital_status_counts['Count'].sum()) * 100

# Plotting the pie chart using plotly.express
fig = px.pie(marital_status_counts, names='MaritalStatus', values='Percentage', title='Marital Status Distribution for Dropout Students', color_discrete_sequence=px.colors.qu
fig.update_traces(labels=['Single','Married','Widower','Divorced','Facto-union','Legally Separated'], hole=0.4, textinfo='percent+label', pull=[0.1,0.1,0.1,0.1,0.1,0.1])

fig.show()
```

```
/usr/local/lib/python3.10/dist-packages/numpy/core/numeric.py:2463: FutureWarning:

elementwise comparison failed; returning scalar instead, but in the future will perform elementwise comparison
```



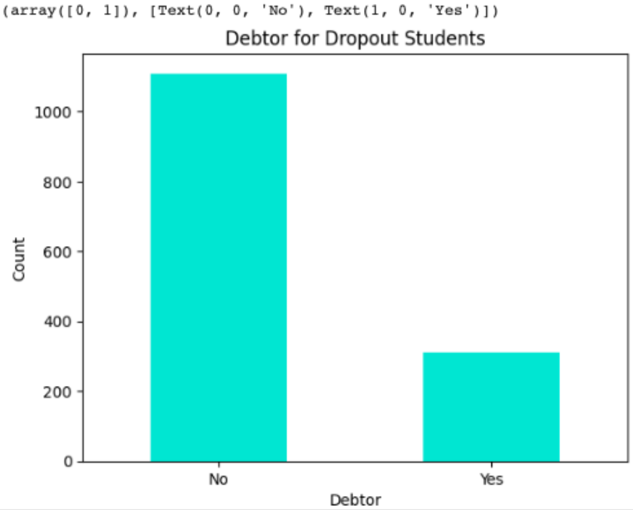
Number of Students with a marital status Single who have dropped out of the academic institutions is the highest among the total dropouts .

5.4 Debtor

Creating a bar chart for the dropped out students who have a debt or not

```
[14] # Debtor

new_data['Debtor']=new_data['Debtor'].replace({
    0: 'No',
    1: 'Yes'
})

filtered_data = new_data[new_data['Target'] == 'Dropout']

sorted_data = filtered_data.sort_values(by='Debtor', ascending=False)

value_counts = sorted_data['Debtor'].value_counts()

value_counts.plot(kind='bar', color='turquoise')
plt.xlabel('Debtor')
plt.ylabel('Count')
plt.title('Debtor for Dropout Students')
plt.xticks(rotation =0)
```

```
(array([0, 1]), [Text(0, 0, 'No'), Text(1, 0, 'Yes')])
```

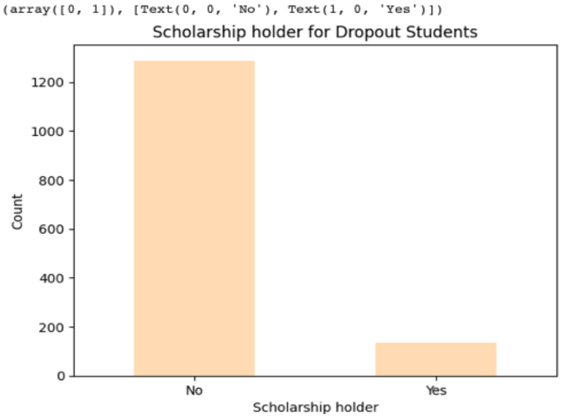


5.5 Scholarship Holder

Creating a bar chart for the dropped out students who have a scholarship or not

```
[15] # Scholarship Holder

new_data['Scholarship holder']=new_data['Scholarship holder'].replace({
    0: 'No',
    1: 'Yes'
})

filtered_data = new_data[new_data['Target'] == 'Dropout']

sorted_data = filtered_data.sort_values(by='Scholarship holder', ascending=False)

value_counts = sorted_data['Scholarship holder'].value_counts()

value_counts.plot(kind='bar', color='peachpuff')
plt.xlabel('Scholarship holder')
plt.ylabel('Count')
plt.title('Scholarship holder for Dropout Students')
plt.xticks(rotation = 0)
```

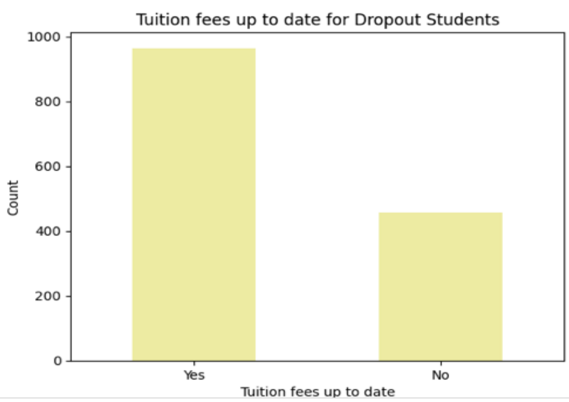
```
(array([0, 1]), [Text(0, 0, 'No'), Text(1, 0, 'Yes')])
```



5.6 Tuition fees up to Date

Creating a bar chart for the dropped out students who have paid tuition fees up to date or not

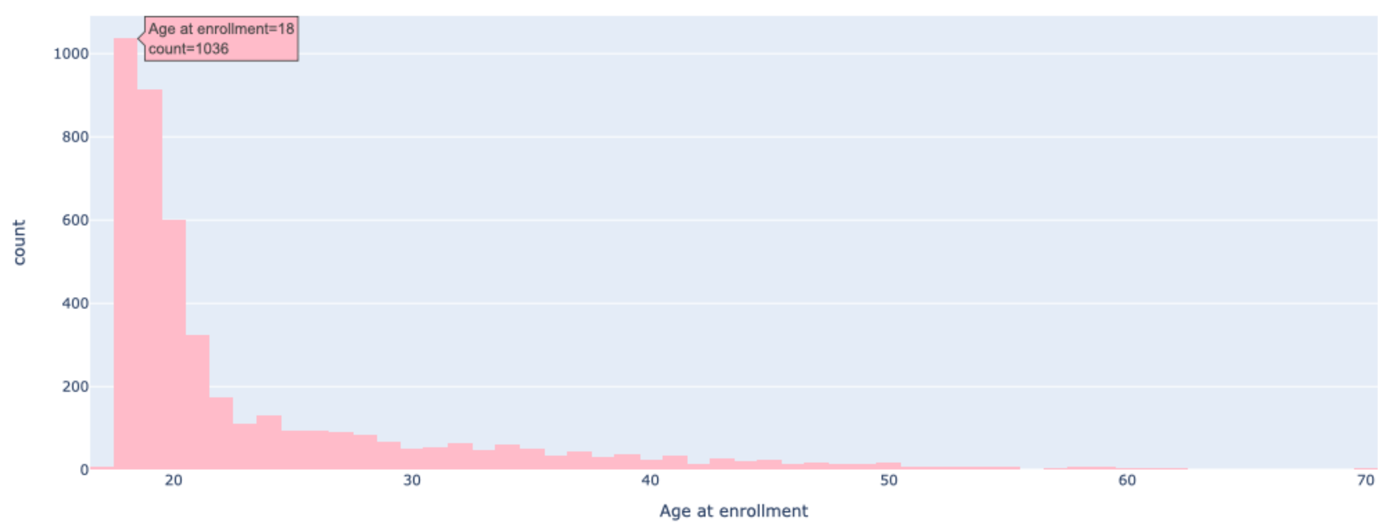
```
[16] # Tuition Fees

new_data['Tuition fees up to date']=new_data['Tuition fees up to date'].replace({
    0: 'No',
    1: 'Yes'
})

filtered_data = new_data[new_data['Target'] == 'Dropout']

sorted_data = filtered_data.sort_values(by='Tuition fees up to date', ascending=False)

value_counts = sorted_data['Tuition fees up to date'].value_counts()

value_counts.plot(kind='bar', color='palegoldenrod')
plt.xlabel('Tuition fees up to date')
plt.ylabel('Count')
plt.title('Tuition fees up to date for Dropout Students')
plt.xticks(rotation=0)
plt.show()
```



5.7 Age at Enrollment

Creating a histogram to depict the age at enrollment distribution

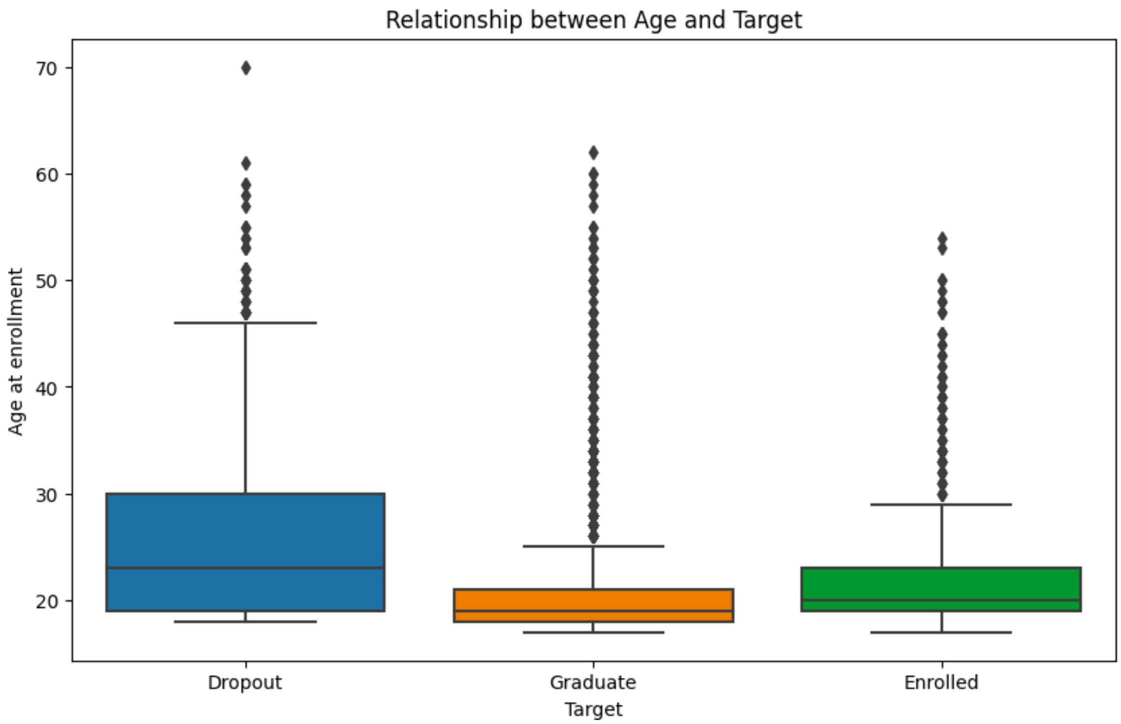
```
#Age At Enrollment

px.histogram(new_data['Age at enrollment'], x='Age at enrollment',color_discrete_sequence=['pink'])
```



The vast majority of students are 17-22 years old. The number of students decreases as the student's age increases.

Creating a boxplot to showcase relationship between Age at enrollment and Target

```
[18] plt.figure(figsize=(10, 6))
     sns.boxplot(x='Target', y='Age at enrollment', data=new_data)
     plt.xlabel('Target')
     plt.ylabel('Age at enrollment')
     plt.title('Relationship between Age and Target')
     plt.show()
```

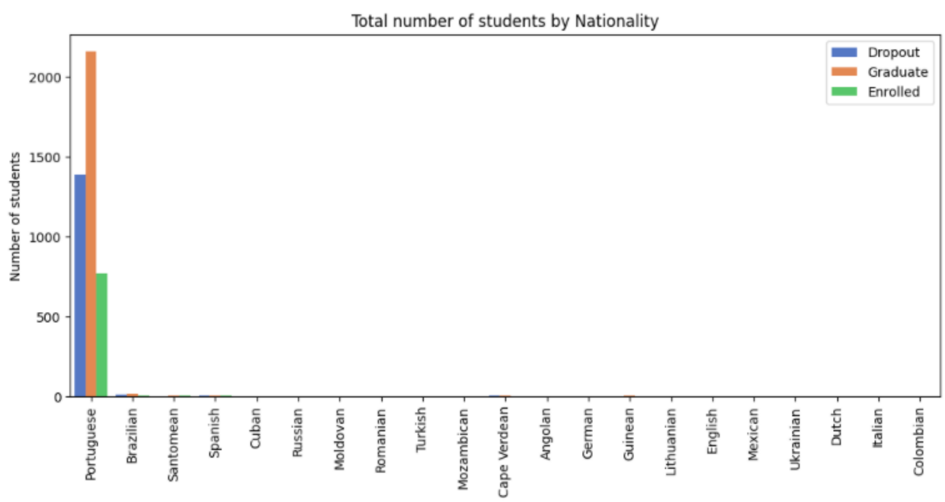


The highest graduation rate is the age group 17-22 with 72% of graduated students and 28% of dropped out students. There is no definitive distinction between other age groups.

5.8 Nationality

Creating a count chart showing the total number of dropout, graduate, and enrolled students based on Nationality.

```
[19] # Nationality

new_data[['Nationality']] = new_data[['Nationality']].replace({'Nationality': {1: 'Portuguese', 2: 'German', 6: 'Spanish',
                                                                 11: 'Italian', 13: 'Dutch', 14: 'English',
                                                                 17: 'Lithuanian', 21: 'Angolan', 22: 'Cape Verdean',
                                                                 24: 'Guinean', 25: 'Mozambican', 26: 'Santomean',
                                                                 32: 'Turkish', 41: 'Brazilian', 62: 'Romanian',
                                                                 100: 'Moldovan', 101: 'Mexican', 103: 'Ukrainian',
                                                                 105: 'Russian', 108: 'Cuban', 109: 'Colombian'}})

fig, ax = plt.subplots(figsize=(12, 5))
order = new_data[new_data['Target'] == 'Enrolled']['Nationality'].value_counts()
ax = sns.countplot(data=new_data, x='Nationality', hue='Target', palette='muted', order=order.index)
ax.set(xlabel=None, ylabel='Number of students', title='Total number of students by Nationality')
plt.xticks(rotation=90)
ax.legend_.set_title(None)
plt.show()
```

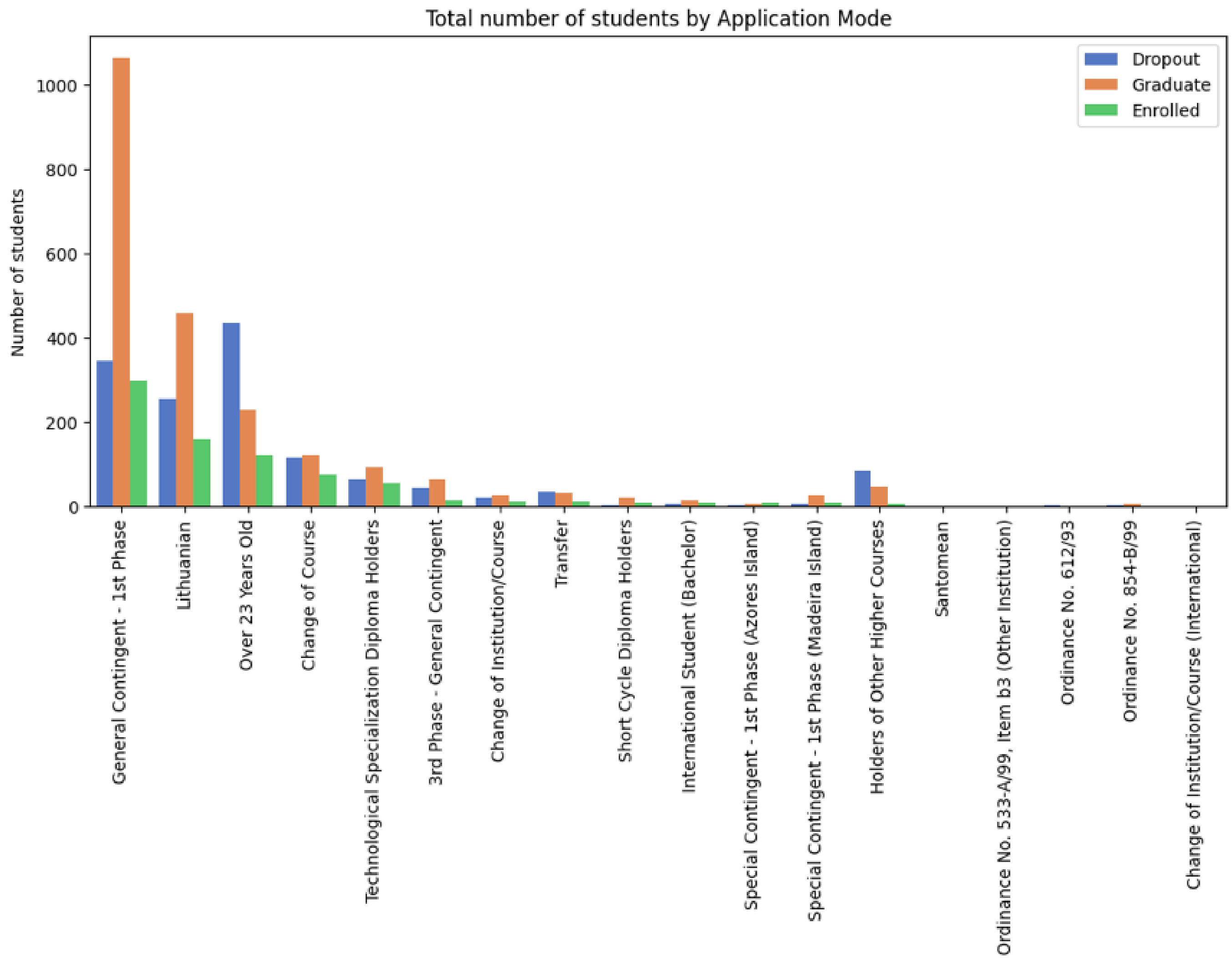


The vast majority of students are Portuguese (4,314 , almost 98% of the whole dataset.). Hence , We can say that the variable “Nationality” is highly imbalanced.
Other than this, There is no pattern to be seen in the above plots.

5.9 Application Mode

Creating a count plot that shows the proportion of dropout, and enrolled students in different application mode groups.

```
[20] # Application Mode

     new_data['Application mode'] = new_data['Application mode'].replace({
         1: 'General Contingent - 1st Phase',
         2: 'Ordinance No. 612/93',
         5: 'Special Contingent - 1st Phase (Azores Island)',
         7: 'Holders of Other Higher Courses',
         10: 'Ordinance No. 854-B/99',
         15: 'International Student (Bachelor)',
         16: 'Special Contingent - 1st Phase (Madeira Island)',
         17: '2nd Phase - General Contingent',
         18: '3rd Phase - General Contingent',
         26: 'Ordinance No. 533-A/99, Item b2) (Different Plan)',
         27: 'Ordinance No. 533-A/99, Item b3 (Other Institution)',
         39: 'Over 23 Years Old',
         42: 'Transfer',
         43: 'Change of Course',
         44: 'Technological Specialization Diploma Holders',
         51: 'Change of Institution/Course',
         53: 'Short Cycle Diploma Holders',
         57: 'Change of Institution/Course (International)',
         6: 'Spanish',
         11: 'Italian',
         13: 'Dutch',
         14: 'English',
         17: 'Lithuanian',
         21: 'Angolan',
         22: 'Cape Verdean',
         24: 'Guinean',
         25: 'Mozambican',
         26: 'Santomean',
         32: 'Turkish',
         41: 'Brazilian',
         62: 'Romanian',
         100: 'Moldovan',
         101: 'Mexican',
         103: 'Ukrainian',
         105: 'Russian',
         108: 'Cuban',
         109: 'Colombian'
     })

     fig, ax = plt.subplots(figsize=(12, 5))
     order = new_data[new_data['Target'] == 'Enrolled']['Application mode'].value_counts()
     ax = sns.countplot(data=new_data, x='Application mode', hue='Target', palette='muted', order=order.index)
     ax.set(xlabel=None, ylabel='Number of students', title='Total number of students by Application Mode')
     plt.xticks(rotation=90)
     ax.legend_.set_title(None)
     plt.show()
```



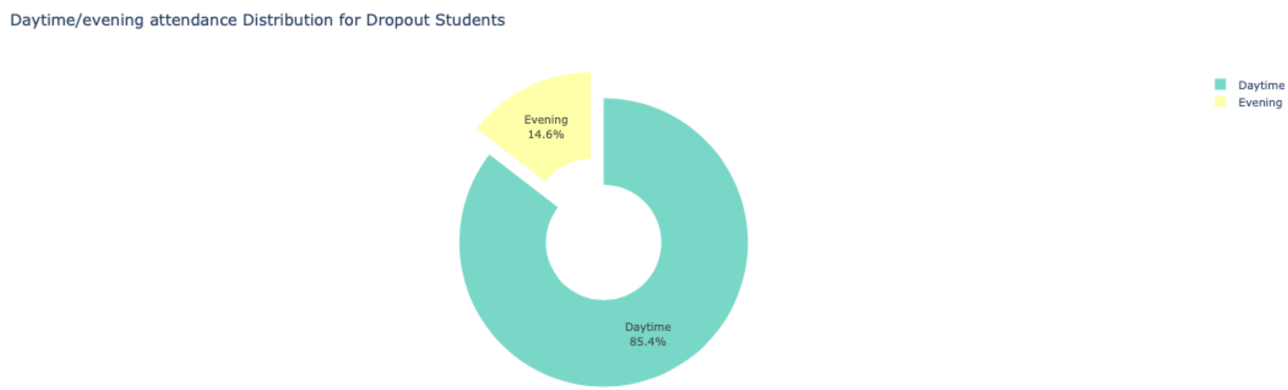
The majority of currently enrolled students have 1st phase - general contingent (1) application mode and the graduation rate of these students is highest.

5.10 Daytime/Evening Attendance

Creating a pie chart that depicts the proportions of total number of daytime, and total number of evening students who dropped out.

```
[21] # Daytime/evening attendance

     # Filter data for 'Target' equal to 0 (dropped out)
     dropout_data = new_data.loc[new_data['Target'] == "Dropout"]

     # Count the occurrences of each marital status
     attendance_counts = dropout_data['Daytime/evening attendance'].value_counts().reset_index()

     # Rename the columns for clarity in the plot
     attendance_counts.columns = ['Daytime/evening attendance', 'Count']

     # Calculate percentages
     attendance_counts['Percentage'] = (attendance_counts['Count'] / attendance_counts['Count'].sum()) * 100

     # Plotting the pie chart using plotly.express
     fig = px.pie(attendance_counts, names='Daytime/evening attendance', values='Percentage', title='Daytime/evening attendance Distribution for Dropout Students', color_discrete_
     fig.update_traces(labels=['Daytime','Evening'], hole=0.4, textinfo='percent+label', pull=[0.1,0.1,0.1,0.1])

     fig.show()
```

```
/usr/local/lib/python3.10/dist-packages/numpy/core/numeric.py:2463: FutureWarning:

elementwise comparison failed; returning scalar instead, but in the future will perform elementwise comparison
```



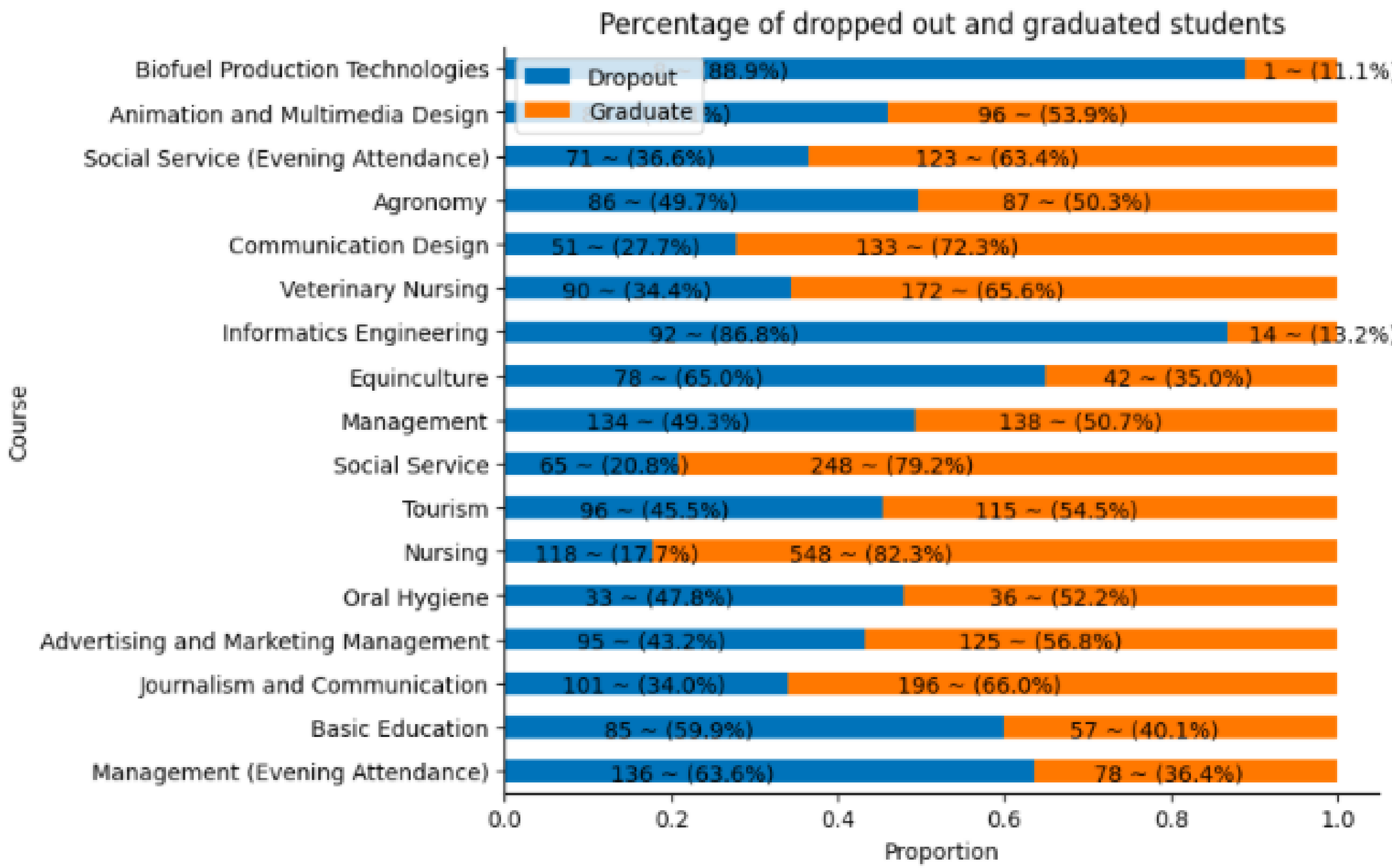
Vast majority of students who dropped out studied during the daytime.
The variable Daytime/evening attendance is imbalanced:

Note: It is not recommended to use Daytime/evening attendance as a predictor variable.

5.11 Course

Creating a stacked Bar-Plot that depicts the proportions of dropped out and Graduated students in a particular Course.

```
[22] # Course

new_data['Course'] = new_data['Course'].replace({
    33: 'Biofuel Production Technologies',
    171: 'Animation and Multimedia Design',
    8014: 'Social Service (Evening Attendance)',
    9003: 'Agronomy',
    9070: 'Communication Design',
    9085: 'Veterinary Nursing',
    9119: 'Informatics Engineering',
    9130: 'Equinculture',
    9147: 'Management',
    9238: 'Social Service',
    9254: 'Tourism',
    9500: 'Nursing',
    9556: 'Oral Hygiene',
    9670: 'Advertising and Marketing Management',
    9773: 'Journalism and Communication',
    9853: 'Basic Education',
    9991: 'Management (Evening Attendance)'
})

order = [
    'Biofuel Production Technologies',
    'Animation and Multimedia Design',
    'Social Service (Evening Attendance)',
    'Agronomy',
    'Communication Design',
    'Veterinary Nursing',
    'Informatics Engineering',
    'Equinculture',
    'Management',
    'Social Service',
    'Tourism',
    'Nursing',
    'Oral Hygiene',
    'Advertising and Marketing Management',
    'Journalism and Communication',
    'Basic Education',
    'Management (Evening Attendance)'
]

# Create a stacked bar plot.
filtered_data = pd.crosstab(index=new_data['Course'], columns=new_data[new_data['Target'] != 'Enrolled']['Target'])
data_prop = pd.crosstab(index=new_data['Course'], columns=new_data[new_data['Target'] != 'Enrolled']['Target'], normalize='index')
ax = data_prop.loc[order[::-1]].plot(kind='barh', stacked=True, figsize=(7, 6))
ax.set(xlabel='Proportion', ylabel='Course')
ax.spines[['right', 'top']].set_visible(False)
ax.legend_.set_title(None)
ax.set_title('Percentage of dropped out and graduated students')
plt.xticks(rotation=0)

for n, x in enumerate([*filtered_data.loc[order[::-1]].index.values]):
    for (proportion, count, y_loc) in zip(data_prop.loc[x], filtered_data.loc[x], data_prop.loc[x].cumsum()):
        plt.text(x=(y_loc - proportion) + (proportion / 5), y=n - 0.2, s=f'{count} ~ ({np.round(proportion * 100, 1)}%)')

plt.show()
```



The majority of students who successfully graduated were studying Nursing.

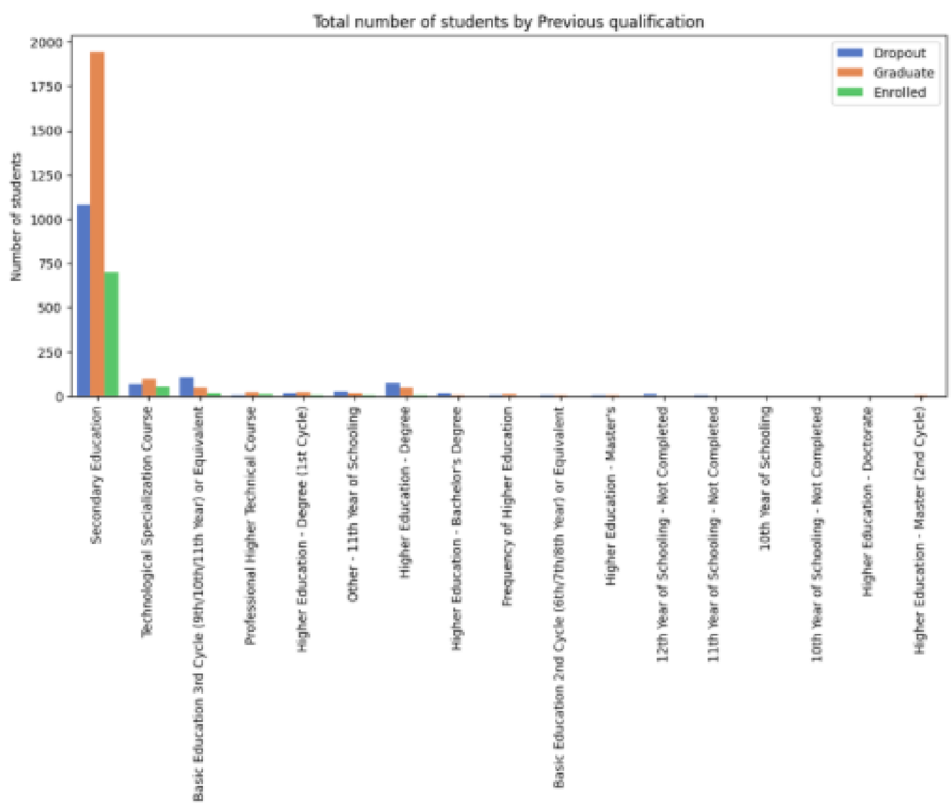
The highest dropout rates are in Informatics Engineering and Biofuel Production Technologies courses with 86.8% and 88.9% respectively.

In general, there is a pattern and the percentage of dropout and graduate students in different courses does not stay the same and we can use Course as a Predictor for our model.

5.12 Previous Qualifications

Creating a Count plot showing relationship between Graduate,Dropout and Enrolled Students with Previous qualification.

```
# Previous Quaifications
new_data['Previous qualification'] = new_data['Previous qualification'].replace({
      1: 'Secondary Education',
      2: "Higher Education - Bachelor's Degree",
      3: 'Higher Education - Degree',
      4: "Higher Education - Master's",
      5: 'Higher Education - Doctorate',
      6: 'Frequency of Higher Education',
      9: '12th Year of Schooling - Not Completed',
      10: '11th Year of Schooling - Not Completed',
      12: 'Other - 11th Year of Schooling',
      14: '10th Year of Schooling',
      15: '10th Year of Schooling - Not Completed',
      19: 'Basic Education 3rd Cycle (9th/10th/11th Year) or Equivalent',
      38: 'Basic Education 2nd Cycle (6th/7th/8th Year) or Equivalent',
      39: 'Technological Specialization Course',
      40: 'Higher Education - Degree (1st Cycle)',
      42: 'Professional Higher Technical Course',
      43: 'Higher Education - Master (2nd Cycle)'
})

fig, ax = plt.subplots(figsize=(12, 5))
order = new_data[new_data['Target'] == 'Enrolled']['Previous qualification'].value_counts()
ax = sns.countplot(data=new_data, x='Previous qualification', hue='Target', palette='muted', order=order.index)
ax.set(xlabel=None, ylabel='Number of students', title='Total number of students by Previous qualification')
plt.xticks(rotation=90)
ax.legend_.set_title(None)
plt.show()
```



The vast majority of students have secondary education (1) as Previous Qualification. The variable Previous qualification is highly imbalanced. Moreover, There is no pattern to be seen in the above plots.

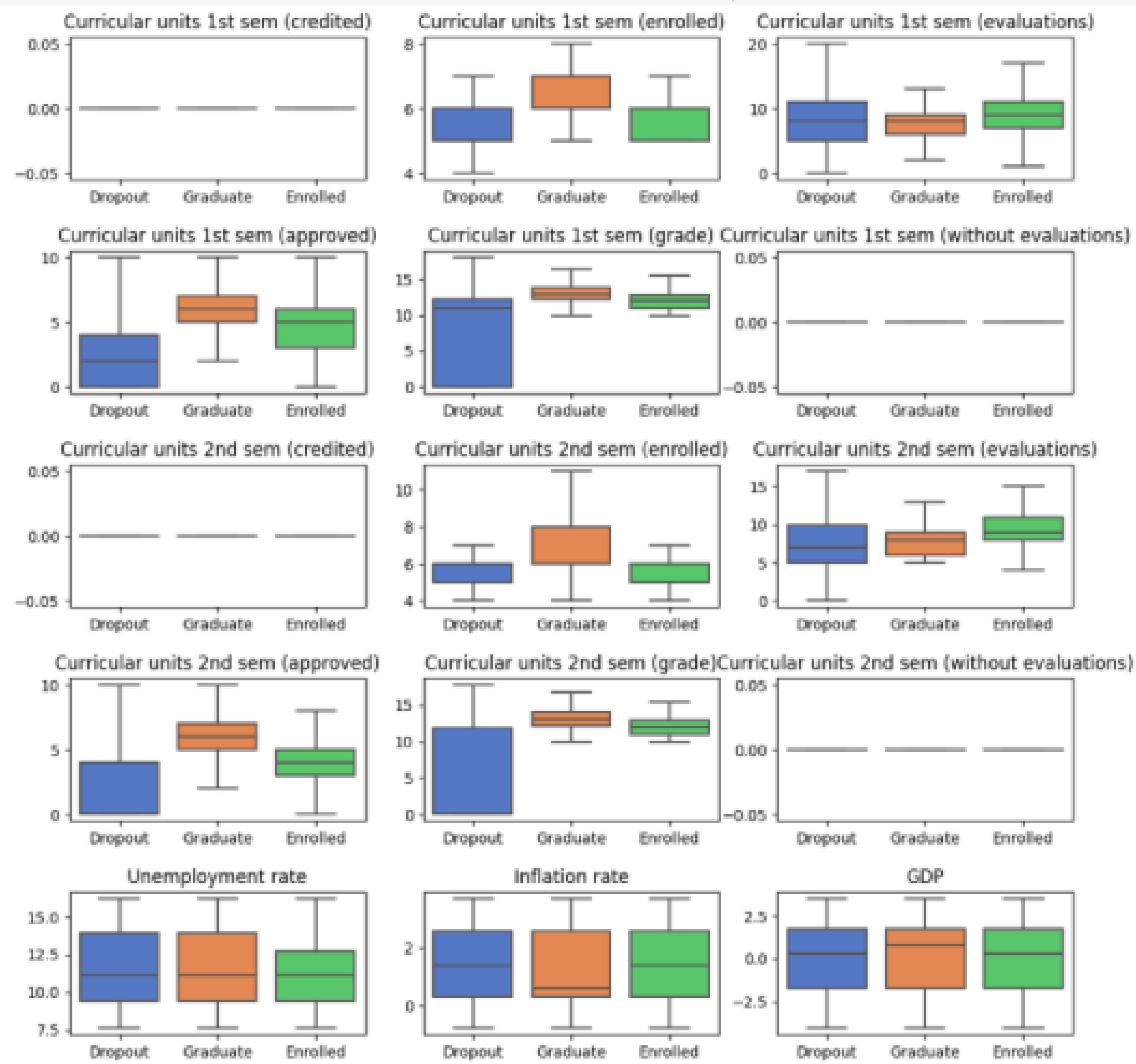
For Continuous data

5.13.1 Curricular units 1st sem (credited)
5.13.2 Curricular units 1st sem (enrolled)
5.13.3 Curricular units 1st sem (evaluations)
5.13.4 Curricular units 1st sem (approved)
5.13.5 Curricular units 1st sem (grade)
5.13.6 Curricular units 1st sem (without evaluations)
5.13.7 Curricular units 2nd sem (credited)
5.13.8 Curricular units 2nd sem (enrolled)
5.13.9 Curricular units 2nd sem (evaluations)
5.13.10 Curricular units 2nd sem (approved)
5.13.11 Curricular units 2nd sem (grade)
5.13.12 Curricular units 2nd sem (without evaluations)
5.13.13 Unemployment rate
5.13.14 Inflation rate
5.13.15 GDP

Creating a box plot for all the above mentioned continuous data.

[24]

```
# Continuous Variables

fig, axs = plt.subplots(5, 3, figsize=(12, 12))
plt.subplots_adjust(hspace=0.5)
cont_cols = ['Curricular units 1st sem (credited)',
        'Curricular units 1st sem (enrolled)',
        'Curricular units 1st sem (evaluations)',
        'Curricular units 1st sem (approved)',
        'Curricular units 1st sem (grade)',
        'Curricular units 1st sem (without evaluations)',
        'Curricular units 2nd sem (credited)',
        'Curricular units 2nd sem (enrolled)',
        'Curricular units 2nd sem (evaluations)',
        'Curricular units 2nd sem (approved)',
        'Curricular units 2nd sem (grade)',
        'Curricular units 2nd sem (without evaluations)',
         'Unemployment rate',
        'Inflation rate',
         'GDP']


ind = 0
for i in range(5):
    for j in range(3):
        sns.boxplot(new_data, x='Target', y=cont_cols[ind], showfliers=False, palette='muted', ax=axs[i, j])
        axs[i, j].set(xlabel=None, ylabel=None, title=cont_cols[ind])
        ind += 1
```

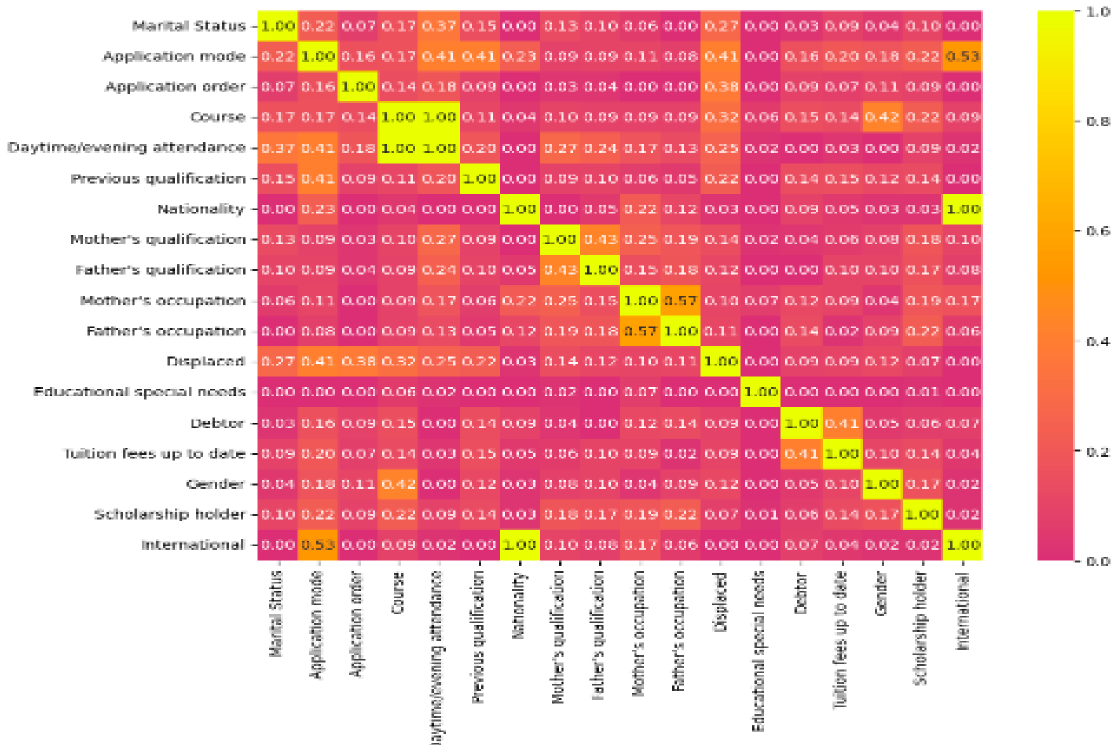


6. Feature Engineering

For Categorical data

Creating a correlation matrix for the categorical data.

```
# Feature Engineering
cat_data=data[cat_cols]
fig, ax = plt.subplots(figsize = (12, 8))
cramers_v = associations(cat_data, nom_nom_assoc='cramer', ax=ax, cmap='plasma')
```

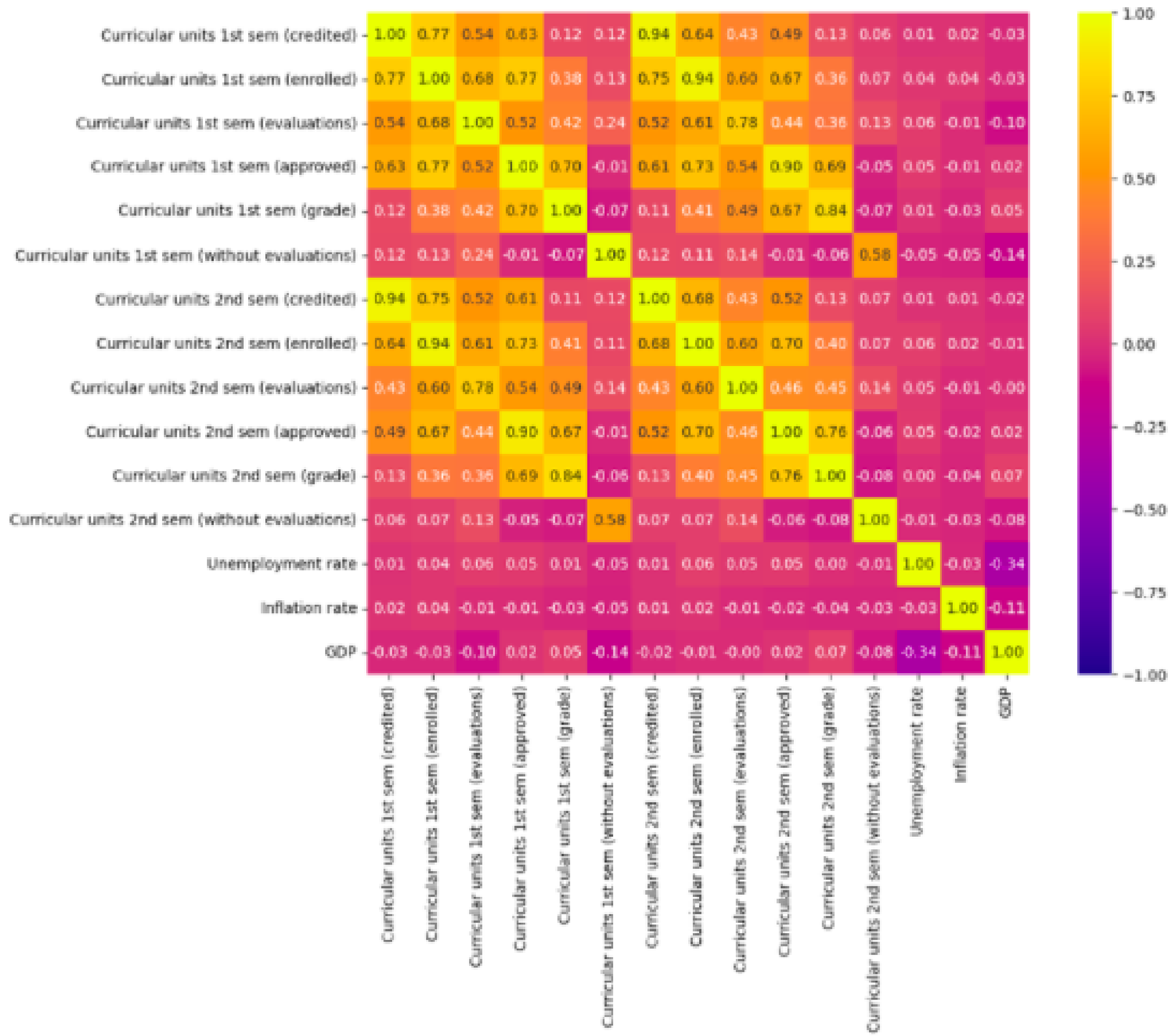


We have some features which have solid association with each other, these features are redundant and will not provide the model new information.
Mother's occupation and Father's occupation have good association with each other (0.57). Keep only Mother's occupation as it has higher association with the Target.
Mother's qualification and Father's qualification have good association with each other (0.43). Keep only Mother's qualification as it has higher association with the Target.
Debtor and Tuition fees up to date have good association with each other (0.41). Keep only Tuition fees up to date as it has higher association with the Target.
Displaced and Application mode have good association with each other (0.41). Keep only Application mode as it has higher association with the Target.

For Continuous data

Creating a correlation matrix for the continuous data.

```
[26] cont_data=data[cont_cols]
     fig, ax = plt.subplots(figsize = (12, 8))
     cramers_v = associations(cont_data, nom_nom_assoc='cramer', ax=ax, cmap='plasma')
```



According to the correlation matrix we will select following Categorical features: Application mode, Course, Previous qualification, Mother's qualification, Tuition fees up to date, Mother's occupation, Gender, Scholarship holder,

According to the correlation matrix we will select following Continuous features: Age at enrollment, Curricular units 1st sem (approved), Curricular units 2nd sem (approved).

```
selected_cols = ['Application mode', 'Course', 'Previous qualification', "Mother's qualification", 'Tuition fees up to date',
        "Mother's occupation", 'Gender', 'Scholarship holder', 'Age at enrollment', 'Curricular units 1st sem (approved)',
        'Curricular units 2nd sem (approved)', 'Target']

# Keep only relevant columns.
new_data=new_data[selected_cols]

# Remove enrolled students.
new_data = new_data[new_data['Target'] != 'Enrolled']

# Convert into numerical data type.

cols = ['Tuition fees up to date', 'Gender', 'Scholarship holder','Target']
new_data[cols] = new_data[cols].astype('int32')
```

Now, The dataset is ready for model building.

7. Model Building

We will be apply the following ML classification algorithms :

- Logistic regression
- Decision Tree Classifier
- Random Forest Classifier
- Naive Bayes Classifier
- Support Vector Machine

The motivation to apply all these algorithms was that we wanted to compare their accuracy results to see which algorithm works better on our dataset.

Splitting the dataset into training and testing set with (80% , 20%) ratio respectively, and evaluating the models based on their accuracy, precision, recall, F1 score

```
# Model Building

results = pd.DataFrame(columns=['Algorithm', 'Accuracy', 'Precision', 'Recall', 'F1 Score'])

# Predicting variable.
y = new_data['Target']

# Predictor features.
X = new_data.copy()
X = X.drop('Target', axis = 1)

# Create training and test sets, 80% and 20% respectively.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

For each case, we've visualized the Confusion Matrix along with it. We've also displayed the accuracy percentage for each case too.

In the end we have displayed the combined ROC graph of these ML classification algorithms.

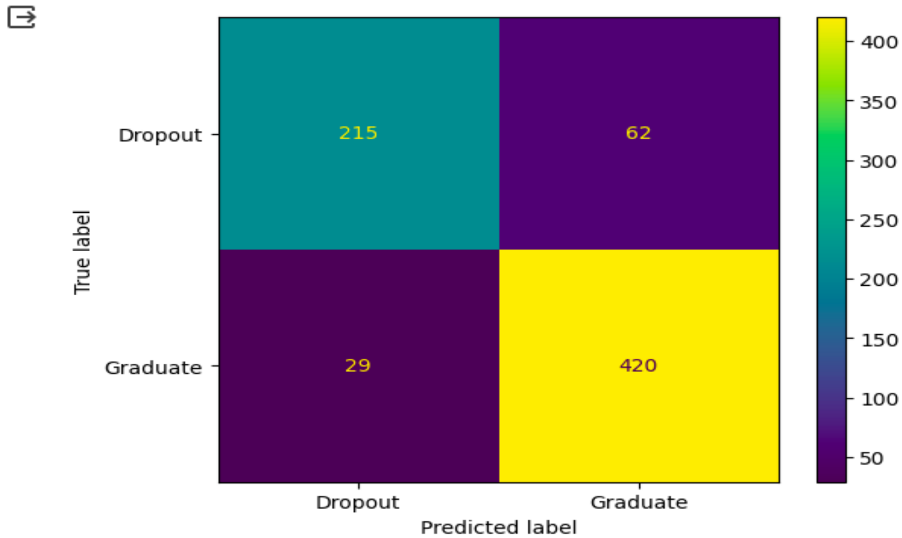
8. Models

8.1 Naive Bayes Classifier

```python
# Naive Bayes

# Normalize data.
X_scaled_train = StandardScaler().fit_transform(X_train)
X_scaled_test = StandardScaler().fit_transform(X_test)
# Fit the model.
gnb = GaussianNB()

gnb.fit(X_scaled_train, y_train)
# Get the predictions on test data.
y_preds = gnb.predict(X_scaled_test)

print_results('Naive Bayes', y_test, y_preds)


num_iterations = 20
accuracy_scores = []

y_preds_prob_gnb = gnb.predict_proba(X_test)[:, 1]

for i in range(num_iterations):
    scores = cross_val_score(gnb, X_scaled_train, y_train, cv=20, scoring='accuracy')
    accuracy_scores.extend(scores)

# Print the average accuracy
average_accuracy = sum(accuracy_scores) / len(accuracy_scores)
print(f"Average Accuracy over {num_iterations} iterations:", average_accuracy)
```

```
Naive Bayes
Accuracy: 0.875
Precision: 0.871
Recall: 0.935
F1 Score: 0.902
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning:

X has feature names, but GaussianNB was fitted without feature names
```

```
conf_matrix_plot(gnb, X_scaled_test, y_test)
```



Naive Bayes

- Accuracy: 0.875
- Precision: 0.871
- Recall: 0.935
- F1 Score: 0.902
- **Average Accuracy over 20 iterations: 0.8636230514879547**

Why we are using this:

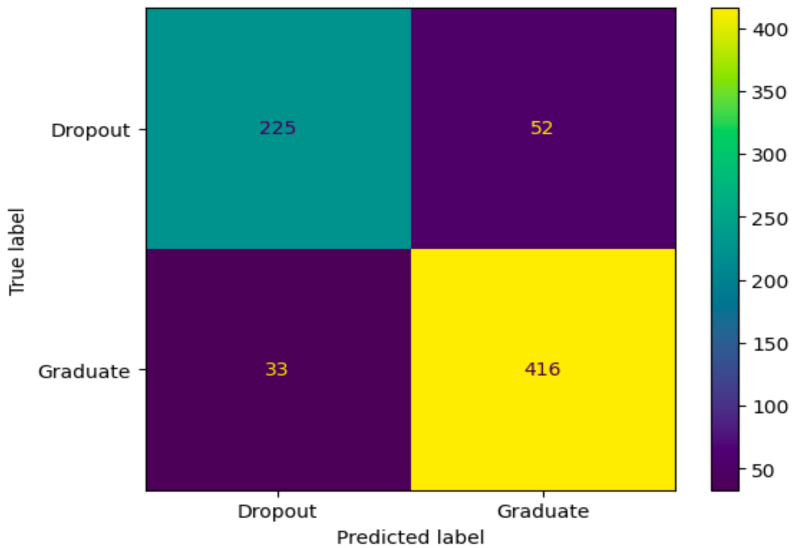
- Naive Bayes assumes independence between features, which can be a strength if this assumption approximately holds.
- Suitable for high-dimensional datasets.
- Naive Bayes might be effective if the features are conditionally independent given the class (dropout or academic success).
- It's computationally efficient and can work well with large datasets.

8.2 Logistic Regression

```
[33] # Logistic Regression

     X_scaled_train = StandardScaler().fit_transform(X_train)
     X_scaled_test = StandardScaler().fit_transform(X_test)
     lr = LogisticRegression()
     lr.fit(X_scaled_train, y_train)
     y_preds = lr.predict(X_scaled_test)

     print_results('Logistic Regression', y_test, y_preds)

     num_iterations = 20
     accuracy_scores = []

     for i in range(num_iterations):
         scores = cross_val_score(lr, X_scaled_train, y_train, cv=20, scoring='accuracy')
         accuracy_scores.extend(scores)

     y_preds_prob_lr = lr.predict_proba(X_scaled_test)[:, 1]

     # Print the average accuracy
     average_accuracy = sum(accuracy_scores) / len(accuracy_scores)
     print(f"Average Accuracy over {num_iterations} iterations:", average_accuracy)
```

```
Logistic Regression
Accuracy: 0.883
Precision: 0.889
Recall: 0.927
F1 Score: 0.907
Average Accuracy over 20 iterations: 0.8894544166273
```

```
[57] conf_matrix_plot(lr, X_scaled_test, y_test)
```



Logistic Regression
- Accuracy: 0.883
- Precision: 0.889
- Recall: 0.927
- F1 Score: 0.907
- **Average Accuracy over 20 iterations: 0.8894544166273**

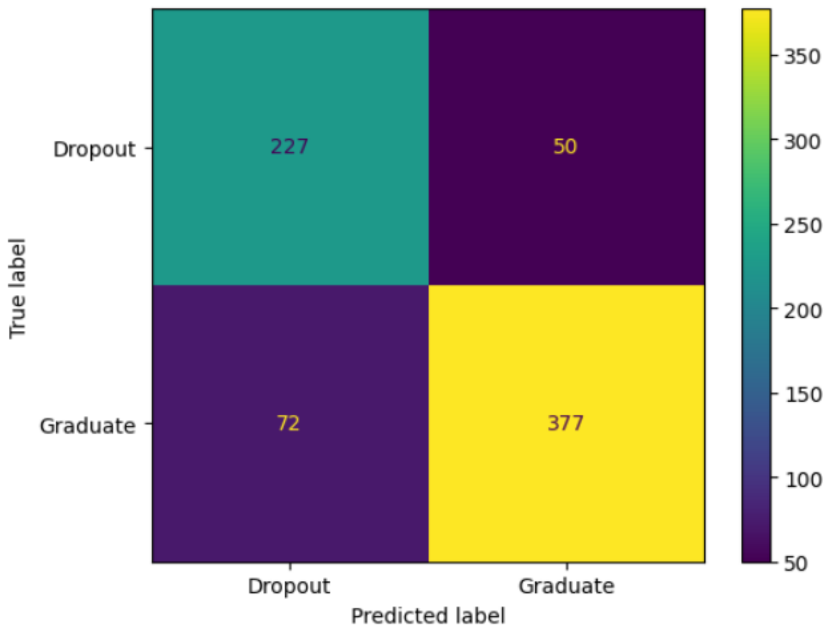
Why we are using this:
- Logistic Regression assumes a linear relationship between the independent variables and the log-odds of the dependent variable.
- Suitable when the classes are linearly separable.
- The linear relationship assumption might be a good fit for your dataset.
- Logistic Regression is less prone to overfitting, which could be beneficial if your dataset is not very large.

8.3 Decision Tree

```python
# Decision Tree

decision_tree = DecisionTreeClassifier(random_state=0)
decision_tree.fit(X_train, y_train)
y_preds = decision_tree.predict(X_test)

print_results('Decision Tree', y_test, y_preds)

num_iterations = 20
accuracy_scores = []

for i in range(num_iterations):
    scores = cross_val_score(decision_tree, X_scaled_train, y_train, cv=20, scoring='accuracy')
    accuracy_scores.extend(scores)

y_preds_prob_dt = decision_tree.predict_proba(X_test)[:, 1]

# Print the average accuracy
average_accuracy = sum(accuracy_scores) / len(accuracy_scores)
print(f"Average Accuracy over {num_iterations} iterations:", average_accuracy)
```

```
Decision Tree
Accuracy: 0.832
Precision: 0.883
Recall: 0.840
F1 Score: 0.861
Average Accuracy over 20 iterations: 0.8401771374586703
```

```
conf_matrix_plot(decision_tree, X_test, y_test)
```



Decision Tree

- Accuracy: 0.832
- Precision: 0.883
- Recall: 0.840
- F1 Score: 0.861
- **Average Accuracy over 20 iterations: 0.8401771374586703**

Why we are using this:

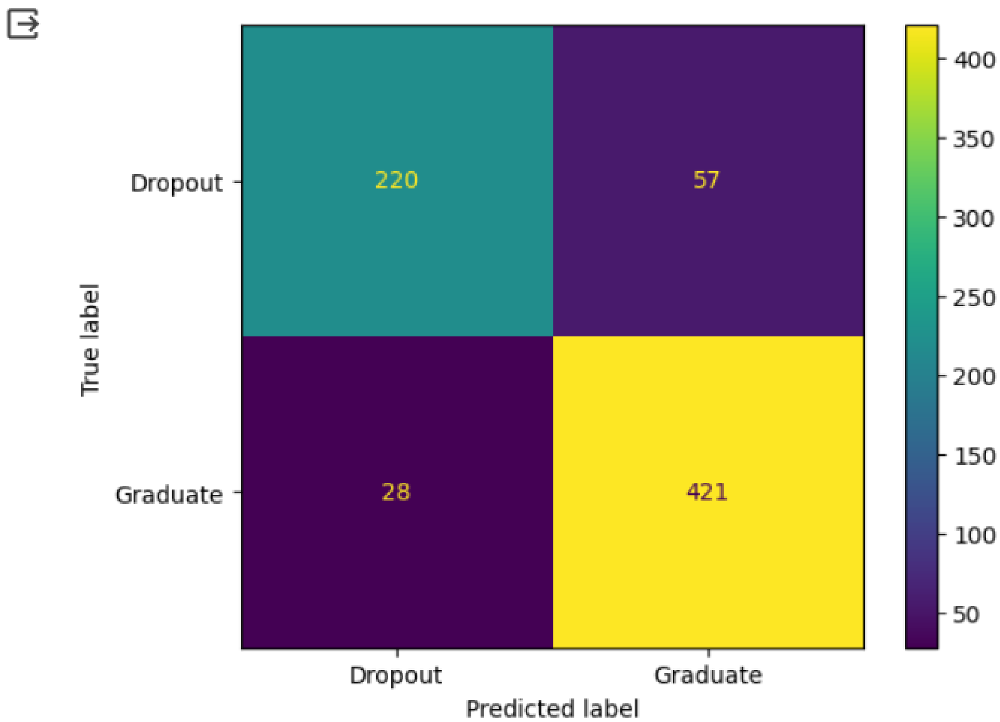
- Decision Trees can capture non linear relationships in the data.
- They are easy to interpret.
- Decision Trees might struggle if the dataset has complex relationships that are hard to capture with a single split at each node.
- Prone to overfitting, especially if the tree is deep and the dataset is not large enough.

8.4 Random Forest

```python
# Random Forest

rf = RandomForestClassifier(random_state=0)
rf.fit(X_train, y_train)
y_preds = rf.predict(X_test)

print_results('Random forest', y_test, y_preds)

num_iterations = 20
accuracy_scores = []

y_preds_prob_rf = rf.predict_proba(X_test)[:, 1]

for i in range(num_iterations):
    scores = cross_val_score(rf, X_scaled_train, y_train, cv=20, scoring='accuracy')
    accuracy_scores.extend(scores)

# Print the average accuracy
average_accuracy = sum(accuracy_scores) / len(accuracy_scores)
print(f"Average Accuracy over {num_iterations} iterations:", average_accuracy)
```

```
Random forest
Accuracy: 0.883
Precision: 0.881
Recall: 0.938
F1 Score: 0.908
Average Accuracy over 20 iterations: 0.8880538497874321
```

```
conf_matrix_plot(rf, X_test, y_test)
```



Random forest

- Accuracy: 0.883
- Precision: 0.881
- Recall: 0.938
- F1 Score: 0.908
- **Average Accuracy over 20 iterations: 0.8880538497874321**

Why we are using this:

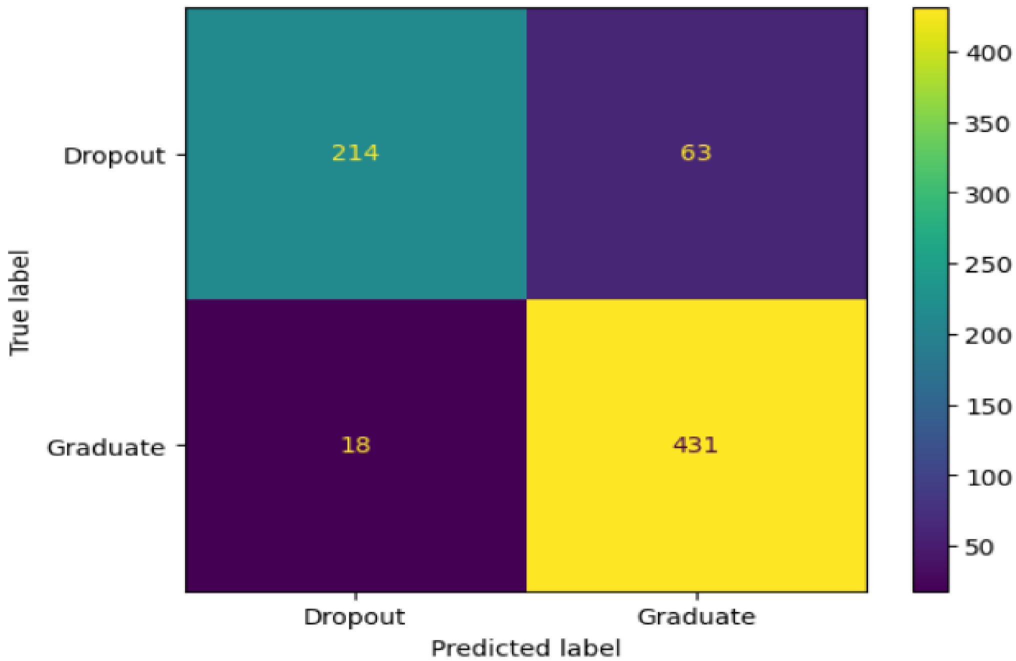
- Random Forest is an ensemble method that can handle non-linearity and complex relationships in the data.
- Effective in dealing with irrelevant features and outliers.
- Random Forest might capture complex interactions between attributes in predicting student outcomes.
- It can handle a mix of numerical and categorical features well.

8.5 Support Vector Machine (SVM)

```python
# SVM

X_scaled_train = StandardScaler().fit_transform(X_train)
X_scaled_test = StandardScaler().fit_transform(X_test)
svm = SVC(probability=True)
svm.fit(X_scaled_train, y_train)
y_preds = svm.predict(X_scaled_test)

print_results('SVM', y_test, y_preds)

num_iterations = 20
accuracy_scores = []

y_preds_prob_svm = svm.predict_proba(X_scaled_test)[:, 1]

for i in range(num_iterations):
    scores = cross_val_score(svm, X_scaled_train, y_train, cv=20, scoring='accuracy')
    accuracy_scores.extend(scores)

# Print the average accuracy
average_accuracy = sum(accuracy_scores) / len(accuracy_scores)
print(f"Average Accuracy over {num_iterations} iterations:", average_accuracy)
```

```
SVM
Accuracy: 0.888
Precision: 0.872
Recall: 0.960
F1 Score: 0.914
Average Accuracy over 20 iterations: 0.898398677373639
```

```
conf_matrix_plot(svm, X_scaled_test, y_test)
```



SVM

- Accuracy: 0.888
- Precision: 0.872
- Recall: 0.960
- F1 Score: 0.914
- **Average Accuracy over 20 iterations: 0.898398677373639**

Why we are using this:

- SVM works well when there is a clear margin of separation between classes.
- Effective in high-dimensional spaces, which is beneficial if your dataset has many features.
- The dataset might have a clear separation between students who drop out and those who succeed academically.
- SVM is robust to outliers, and if there are outliers in the data, SVM can handle them effectively.

9. Model Evaluation and Conclusion

9.1 Result matrix for different ML classification algorithms

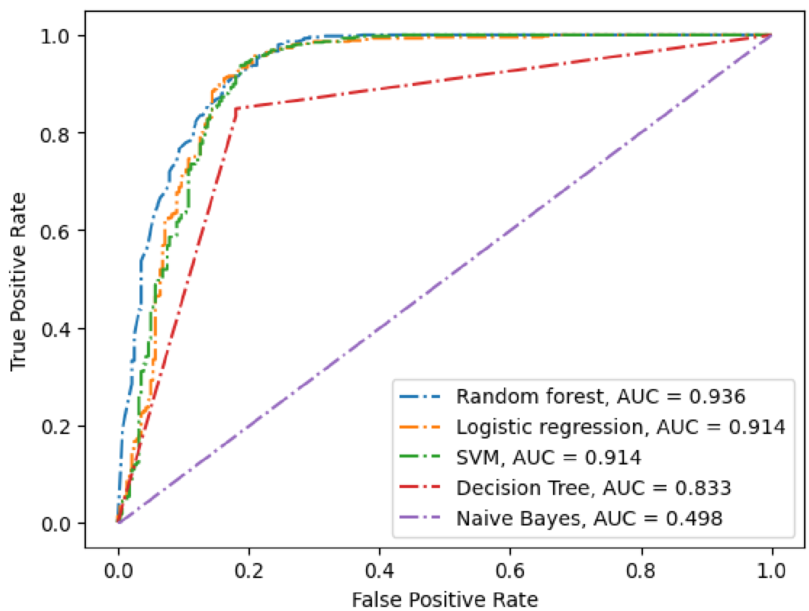
```
[80] # Compare models.
     results_unique=results.drop_duplicates()
     results_unique.sort_values(by=['Accuracy'], ascending=False)
```

	Algorithm	Accuracy	Precision	Recall	F1 Score
4	SVM	0.888430	0.872470	0.959911	0.914104
1	Logistic Regression	0.882920	0.888889	0.926503	0.907306
3	Random forest	0.882920	0.880753	0.937639	0.908306
0	Naive Bayes	0.874656	0.871369	0.935412	0.902256
2	Decision Tree	0.831956	0.882904	0.839644	0.860731

From the above table , it can be clearly observed that SVM,Logistic Regression and Random Forest are the models with highest Accuracy,Precision,Recall and F1 Score.But only the above parameters might not be sufficient to arrive at our decision for best classifier. Let's explore another approach.

9.2 ROC Graph

```
y_preds_probs = {'Naive Bayes': y_preds_prob_gnb, 'Logistic regression': y_preds_prob_lr, 'SVM': y_preds_prob_svm,
                 'Decision Tree': y_preds_prob_dt, 'Random forest': y_preds_prob_rf}

aucs = {}
for i in y_preds_probs.keys():
    aucs[i] = metrics.roc_auc_score(y_test, y_preds_probs[i])

for i in dict(sorted(aucs.items(), key=operator.itemgetter(1), reverse=True)).keys():
    fpr, tpr, thresholds = metrics.roc_curve(y_test, y_preds_probs[i])
    auc = aucs[i]
    plt.plot(fpr, tpr, linestyle='dashdot', label=f'{i}, AUC = {auc:.3f}')

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='best')
plt.show()
```



9.3 Conclusion: Here, Random Forest has the highest AUC (Area Under Curve) among all models. Hence, now we can confidently say that Random Forest is the best Classifier for our Dataset.

10. References:

1. M.V.Martins, D. Tolledo, J. Machado, L. M.T. Baptista, V.Realinho. (2021) "Early prediction of student's performance in higher education: a case study" Trends and Applications in Information Systems and Technologies, vol.1, in Advances in Intelligent Systems and Computing series. Springer. DOI: 10.1007/978-3-030-72657-7_16 , Source of Dataset: Predict students' dropout and academic success

2. Official Documentation of Scikit-Learn Documentation, Matplotlib Documentation, Pandas Documentation,Seaborn Documentation, NumPy Documentation, Operator Module Documentation, Dython Documentation

3. Class Notes of IDS Course.

4. Machine Learning by Andrew Ng on Coursera, Towards Data Science on Medium