

Final Project Submission: Student Academic Success Classification Model Analysis

(Phase 3)

- Student Name: Tenicka Norwood
- Program Pace: self paced
- Scheduled Project Review Time:
- Instructor name: Morgan Jones
- Blog post Url: <https://medium.com/@tenicka.norwood/the-meaning-of-life-the-universe-and-everything-9423dcb79c06>



Business Understanding

- **Stakeholder:** Instituto Politecnico de Portalegre
- **Business Case:** I have been tasked by the Instituto Politecnico de Portalegre to accurately classify students' academic success. Researchers at the Instituto Politecnico de Portalegre want to reduce the rate of student academic failure in higher education.

In this case, the goal is craft a reliable model that can be refined over time as more information becomes available by using machine learning techniques to identify which students are at risk at earlier stages of their academic path in order to put strategies to support students in place to mitigate their likelihood of dropping out of higher education. While the model will be created with data from higher education institutions in Portugal, the lag in academic success within higher education is also a challenge within the United States. Colleges and Universities face the ever pressing challenge of identifying students who are at risk of not graduating on time and providing effective interventions to move those students back onto a positive pathway to graduation.

The completion rates of (US) undergraduates within six years of enrollment stand at only **62.3%** as of 2022.

-- Hanneh Bareham and Chelsea Wing (bankrate.com)

Carnevale et. al. (2021) Georgetown University in their study highlighted the substantial disparity in lifetime earnings based on educational attainment. In particular, the gist of this study showed that an average person with only a high school diploma or GED will earn an estimated **1.6 million dollars** in their lifetime, compared to a person with a bachelor's degree who potentially can earn **~2.8 million dollars** in their lifetime. This profound disparity in earning potential underscores the transformative power of higher education and highlights the significance of timely graduation in enhancing students' long-term financial security.

Objectives

We will used the **OSEMiN** pipeline to:

- Obtain → Import the data.
- Scrub → Manage the datatypes, resolve missing data or duplicates.
- Explore → Identify patterns within the relationships between variables in the data.
- Model → Create a set of predictive models.
- iNterpret → Identify insights and create visualizations of findings.

Data Understanding

The University of California (Irvine) hosts a machine learning repository with datasets that can be evaluated using machine learning techniques. The dataset used in this exploration can be found [here](#). This dataset was created as a part of a project to reduce academic dropout and failure rates in higher education by leveraging machine learning to identify at risk students early within their academic careers and provide them support to improve their likelihood of on time graduation.

The data includes information that is known from the time a student enrolls including their:

- academic path
- demographics
- social-economic factors

There are three classifications of students within this dataset:

- dropout
- enrolled
- graduate

The data was preprocessed to address missing values and anomalies. The dataset has **37** columns with **4424** rows of data. I am going to use a minimum of 4 machine learning algorithms to classify this data and will also use exploration to determine:

- How does the number of credits students enrolled in each semester influence the likelihood of students graduating on time?
- Does the timely payment of tuition fees have any impact on the graduation status of students?
- Is there a relationship between the age a student is at enrollment and the graduation status of students?

Features

Name	Role	Type	Data Type	Description
Marital status	Feature	Discrete	int	1 — single 2 — married 3 — widowed 4 — divorced 5 — facto union 6 — legally separated
Application mode	Feature	Discrete	int	1 – 1st phase – general contingent 2 – Ordinance No. 612/93 5 – 1st phase – special contingent (Azores Island) 7 – Holders of other higher courses 10 – Ordinance No. 854-B/99 15 – International student (bachelor) 16 – 1st phase – special contingent (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 –

Name	Role	Type	Data Type	Description
				Change of institution/course 53 – Short cycle diploma holders 57 – Change of institution/course (International)
Application order	Feature	Discrete	int	Application order between 0 - first choice; and 9 last choice
Course	Feature	Discrete	int	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 - Equiculture 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)
Daytime/evening attendance	Feature	Discrete	int	1 — daytime 0 — evening
Previous qualification	Feature	Discrete	int	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 equiv. 38 – Basic education 2nd cycle (6th/7th/8th year) or equiv. 39 – Technological specialization course 40 – Higher education - degree (1st cycle) 42 – Professional higher technical course 43 – Higher education - master (2nd cycle)
Previous qualification (grade)	Feature	Continuous	float	Grade of previous qualification between 0 and 200
Nacionality	Feature	Discrete		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 — Moldova (Republic of); 101 — Mexican; 103 — Ukrainian; 105 — Russian; 108 — Cuban; 109 — Colombian
Mother's qualification	Feature	Discrete	int	1 - Secondary Education - 12th Year of Schooling or Eq. 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 11 - 7th Year (Old) 12 - Other - 11th Year of Schooling 14 - 10th Year of Schooling 18 - General commerce course 19 - Basic Education 3rd Cycle (9th/10th/11th Year) or Equiv. 22 - Technical-professional course 26 - 7th year of schooling 27 - 2nd cycle of the general high school course 29 - 9th Year of Schooling - Not Completed 30 - 8th year of schooling 34 - Unknown 35 - Can't read or write 36 - Can read without having a 4th year of schooling 37 - Basic education 1st cycle (4th/5th year) or equiv. 38 - Basic Education 2nd Cycle (6th/7th/8th Year) or Equiv. 39 - Technological

Name	Role	Type	Data Type	Description
Father's qualification	Feature	Discrete	int	<p>specialization course 40 - Higher education - degree (1st cycle) 41 - Specialized higher studies course 42 - Professional higher technical course 43 - Higher Education - Master (2nd cycle) 44 - Higher Education - Doctorate (3rd cycle)</p> <p>1 - Secondary Education - 12th Year of Schooling or Eq. 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 11 - 7th Year (Old) 12 - Other - 11th Year of Schooling 13 - 2nd year complementary high school course 14 - 10th Year of Schooling 18 - General commerce course 19 - Basic Education 3rd Cycle (9th/10th/11th Year) or Equiv. 20 - Complementary High School Course 22 - Technical-professional course 25 - Complementary High School Course - not concluded 26 - 7th year of schooling 27 - 2nd cycle of the general high school course 29 - 9th Year of Schooling - Not Completed 30 - 8th year of schooling 31 - General Course of Administration and Commerce 33 - Supplementary Accounting and Administration 34 - Unknown 35 - Can't read or write 36 - Can read without having a 4th year of schooling 37 - Basic education 1st cycle (4th/5th year) or equiv. 38 - Basic Education 2nd Cycle (6th/7th/8th Year) or Equiv. 39 - Technological specialization course 40 - Higher education - degree (1st cycle) 41 - Specialized higher studies course 42 - Professional higher technical course 43 - Higher Education - Master (2nd cycle) 44 - Higher Education - Doctorate (3rd cycle)</p>
Mother's occupation	Feature	Discrete	int	<p>0 – Student 1 – Representatives of the Legislative Power and Executive Bodies, Directors, Directors and Executive Managers 2 – Specialists in Intellectual and Scientific Activities 3 – Intermediate Level Technicians and Professions 4 – Administrative staff 5 – Personal Services, Security and Safety Workers and Sellers 6 – Farmers and Skilled Workers in Agriculture, Fisheries and Forestry 7 – Skilled Workers in Industry, Construction and Craftsmen 8 – Installation and Machine Operators and Assembly Workers 9 – Unskilled Workers 10 – Armed Forces Professions 90 – Other Situation 99– (blank) 122 – Health professionals 123 - teachers 125 - Specialists in information and communication technologies (ICT) 131 - Intermediate level science and engineering technicians and professions 132 - Technicians and professionals, of intermediate level of health 134 - Intermediate level technicians from legal, social, sports, cultural and similar services 141 - Office workers, secretaries in general and data processing operators 143 - Data, accounting, statistical, financial services and registry-related operators 144 - Other administrative support staff 151 - personal service workers 152 - sellers 153 - Personal care workers and the like 171 - Skilled construction workers and the like, except electricians 173 - Skilled workers in printing, precision instrument manufacturing, jewelers, artisans and the like 175 - Workers in food processing, woodworking, clothing and other industries and crafts 191 - cleaning workers 192 - Unskilled workers in agriculture, animal production, fisheries and forestry 193 - Unskilled workers in extractive industry, construction, manufacturing and transport 194 - Meal preparation assistants</p>
Father's occupation	Feature	Discrete	int	<p>0 – Student 1 – Representatives of the Legislative Power and Executive Bodies, Directors, Directors and Executive Managers 2 – Specialists in Intellectual and Scientific Activities 3 –Intermediate Level Technicians and Professions 4 – Administrative staff 5 – Personal Services, Security and Safety Workers and Sellers 6 – Farmers and Skilled Workers in Agriculture, Fisheries and Forestry 7 – Skilled Workers in Industry, Construction and Craftsmen 8 –</p>

Name	Role	Type	Data Type	Description
				Installation and Machine Operators and Assembly Workers 9 – Unskilled Workers 10 – Armed Forces Professions 90 – Other Situation 99 – (blank) 101 – Armed Forces Officers 102 – Armed Forces Sergeants 103 – Other Armed Forces personnel 112 – Directors of administrative and commercial services 114 – Hotel, catering, trade and other services directors 121 - Specialists in the physical sciences, mathematics, engineering and related techniques 122 - Health professionals 123 - teachers 124 - Specialists in finance, accounting, administrative organization, public and commercial relations 131 - Intermediate level science and engineering technicians and professions 132 - Technicians and professionals, of intermediate level of health 134 - Intermediate level technicians from legal, social, sports, cultural and similar services 135 - Information and communication technology technicians 141 - Office workers, secretaries in general and data processing operators 143 - Data, accounting, statistical, financial services and registry-related operators 144 - Other administrative support staff 151 - personal service workers 152 - sellers 153 - Personal care workers and the like 154 - Protection and security services personnel 161 - Market-oriented farmers and skilled agricultural and animal production workers 163 - Farmers, livestock keepers, fishermen, hunters and gatherers, subsistence 171 - Skilled construction workers and the like, except electricians 172 - Skilled workers in metallurgy, metalworking and similar 174 - Skilled workers in electricity and electronics 175 - Workers in food processing, woodworking, clothing and other industries and crafts 181 - Fixed plant and machine operators 182 - assembly workers 183 - Vehicle drivers and mobile equipment operators 192 - Unskilled workers in agriculture, animal production, fisheries and forestry 193 - Unskilled workers in extractive industry, construction, manufacturing and transport 194 - Meal preparation assistants 195 - Street vendors (except food) and street service providers
Admission grade	Feature	Continuous	float	Admission grade (between 0 and 200)
Displaced	Feature	Discrete	int	1 — yes 0 — no
Educational special needs	Feature	Discrete	int	1 — yes 0 — no
Debtor	Feature	Discrete	int	1 — yes 0 — no
Tuition fees up to date	Feature	Discrete	int	1 — yes 0 — no
Gender	Feature	Discrete	int	1 — male 0 — female
Scholarship holder	Feature	Discrete	int	1 — yes 0 — no
Age at enrollment	Feature	Discrete	int	Age of student at enrollment
Scholarship holder	Feature	Discrete	int	1 — yes 0 — no
International	Feature	Discrete	int	1 — yes 0 — no

Name	Role	Type	Data Type	Description
Curricular units 1st sem (credited)	Feature	Discrete	int	Number of curricular units credited in the 1st semester
Curricular units 1st sem (enrolled)	Feature	Discrete	int	Number of curricular units enrolled in the 1st semester
Curricular units 1st sem (evaluations)	Feature	Discrete	int	Number of evaluations to curricular units in the 1st semester
Curricular units 1st sem (approved)	Feature	Discrete	int	Number of curricular units approved in the 1st semester
Curricular units 1st sem (grade)	Feature	Discrete	float	Grade average in the 1st semester (between 0 and 20)
Curricular units 1st sem (without evaluations)	Feature	Discrete	int	Number of curricular units without evaluations in the 1st semester
Curricular units 2nd sem (credited)	Feature	Discrete	int	Number of curricular units credited in the 2nd semester
Curricular units 2nd sem (enrolled)	Feature	Discrete	int	Number of curricular units enrolled in the 2nd semester
Curricular units 2nd sem (evaluations)	Feature	Discrete	int	Number of evaluations to curricular units in the 2nd semester
Curricular units 2nd sem (approved)	Feature	Discrete	int	Number of curricular units approved in the 2nd semester
Curricular units 2nd sem (grade)	Feature	Discrete	float	Grade average in the 2nd semester (between 0 and 20)
Curricular units 2nd sem (without evaluations)	Feature	Discrete	int	Number of curricular units without evaluations in the 1st semester
Unemployment	Feature	Continuous	float	Unemployment rate (%)

Name	Role	Type	Data Type	Description
rate				
Inflation rate	Feature	Continuous	float	Inflation rate (%)
GDP	Feature	Continuous	float	GDP
Target	Target	Categorical	string	Target. The problem is formulated as a three category classification task (dropout, enrolled, and graduate) at the end of the normal duration of the course

Obtain

Import libraries and Visualization Packages

Importing libraries at the beginning allows access to modules and other tools throughout this project that help to make the tasks within this project manageable to implement. The main libraries that will be used within this project include:

- `pandas` : a data analysis and manipulation library which allows for flexible reading, writing, and reshaping of data
- `numpy` : a key library that brings the computationaly power of languages like C to Python
- `matplotlib` : a comprehensive visualization library
- `seaborn` : a data visualization library based on matplotlib
- `statsmodels` : a library used to understand statistical relationships between variables, often used in the field of economics.
- `sklearn` : a machine learning libary for data processing and modeling.

```
In [1]: # Import Libraries and visualization packages
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import statsmodels.api as sm
import sklearn
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer, recall_score
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.feature_selection import RFE
from sklearn.preprocessing import MinMaxScaler

import warnings
warnings.filterwarnings("ignore")
warnings.simplefilter(action = 'ignore', category = DeprecationWarning)
warnings.simplefilter(action = 'ignore', category = FutureWarning)
warnings.filterwarnings("ignore", category=UserWarning, module="pandas")

# Allow plots to display and be stored inline within a notebook
%matplotlib inline

# Used for working with the z-score
from scipy import stats

# Used for working with long url
from urllib.parse import urlencode

# Set display option to readable format
pd.set_option('display.float_format', lambda x: '%.2f' % x)

# Filter warnings from pandas
pd.options.mode.chained_assignment = None
```

Check library versions:

One of the important parts of modeling is the ability to reproduce results. Science is indeed model dependent and those models improve as more information is gathered over time. Communicating the version of the tools that were used to generate a model is an important part of collaboration. So here we take note of the version of key libraries so that other researchers can have a place to start when attempting to reproduce this work.

```
In [2]: # Check Pandas version
print ("Pandas version:")
pd.__version__
```

```
Pandas version:  
Out[2]: '1.1.3'
```

```
In [3]: # Check Numpy version  
print("Numpy version:")  
np.__version__
```

```
Numpy version:  
Out[3]: '1.24.4'
```

```
In [4]: # Check Seaborn version  
print("Seaborn version:")  
sns.__version__
```

```
Seaborn version:  
Out[4]: '0.11.0'
```

```
In [5]: # Check Statsmodel version  
print("Statsmodel version:")  
sm.__version__
```

```
Statsmodel version:  
Out[5]: '0.13.5'
```

```
In [6]: # Check sci-kit Learn version  
print("Sci-kit learn version:")  
sklearn.__version__
```

```
Sci-kit learn version:  
Out[6]: '1.3.0'
```

```
In [7]: class ObtainData:  
    def __init__(self, data_path):  
        """  
        Constructor for the ObtainData class.  
  
        Parameters:  
        - data_path(string): Path to the dataset file.  
  
        Initializes the data_path attribute and sets data to None.  
        """
```

```
    self.data_path = data_path
    self.data = None

def load_data(self):
    """
    Loads the dataset from the specified data_path using pandas.
    Assumes the dataset is in csv format with semicolons(;) as the separator.
    Assigns the loaded data to the data attribute.
    """
    self.data = pd.read_csv(self.data_path, sep = ';')

def obtain_data(self):

    self.load_data()
    return self.data
```

```
In [8]: # Main excution

if __name__ == '__main__':
    # Initialize the ObtainData object
    data_obtained = ObtainData(data_path = 'data/data.csv')

    # Obtain the data
    data_obtained.obtain_data()
```

Scrub

```
In [9]: class ScrubData:
    def __init__(self, data):
        """
        Constructor for the ScrubData class.

        Parameters:
        - data (pandas DataFrame): The dataset to be explored

        Initializes the data attribute.
        """

        self.data = data
```

```
def check_placeholders(self):
    """
    Checks and displays the presence of placeholders throughout the entire dataframe.
    Placeholders checked: '?', '#', 'NaN', 'null', 'N/A', '-'.
    """
    placeholders = ['?', '#', 'NaN', 'null', 'N/A', '-']
    placeholder_mask = self.data.isin(placeholders)

    if placeholder_mask.any().any():
        print('\nPlaceholders Detected:')
        display(placeholder_mask)
    else:
        print('\nNo Placeholders Detected.')

def clean(self):
    """
    Perform data cleaning operations.
    """
    cleaned_data = self.data.dropna() # Drop rows with missing data

    return cleaned_data
```

```
In [10]: # Create an instance of the Obtained Data Class
data = data_obtained.obtain_data()

scrubbed_data = ScrubData(data = data)

cleaned_data = scrubbed_data.clean()
```

Explore

```
In [11]: class ExploreData:
    def __init__(self,data):
        """
        Constructor for the ExploreData class.

        Parameters:
        - data (pandas DataFrame): The dataset to be explored.

        Initializes the data attribute
        """
        self.data = data
```

```
def examine_structure(self):
    """
    Prints the shape of the dataset and displays the first few rows.
    Also displays the data information including the number of rows and columns in the dataset.
    """
    print('Data Shape:')
    display(self.data.shape)

    print('Data Structure:')
    display(self.data.head())

    print('\nData Information:')
    display(self.data.info())

def check_duplicates(self):
    """
    Checks and displays the number of duplicate rows in the dataset.
    """
    num_duplicates = self.data.duplicated().sum()
    print(f'\nNumber of Duplicate Rows: {num_duplicates}')

def drop_duplicates(self):
    """
    Drops the duplicate rows from the dataset.
    """
    self.data.drop_duplicates(inplace = True)
    print("Duplicate rows dropped.")

def generate_correlation_map(self, title):
    """
    Generates and displays a correlation matrix heatmap for the dataset.

    Parameters:
    -title(str): The title of the correlation map
    """
    correlation_matrix = self.data.corr()

    # Create a figure and set the title
    plt.figure(figsize = (30,25))
    plt.suptitle(title)

    # Generate the correlation heatmap
    sns.heatmap(correlation_matrix, annot = True, cmap = 'Blues')
```

```

# Show the correlation map
plt.show()

def plot_pairplot(self, correlation_table):
    # Get the columns from the correlation table
    columns = list(correlation_table.index.levels[0])

    # Filter the data based on the columns
    data_filtered = self.data[columns]

    # Create a pairplot using the filtered data
    pairplot = sns.pairplot(data_filtered, diag_kind="kde", markers=".",
                           height=2)

    # Customize the scatter matrix plot
    for ax in pairplot.axes.flat:
        ax.xaxis.label.set_rotation(90)
        ax.yaxis.label.set_rotation(0)
        ax.yaxis.label.set_ha('center')
        ax.get_yaxis().set_label_coords(-0.5, 0.5)
        ax.set_xticklabels(ax.get_xticklabels(), rotation=45, ha='right',
                           fontsize=18)
        ax.set_yticklabels(ax.get_yticklabels(), rotation=90, ha='center',
                           fontsize=18)

    # Show the plot
    plt.tight_layout()
    plt.show()

def generate_correlation_table(self, threshold_min = 0.6, threshold_max = 1.0):
    """
    Generates a correlation table with filtered correlation pairs within specified thresholds.

    Parameters:
    - threshold_min (float): The minimum threshold for correlation values (inclusive). Default is 0.6.
    - threshold_max (float): The maximum threshold for correlation values (exclusive). Default is 1.0.

    Returns:
    - filtered_df (pandas DataFrame): The filtered correlation table.
    """
    correlation_matrix = self.data.corr().abs()

    correlation_df = correlation_matrix.stack().reset_index()
    correlation_df.columns = ['feature1', 'feature2', 'correlation']

```

```

        filtered_df = correlation_df[
            (correlation_df['correlation'] > threshold_min) & (correlation_df['correlation'] < threshold_max)
        ]

        filtered_df.drop_duplicates(inplace = True)
        filtered_df.set_index(['feature1', 'feature2'], inplace = True)
        filtered_df.columns = ['correlation']

    return filtered_df

def plot_gender_distribution(self):
    """
    Plots the gender distribution from the dataset.
    """
    plt.style.use('fivethirtyeight')
    gender_counts = self.data['Gender'].value_counts()
    gender_labels = ['Female', 'Male']
    gender_values = [gender_counts[0], gender_counts[1]]

    plt.figure(figsize = (8,6))
    ax = sns.barplot(x = gender_labels, y = gender_values, color = 'blue')

    plt.xlabel('Gender', fontsize = 14, weight = 'bold')
    plt.ylabel('Number of Students', fontsize = 14, weight = 'bold')
    plt.title('Gender Distribution of College Students',fontsize = 18, weight = 'bold')

    plt.show()

def plot_dropout_by_gender(self):
    """
    Plots the gender distribution of students who dropout from the dataset.
    """
    plt.style.use('fivethirtyeight')
    dropout_data = self.data[self.data['Target']=='Dropout']
    dropout_data['Gender']= dropout_data['Gender'].map({1:'Male', 0:'Female'})

    plt.figure(figsize = (8,6))
    ax = sns.countplot(x = 'Gender', data = dropout_data, color = 'blue')

    plt.xlabel('Gender', fontsize = 14, weight = 'bold')
    plt.ylabel('Number of Students', fontsize = 14, weight = 'bold')
    plt.title('Distribution of Dropouts by Gender',fontsize = 18, weight = 'bold')

```

```
plt.show()

def plot_target_distribution(self):
    """
    Plots the gender distribute of the target variable from the dataset.
    """

    target_data = self.data['Target']
    target_labels = ['Dropout', 'Enrolled', 'Graduate']

    plt.figure(figsize = (8,6))
    ax = sns.countplot(x = target_labels, data = target_data, color = 'blue')

    plt.xlabel('Status', fontsize = 14, weight = 'bold')
    plt.ylabel('Percentage of Students', fontsize = 14, weight = 'bold')
    plt.title('Progress to Graduation Distribution of College Students', fontsize = 18, weight = 'bold')

    plt.show()
```

```
In [12]: # Review correlations on dataset

data_exploration = ExploreData(data = cleaned_data)
data_exploration.examine_structure()
data_exploration.generate_correlation_map("Correlation Between Student Academic Success Features")
```

Data Shape:

(4424, 37)

Data Structure:

Marital status	Application mode	Application order	Course	Daytime/evening attendance\l	Previous qualification	Previous qualification (grade)	Nacionality	Mother's qualification	Father's qualification	Curricu... units & s (credit)
0	1	17	5	171	1	1	122.00	1	19	12 ...
1	1	15	1	9254	1	1	160.00	1	1	3 ...
2	1	1	5	9070	1	1	122.00	1	37	37 ...
3	1	17	2	9773	1	1	122.00	1	38	37 ...
4	2	39	1	8014	0	1	100.00	1	37	38 ...

5 rows × 37 columns

Data Information:

<class 'pandas.core.frame.DataFrame'>

Int64Index: 4424 entries, 0 to 4423

Data columns (total 37 columns):

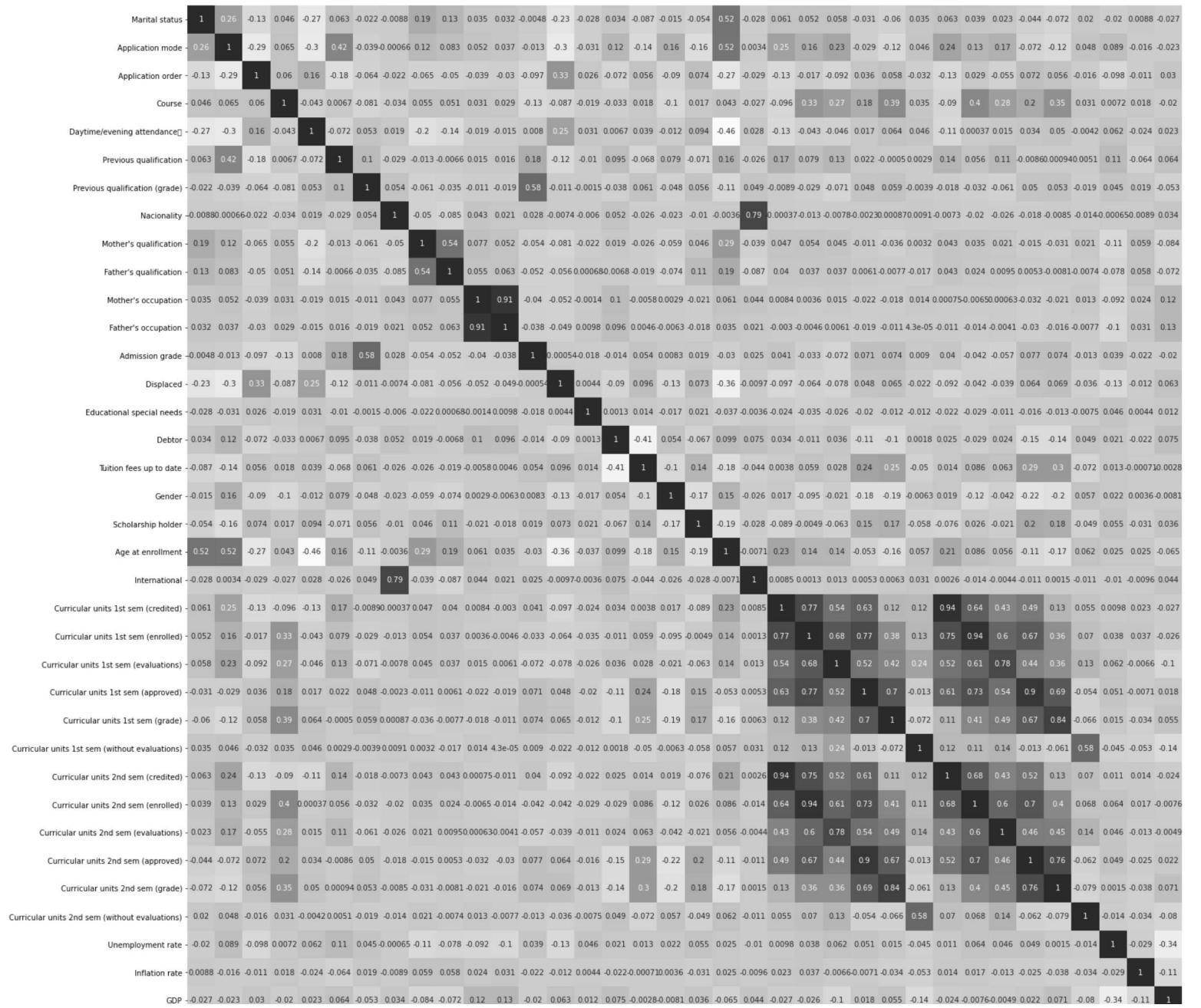
#	Column	Non-Null Count	Dtype
0	Marital status	4424 non-null	int64
1	Application mode	4424 non-null	int64
2	Application order	4424 non-null	int64
3	Course	4424 non-null	int64
4	Daytime/evening attendance	4424 non-null	int64
5	Previous qualification	4424 non-null	int64
6	Previous qualification (grade)	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	Debtors	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: 1.3+ MB

None

Correlation Between Student Academic Success Features



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 (evaluated) -	
Curricular units 1st sem (grade) -	
Curricular units 1st sem (without evaluations) -	
Curricular units 2nd sem (approved) -	
Curricular units 2nd sem (credited) -	
Curricular units 2nd sem (enrolled) -	
Curricular units 2nd sem (evaluated) -	
Curricular units 2nd sem (grade) -	
Curricular units 2nd sem (without evaluations) -	
Unemployment rate -	
Inflation rate -	
GDP -	

This seems complicated, so let's simplify it to something that is a bit more useful.

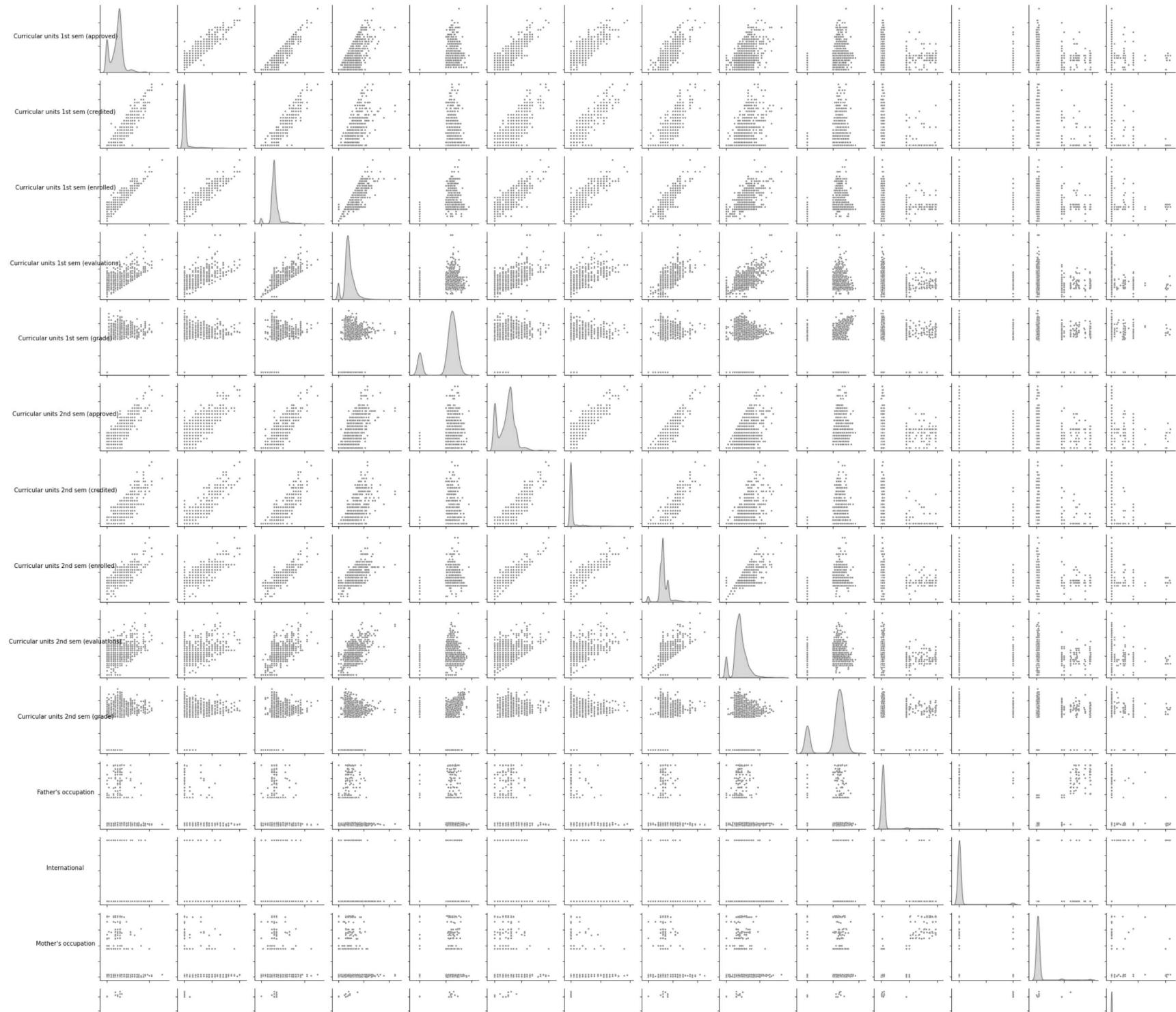
```
In [13]: correlation_table = data_exploration.generate_correlation_table(0.6, 1.0)
display(correlation_table)
```

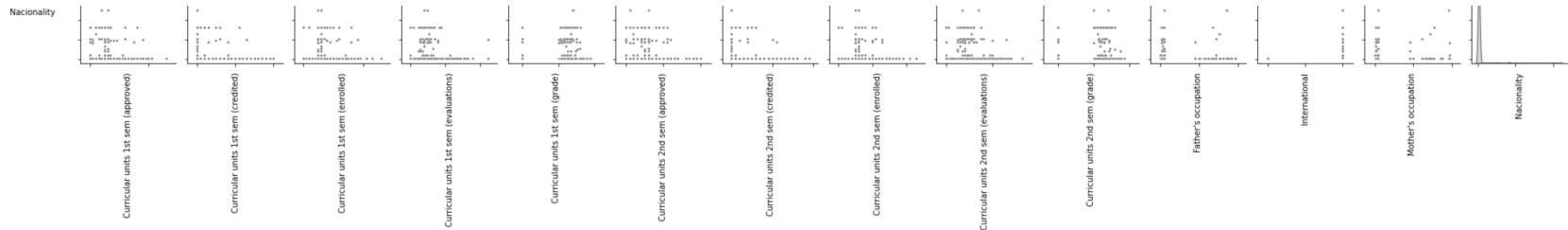
		correlation
feature1	feature2	
Nacionality	International	0.79
Mother's occupation	Father's occupation	0.91
Father's occupation	Mother's occupation	0.91
International	Nacionality	0.79
Curricular units 1st sem (credited)	Curricular units 1st sem (enrolled)	0.77
	Curricular units 1st sem (approved)	0.63
	Curricular units 2nd sem (credited)	0.94
	Curricular units 2nd sem (enrolled)	0.64
Curricular units 1st sem (enrolled)	Curricular units 1st sem (credited)	0.77
	Curricular units 1st sem (evaluations)	0.68
	Curricular units 1st sem (approved)	0.77
	Curricular units 2nd sem (credited)	0.75
	Curricular units 2nd sem (enrolled)	0.94
	Curricular units 2nd sem (approved)	0.67
Curricular units 1st sem (evaluations)	Curricular units 1st sem (enrolled)	0.68
	Curricular units 2nd sem (enrolled)	0.61
	Curricular units 2nd sem (evaluations)	0.78
Curricular units 1st sem (approved)	Curricular units 1st sem (credited)	0.63
	Curricular units 1st sem (enrolled)	0.77
	Curricular units 1st sem (grade)	0.70
	Curricular units 2nd sem (credited)	0.61
	Curricular units 2nd sem (enrolled)	0.73
	Curricular units 2nd sem (approved)	0.90
	Curricular units 2nd sem (grade)	0.69

		correlation
feature1	feature2	
Curricular units 1st sem (grade)	Curricular units 1st sem (approved)	0.70
	Curricular units 2nd sem (approved)	0.67
	Curricular units 2nd sem (grade)	0.84
Curricular units 2nd sem (credited)	Curricular units 1st sem (credited)	0.94
	Curricular units 1st sem (enrolled)	0.75
	Curricular units 1st sem (approved)	0.61
	Curricular units 2nd sem (enrolled)	0.68
Curricular units 2nd sem (enrolled)	Curricular units 1st sem (credited)	0.64
	Curricular units 1st sem (enrolled)	0.94
	Curricular units 1st sem (evaluations)	0.61
	Curricular units 1st sem (approved)	0.73
	Curricular units 2nd sem (credited)	0.68
	Curricular units 2nd sem (evaluations)	0.60
	Curricular units 2nd sem (approved)	0.70
Curricular units 2nd sem (evaluations)	Curricular units 1st sem (evaluations)	0.78
	Curricular units 2nd sem (enrolled)	0.60
Curricular units 2nd sem (approved)	Curricular units 1st sem (enrolled)	0.67
	Curricular units 1st sem (approved)	0.90
	Curricular units 1st sem (grade)	0.67
	Curricular units 2nd sem (enrolled)	0.70
	Curricular units 2nd sem (grade)	0.76
Curricular units 2nd sem (grade)	Curricular units 1st sem (approved)	0.69
	Curricular units 1st sem (grade)	0.84
	Curricular units 2nd sem (approved)	0.76

In [14]:

```
%time  
# Plot pairplot using the correlation table  
data_exploration.plot_pairplot(correlation_table)
```



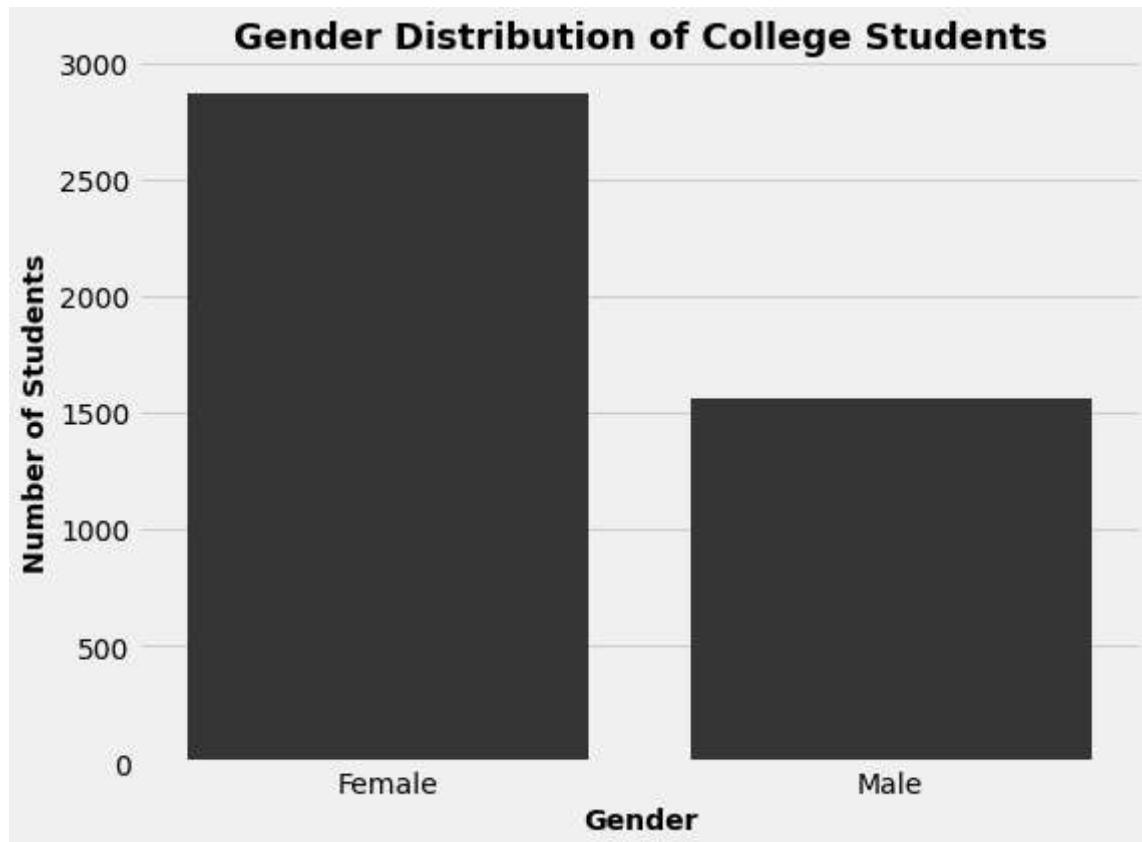


Wall time: 1min 53s

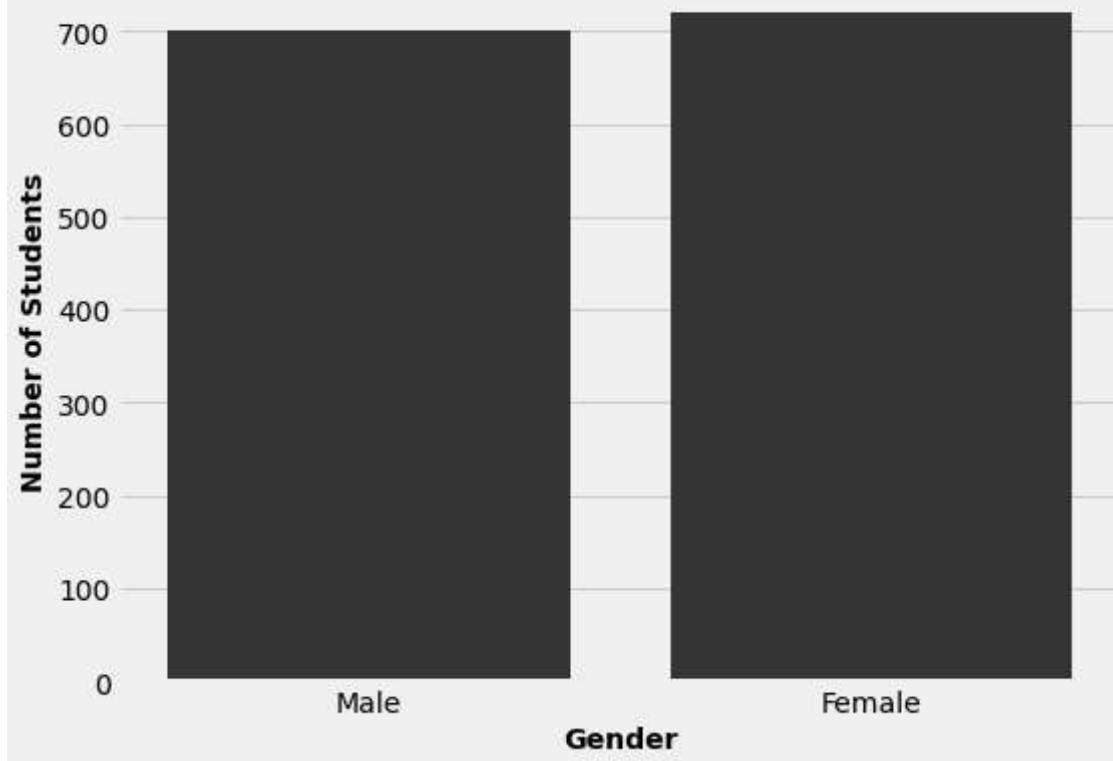
```
In [15]: if __name__ == '__main__':
    # Plot the gender distribution of the data
    data_exploration.plot_gender_distribution()

    # Plot the dropout distribution by gender
    data_exploration.plot_dropout_by_gender()

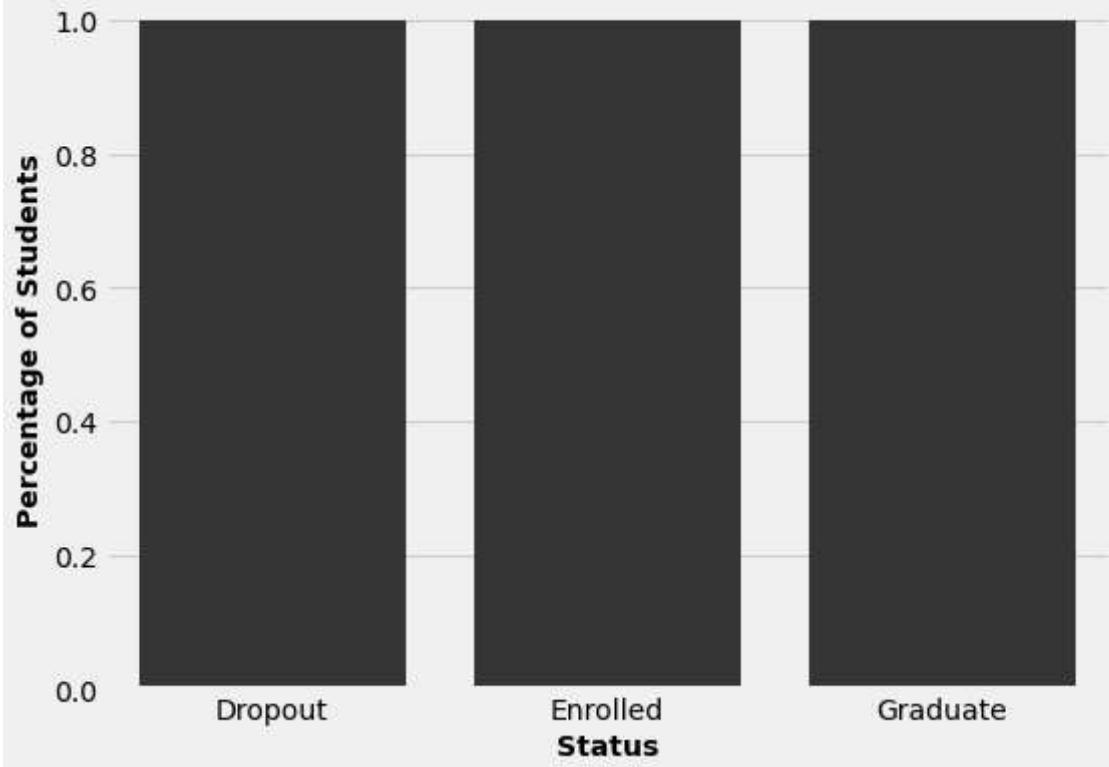
    # Plot the Progress to graduation distribution by gender
    data_exploration.plot_target_distribution()
```



Distribution of Dropouts by Gender



Progress to Graduation Distribution of College Students



```
In [30]: class AnalyzeData:  
    def __init__(self, data):  
        self.data = data  
  
    def analyze(self):  
        # Map the revised target variable values to their corresponding Labels  
        target_labels = {  
            "Enrolled": "Enrolled",  
            "Graduate": "Graduated",  
            "Dropout": "Dropout"  
        }  
  
        # Map the revised "Tuition fees up to date" variable values to their corresponding Labels  
        tuition_fees_labels = {  
            1: "Yes",  
            0: "No"  
        }
```

```
# Map the revised "Marital status" variable values to their corresponding Labels
marital_status_labels = {
    1: "Single",
    2: "Married",
    3: "Widower",
    4: "Divorced",
    5: "Facto Union",
    6: "Legally Separated"
}

# Map the revised "Marital status" variable values to their corresponding Labels
attendance_labels = {
    1: "daytime",
    0: "evening",
}

# Map the revised "Debtor" variable values to their corresponding Labels
debtor_labels = {
    1: "Yes",
    0: "No"
}

# Map the revised "Scholarship holder" variable values to their corresponding Labels
scholarship_labels = {
    1: "Yes",
    0: "No"
}

# Map the revised "Previous qualification" variable values to their corresponding Labels
previous_qualification_labels = {
    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 equiv.",
    38: "Basic education 2nd cycle (6th/7th/8th year) or equiv.",
    39: "Technological specialization course",
    40: "Higher education - degree (1st cycle)",
    42: "Professional higher technical course",
}
```

```

        43: "Higher education - master (2nd cycle)"
    }

# Filter the data for relevant columns
relevant_columns = ['Marital status',
                     'Curricular units 1st sem (credited)',
                     'Curricular units 2nd sem (approved)',
                     'Tuition fees up to date',
                     'Curricular units 2nd sem (grade)',
                     "Debtors",
                     "Age at enrollment",
                     'Target',
                     'Scholarship holder',
                     'Displaced',
                     'Unemployment rate',
                     'Previous qualification']

data_subset = self.data[relevant_columns]

data_subset['Target'] = data_subset['Target'].map(target_labels)
data_subset['Marital status'] = data_subset['Marital status'].map(marital_status_labels)
data_subset['Tuition fees up to date'] = data_subset['Tuition fees up to date'].map(tuition_fees_labels)
data_subset['Debtors'] = data_subset['Debtors'].map(debtor_labels)
data_subset['Previous qualification'] = data_subset['Previous qualification'].map(previous_qualification_labels)
data_subset['Scholarship holder'] = data_subset['Scholarship holder'].map(scholarship_labels)

age_ranges = [16, 25, 36, 50, 100]
data_subset['Age Range'] = pd.cut(data_subset['Age at enrollment'], bins=age_ranges, labels=["16-24",
                                                                                      "25-35",
                                                                                      "36-49",
                                                                                      "50 or older"])

approved_units_bins = [-1, 3, 6, float('inf')]
approved_units_labels = ["0-3", "4-6", "7 or greater"]
data_subset['Approved Units Range'] = pd.cut(data_subset['Curricular units 2nd sem (approved)'],
                                              bins = approved_units_bins,
                                              labels = approved_units_labels)
data_subset['Credited Units'] = pd.cut(data_subset['Curricular units 1st sem (credited)'],
                                       bins = approved_units_bins,
                                       labels = approved_units_labels)

# Plot the results
self.plot_bar_chart(data_subset,
                     'Age Range',
                     'Influence of Age Range on Graduation Status',
                     'Number of Students',

```

```

        'Age Range')
self.plot_boxplot(data_subset,
                  'Curricular units 2nd sem (grade)',
                  'Influence of Curricular units 2nd sem (grade) on Graduation Status',
                  'Curricular units 2nd sem (grade)',
                  'Graduation Status')
self.plot_bar_chart(data_subset,
                    'Approved Units Range',
                    'Influence of Curricular units 2nd sem (approved) on Graduation Status',
                    'Number of Students',
                    'Curricular Units Range')

self.plot_bar_chart(data_subset,
                    'Credited Units',
                    'Influence of Curricular units 1st sem (credited) on Graduation Status',
                    'Number of Students',
                    'Curricular Units')

self.plot_bar_chart(data_subset,
                    'Scholarship holder',
                    'Influence of Scholarship Status on Graduation Status',
                    'Number of Students',
                    'Scholarship holder')
self.plot_bar_chart(data_subset,
                    'Tuition fees up to date',
                    'Influence of Tuition Fees on Graduation Status',
                    'Tuition Fees Up to Date',
                    'Number of Students')
self.plot_horizontal_bar_chart(data_subset,
                               'Previous qualification',
                               'Influence of Previous qualification on Graduation Status',
                               'Number of Students',
                               'Graduation Status')

def plot_bar_chart(self, data, x, title, xlabel, ylabel):
    plt.figure(figsize=(15, 10))
    sns.countplot(data=data, x=x, hue='Target', palette='cool')
    plt.legend(title="Graduation Status")
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title, fontsize=18)
    plt.show()

def plot_boxplot(self, data, x, title, xlabel, ylabel):
    plt.figure(figsize=(15, 10))

```

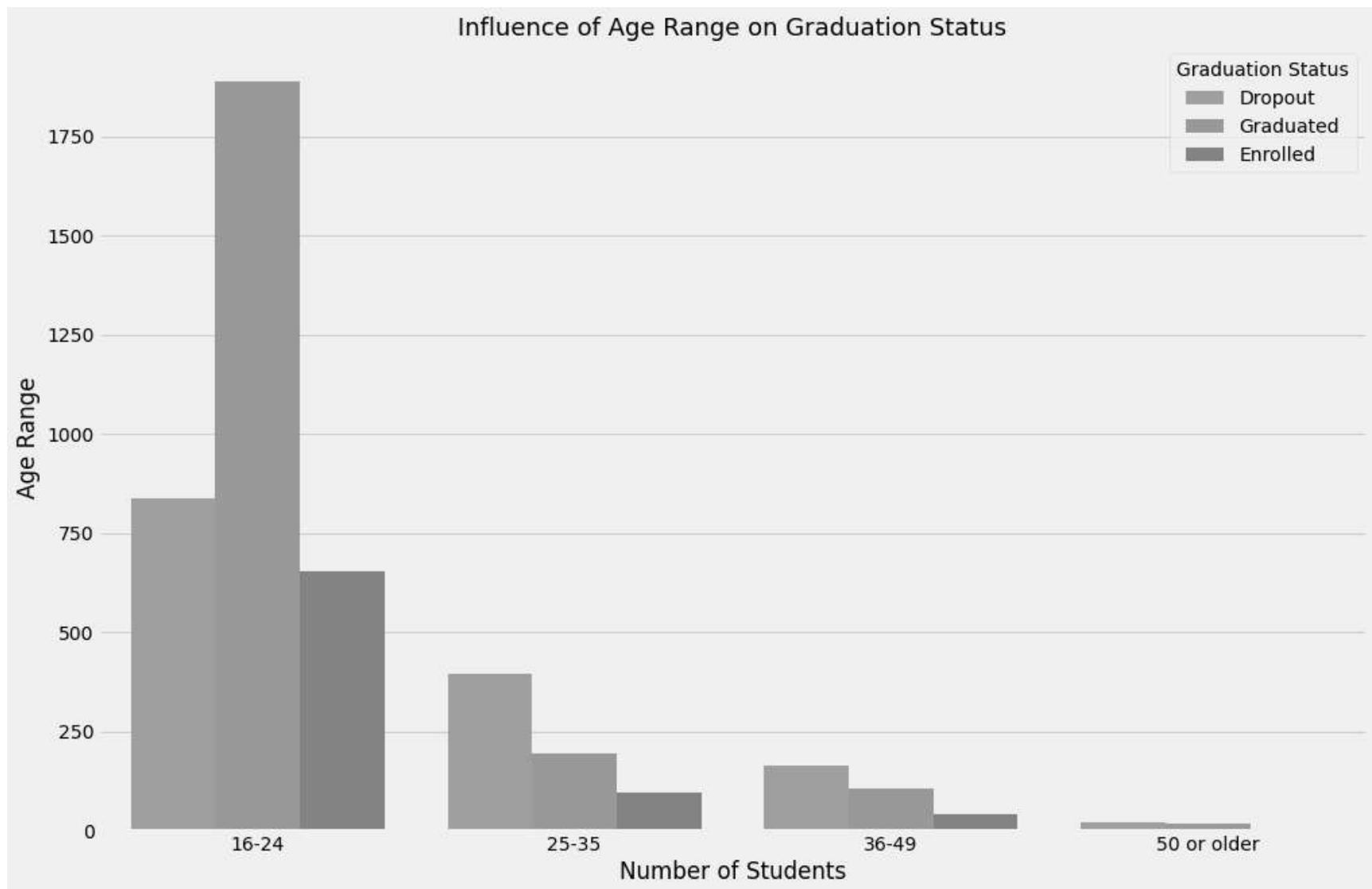
```
    sns.boxplot(data=data, x='Target', y=x, palette='cool')
    custom_legend = [plt.Line2D([], [], color=sns.color_palette('cool')[1], marker='s', markersize=10, label='Enrol'),
                     plt.Line2D([], [], color=sns.color_palette('cool')[3], marker='s', markersize=10, label='Gradu'),
                     plt.Line2D([], [], color=sns.color_palette('cool')[5], marker='s', markersize=10, label='Dropo
    plt.legend(handles=custom_legend, title="Graduation Status")
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title, fontsize=18)
    plt.show()

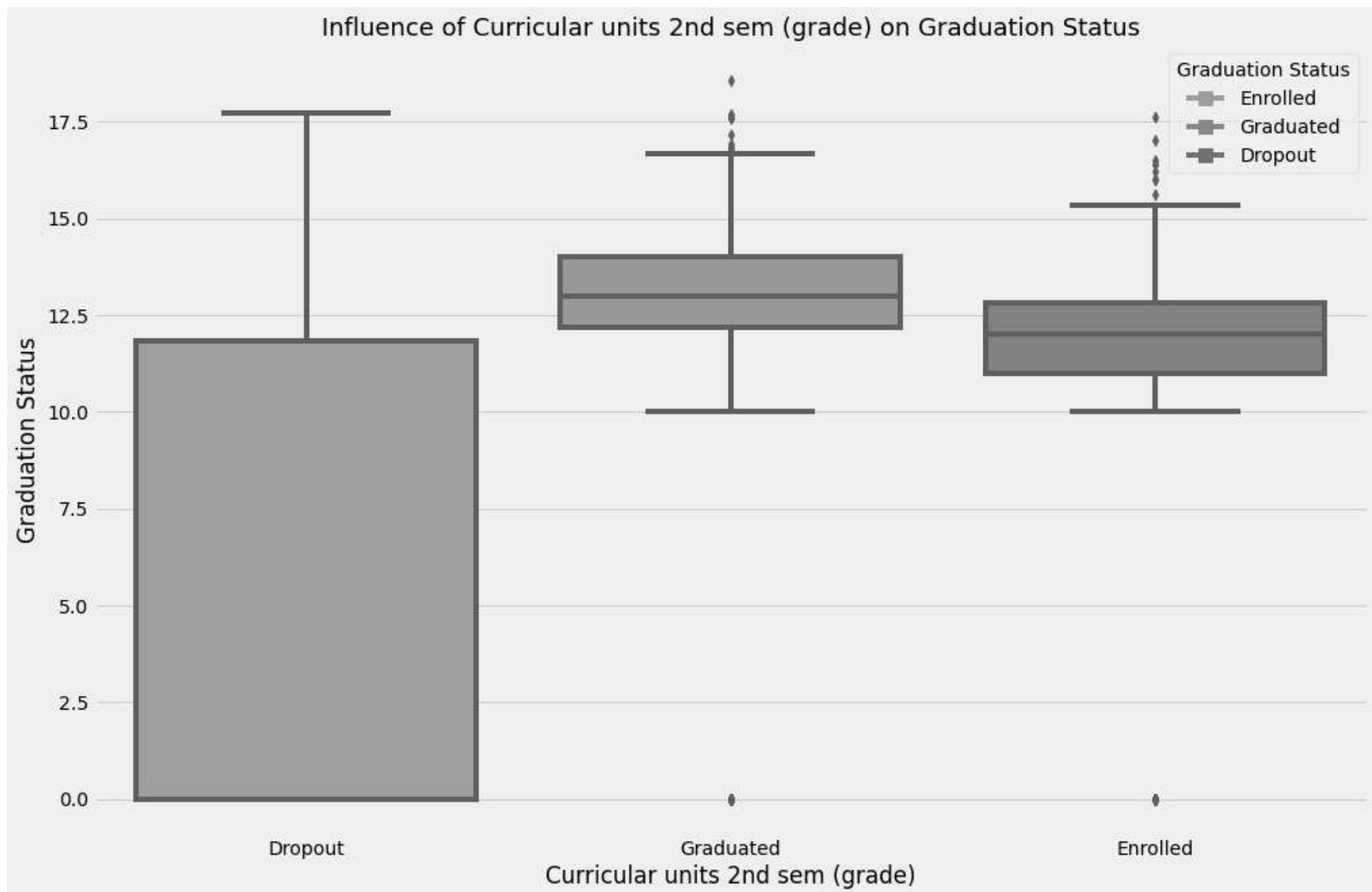
def plot_horizontal_bar_chart(self, data, x, title, xlabel, ylabel):
    plt.figure(figsize=(15, 10))
    sns.countplot(data=data, y=x, hue='Target', palette='cool')
    plt.legend(title="Graduation Status")
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title, fontsize=18)
    plt.show()

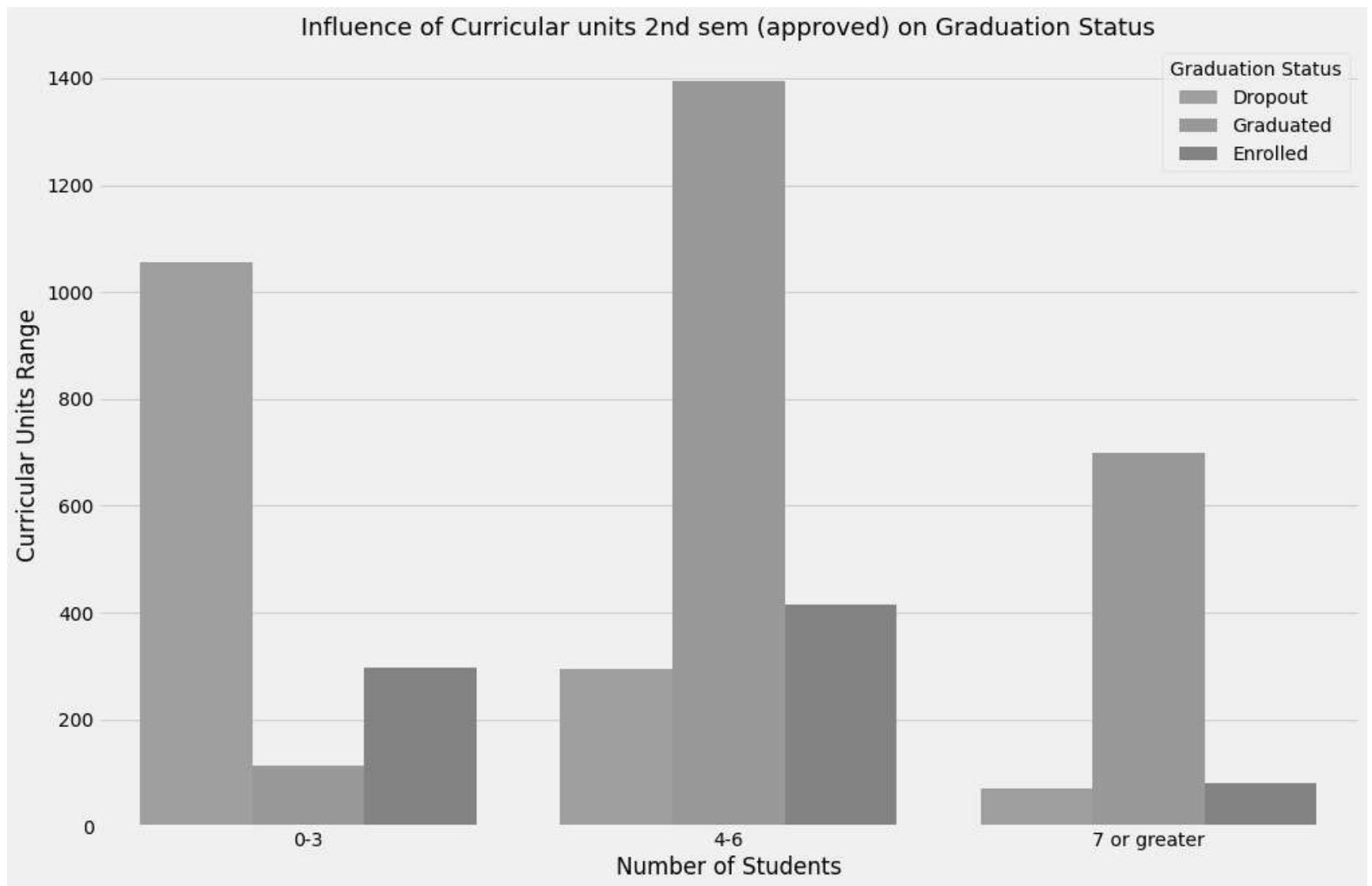
# Instantiate the AnalyzeData class with the data
analysis = AnalyzeData(data)

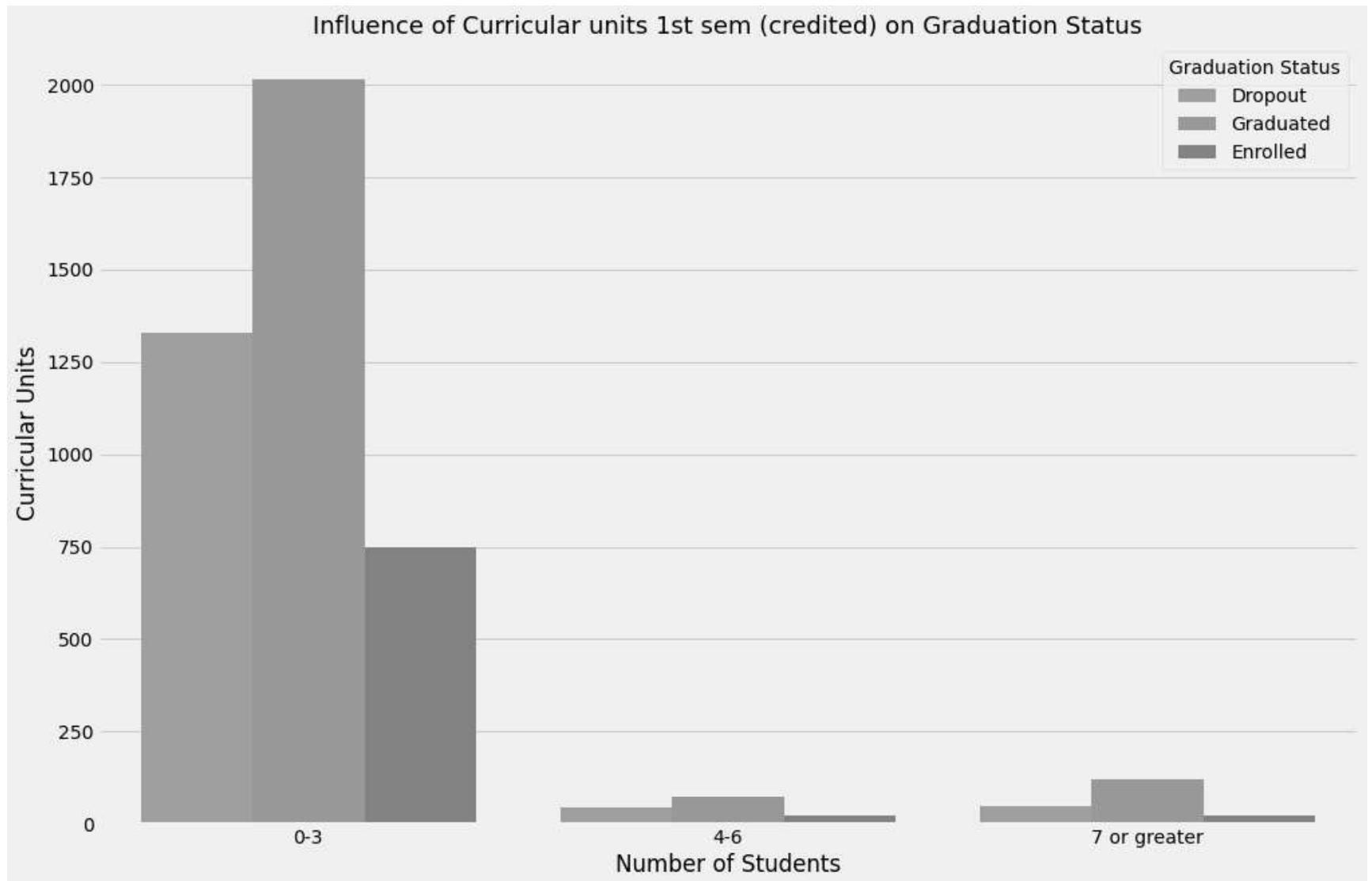
# Perform the analysis
analysis.analyze()
```

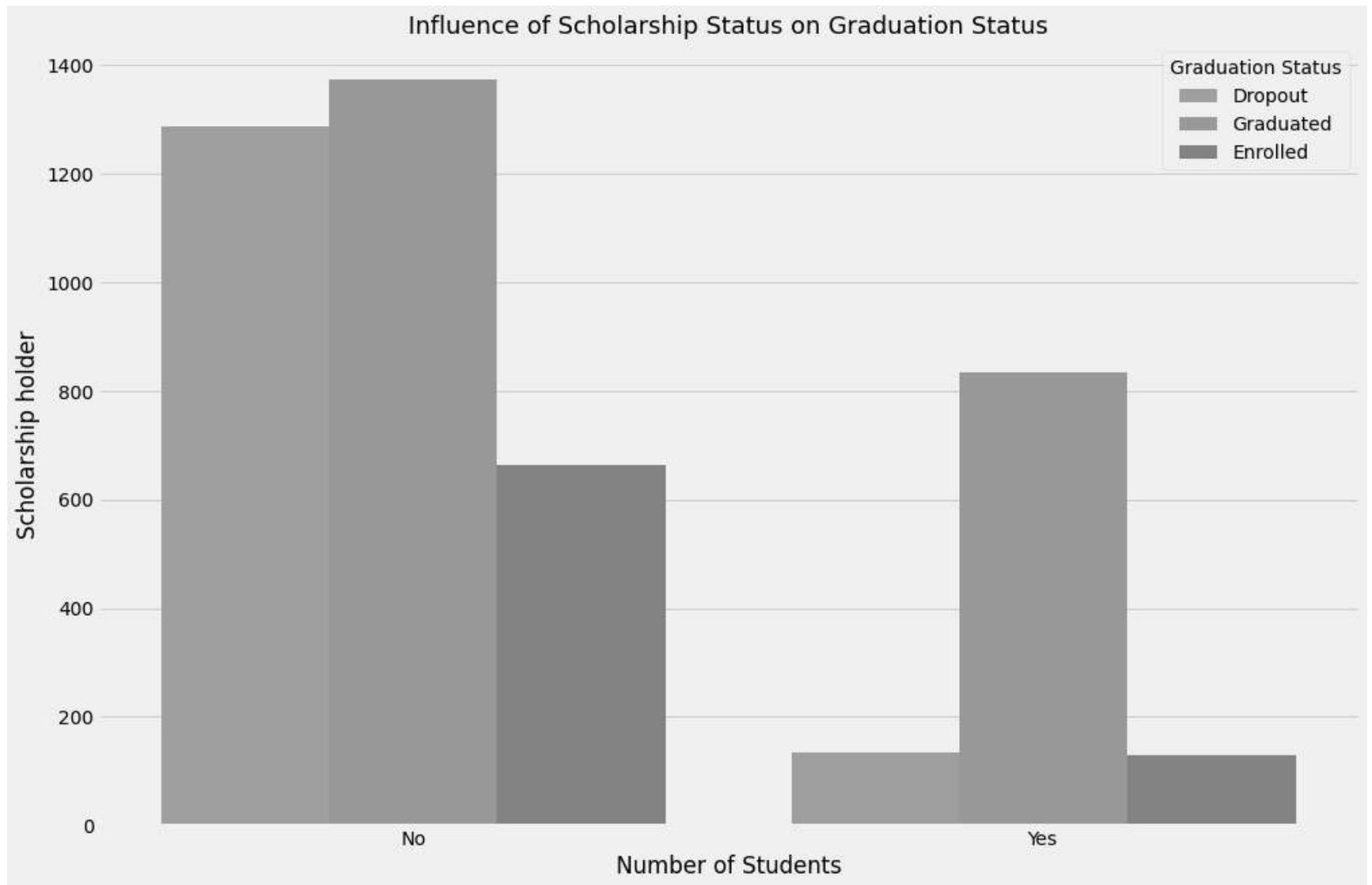
Influence of Age Range on Graduation Status



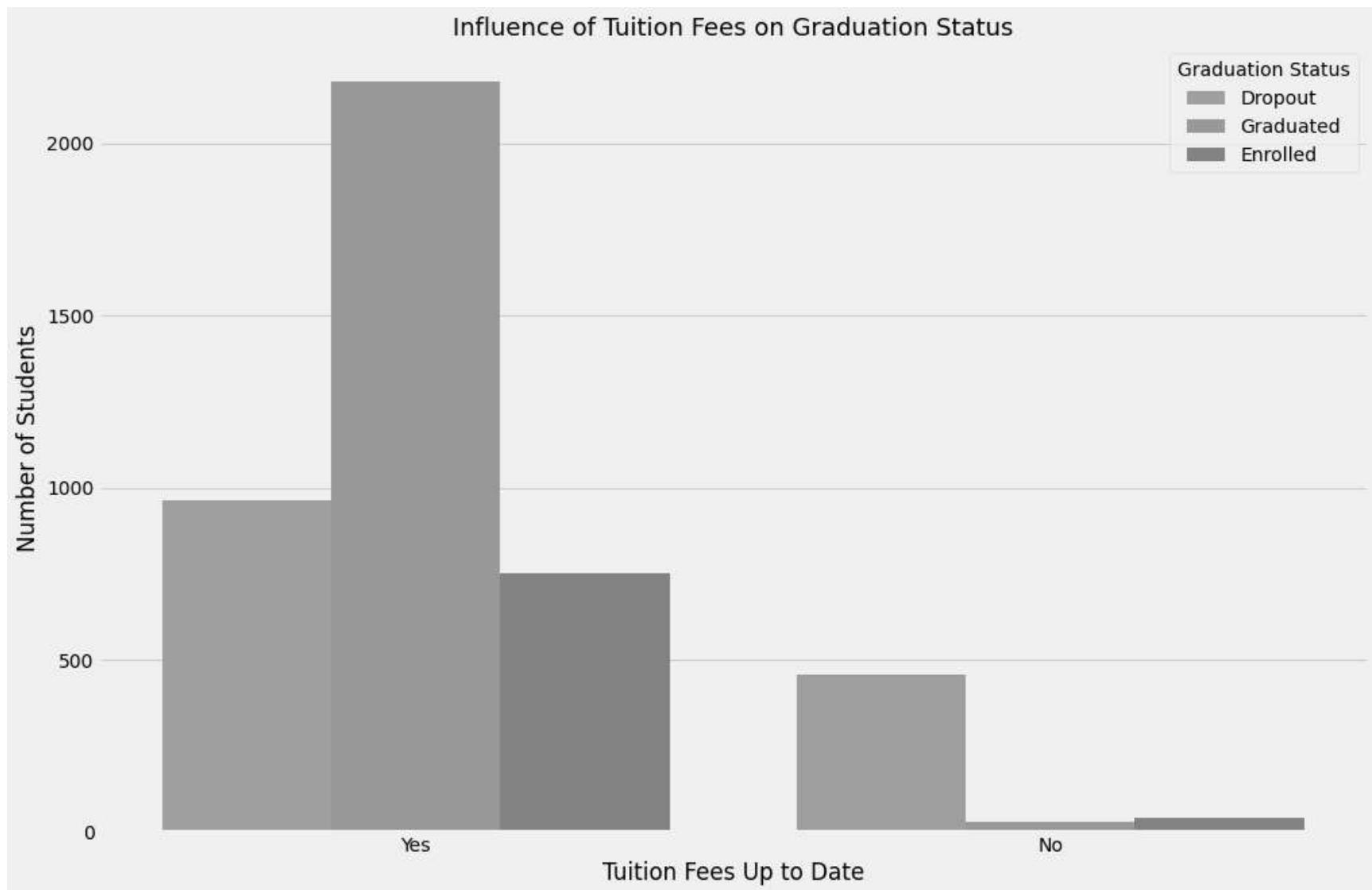


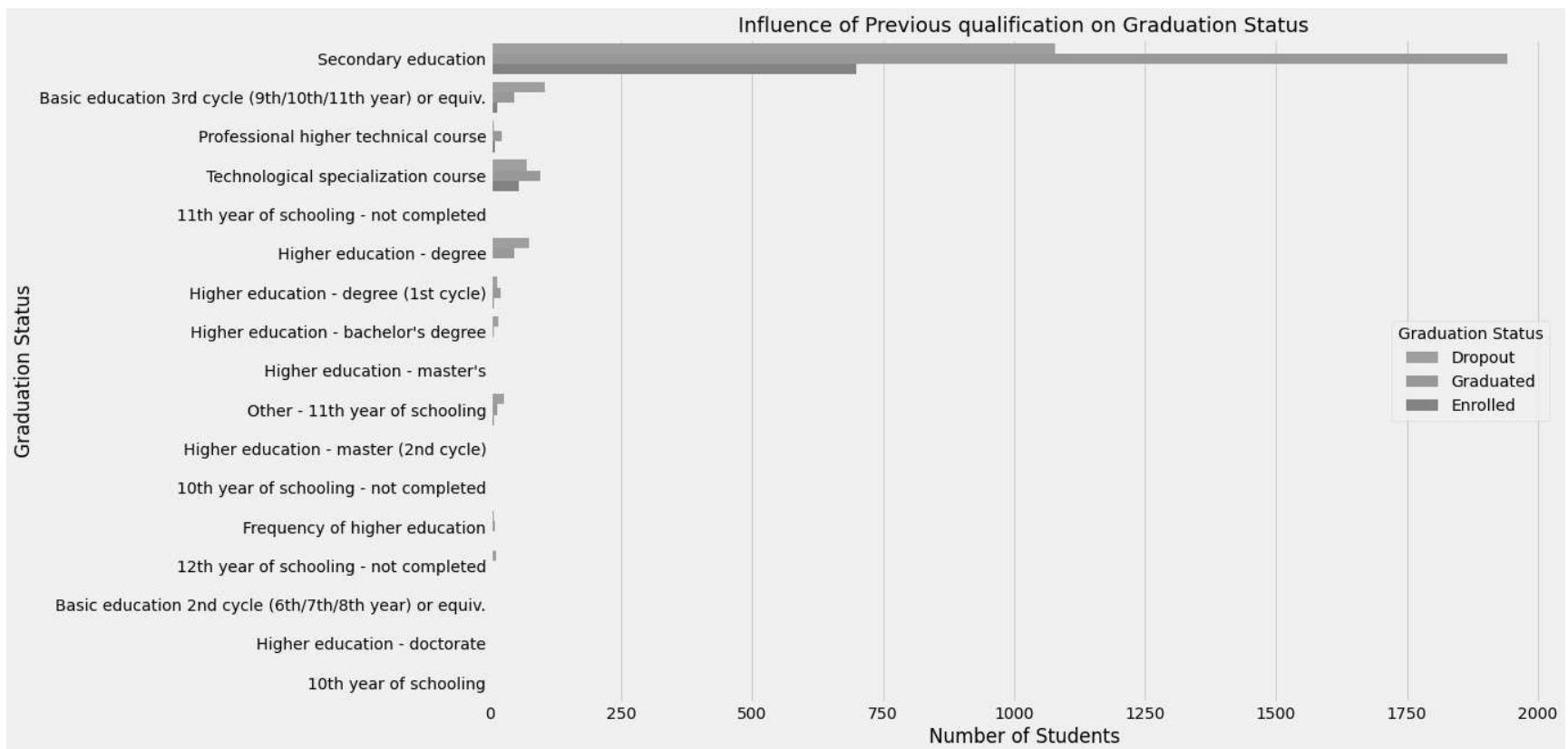






Influence of Tuition Fees on Graduation Status





Model

```
In [ ]: class TernaryClassifier:
    def __init__(self, data):
        """
        Constructor for the TernaryClassifier class.

        Parameters:
        - data (pandas DataFrame): The dataset for the classifier.
        """

        self.data = data
        self.scaler = StandardScaler()
        # Define a Library of models
        self.models = {
            'Logistic Regression': LogisticRegression(max_iter=1000),
            'K-Nearest Neighbors': KNeighborsClassifier(),
```

```

        'Support Vector Machine': SVC(),
        'Decision Trees': DecisionTreeClassifier(),
        'Random Forest': RandomForestClassifier()
    }

    def preprocess_data(self):
        # Separate features (X) and target variable (y)
        X = self.data.drop('Target', axis=1)
        y = self.data['Target']

        # Scale the input features
        scaler = StandardScaler()
        X_scaled = scaler.fit_transform(X)

        # Encode the target variable
        label_encoder = LabelEncoder()
        y_encoded = label_encoder.fit_transform(y)

        # Convert X_scaled to DataFrame and set column names
        X_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)

        return X_scaled_df, y_encoded

    def train_model(self, X, y):
        """
        Trains the ternary classifier model on the preprocessed data.

        Parameters:
        - X (pandas DataFrame): The preprocessed feature data.
        - y (pandas DataFrame): The preprocessed target variable.

        Returns:
        - model: The trained ternary classifier model.
        """
        # Train each model in the library
        for model_name, model in self.models.items():
            model.fit(X, y)

        # Return the trained models dictionary
        return self.models

    def predict(self, X_test):
        """
        Predicts the target variable values using all the trained models.
        """

```

```
Parameters:  
- X_test (pandas DataFrame): Testing feature data.  
  
Returns:  
- predictions (dict): Dictionary containing the predictions of each model.  
"""  
predictions = {}  
  
for model_name, model in self.models.items():  
    model_predictions = model.predict(X_test)  
    predictions[model_name] = model_predictions  
  
return predictions  
  
def tune_parameters(self, classifier, param_grid, X, y, scorer):  
    """  
    Tune hyperparameters of the given classifier using GridSearchCV.  
  
    Parameters:  
    - classifier: The classifier to be tuned.  
    - param_grid: The dictionary containing hyperparameter grids for the classifier.  
    - X (pandas DataFrame): The preprocessed feature data.  
    - y (pandas DataFrame): The preprocessed target variable.  
    - scorer: Scorer function to evaluate the models.  
  
    Returns:  
    - best_params: The best hyperparameters found by GridSearchCV.  
    - best_score: The best recall score achieved by the classifier.  
    """  
    grid_search = GridSearchCV(estimator=classifier, param_grid=param_grid, scoring=scorer, cv=5)  
    grid_search.fit(X, y)  
    best_params = grid_search.best_params_  
    best_score = grid_search.best_score_  
  
    return best_params, best_score  
  
def train_model_with_parameter_tuning(self, X, y):  
    """  
    Trains the ternary classifier model with parameter tuning on the preprocessed data.  
  
    Parameters:  
    - X (pandas DataFrame): The preprocessed feature data.  
    - y (pandas DataFrame): The preprocessed target variable.
```

```

>Returns:
- model: The trained ternary classifier model with tuned hyperparameters.
"""

# Define a library of models with default parameters
models = {
    'Logistic Regression': LogisticRegression(),
    'K-Nearest Neighbors': KNeighborsClassifier(),
    'Support Vector Machine': SVC(),
    'Decision Trees': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier()
}

# Create an empty dictionary to store the best models with tuned parameters
best_models = {}
best_recall_scores = {}

# Iterate through each model in the library and perform grid search for parameter tuning
for model_name, model in models.items():
    # Get the corresponding parameter grid for the model
    model_param_grid = param_grid[model_name]

    # Perform parameter tuning using the tune_parameters function
    best_params, best_score = self.tune_parameters(model, model_param_grid, X, y, scorer=make_scoring(recall_sco

    # Set the best hyperparameters to the model
    model.set_params(**best_params)

    # Train the model with the best hyperparameters
    model.fit(X, y)

    # Store the best model in the dictionary
    best_models[model_name] = model
    best_recall_scores[model_name] = best_score

# Store the trained models with tuned parameters
self.models = best_models

# Identify the most performative model
most_performative_model = max(best_recall_scores, key=best_recall_scores.get)

# Return the dictionary of best models and the most performative model
return best_models, most_performative_model

```

Evaluation

```
In [ ]: class ModelEvaluation:
    def __init__(self, X_train, y_train, X_test, y_test):
        """
        Constructor for the ModelEvaluation class.

        Parameters:
        - X_train (pandas DataFrame): Training feature data.
        - y_train (pandas Series or numpy array): Training target variable.
        - X_test (pandas DataFrame): Testing feature data.
        - y_test (pandas Series or numpy array): Testing target variable.
        """
        self.X_train = X_train
        self.y_train = y_train
        self.X_test = X_test
        self.y_test = y_test
        self.models = None
        self.evaluation_df = None
        self.feature_importance_df = None

    def evaluate_models(self, classifier):
        """
        Evaluates different classifier models and stores the evaluation results in a dataframe.

        Parameters:
        - classifier (TernaryClassifier): The TernaryClassifier instance.

        Returns:
        - evaluation_df (pandas DataFrame): The evaluation results dataframe.
        """
        models = classifier.train_model(self.X_train, self.y_train)
        self.models = models

        evaluation_results = []
        for model_name, model in models.items():
            y_pred = model.predict(self.X_test)

            accuracy = accuracy_score(self.y_test, y_pred)
            precision = precision_score(self.y_test, y_pred, average='weighted')
            recall = recall_score(self.y_test, y_pred, average='weighted')
            f1 = f1_score(self.y_test, y_pred, average='weighted')

            evaluation_results.append({
                'Model': model_name,
```

```

        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1 Score': f1
    })

evaluation_df = pd.DataFrame(evaluation_results)
self.evaluation_df = evaluation_df

return evaluation_df

def analyze_feature_importances(self):
    """
    Analyzes and stores feature importances for the RandomForestClassifier.

    Returns:
    - feature_importance_df (pandas DataFrame): The feature importances dataframe.
    """
    if 'Random Forest' in self.models:
        random_forest_model = self.models['Random Forest']
        if hasattr(random_forest_model, 'feature_importances_'):
            feature_importances = random_forest_model.feature_importances_
            feature_importance_df = pd.DataFrame({'Feature': self.X_train.columns, 'Importance': feature_importance})
            self.feature_importance_df = feature_importance_df
            return feature_importance_df.info()
        else:
            print("Random Forest model does not have feature importances.")
    else:
        print("Random Forest model not found in trained models.")

    return None

def calculate_best_scores(self, param_grids, scoring):
    """
    Calculate the best scores using cross-validation with recall scoring for all specified models.

    Parameters:
    - param_grids (dict): Dictionary containing the parameter grids for each model.
    - scoring (str or callable): The scoring metric to use for evaluation.

    Returns:
    - best_scores (dict): Dictionary containing the best recall scores for each model.
    """
    best_scores = {}

```

```

    for model_name, model in self.models.items():
        param_grid = param_grids.get(model_name)

        if param_grid is not None:
            grid_search = GridSearchCV(estimator=model, param_grid=param_grid, scoring=scoring, cv=5)
            grid_search.fit(self.X_train, self.y_train)

            best_scores[model_name] = grid_search.best_score_

    return best_scores

def plot_feature_importances(self, trained_models, feature_names):
    # Create a DataFrame to store feature importances
    feature_importances_df = pd.DataFrame(index=feature_names)

    for model_name, model in trained_models.items():
        if hasattr(model, 'feature_importances_'):
            # Get the feature importances from the model
            feature_importances = model.feature_importances_

            # Store the feature importances in the DataFrame
            feature_importances_df[model_name] = feature_importances

    # Plot the feature importances
    plt.figure(figsize=(12, 8))
    sns.barplot(data=feature_importances_df.T, x=feature_importances_df.columns, y=feature_importances_df.T.mean(ax
    plt.title('Feature Importances')
    plt.xticks(rotation=45, ha='right')
    plt.show()

def identify_best_parameters(self, param_grids, scoring):
    """
    Identify the best parameters that yield the best recall score for all specified models.

    Parameters:
    - param_grids (dict): Dictionary containing the parameter grids for each model.
    - scoring (str or callable): The scoring metric to use for evaluation.

    Returns:
    - best_parameters (dict): Dictionary containing the best parameters for each model.
    - best_recall_scores (dict): Dictionary containing the best recall scores for each model.
    """

```

```

best_parameters = {}
best_recall_scores = {}

for model_name, model in self.models.items():
    param_grid = param_grids.get(model_name)

    if param_grid is not None:
        grid_search = GridSearchCV(estimator=model, param_grid=param_grid, scoring=scoring, cv=5)
        grid_search.fit(self.X_train, self.y_train)

        best_parameters[model_name] = grid_search.best_params_
        best_recall_scores[model_name] = grid_search.best_score_

return best_parameters, best_recall_scores

```

iNterpret

```

In [ ]: # Instantiate the TernaryClassifier class with your data
ternary_classifier = TernaryClassifier(data)
#print('trace1')
# Preprocess the data using the TernaryClassifier class
X_scaled, y_encoded = ternary_classifier.preprocess_data()
#print('trace2')
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_encoded, test_size=0.2, random_state=42)
#print('trace3')
# Define parameter grids for each model
param_grid = {
    'Logistic Regression': {'C': [0.1, 1.0, 10.0]},
    'K-Nearest Neighbors': {'n_neighbors': [3, 5, 7]},
    'Support Vector Machine': {'C': [0.1, 1.0, 10.0], 'kernel': ['linear', 'rbf']},
    'Decision Trees': {'max_depth': [None, 5, 10]},
    'Random Forest': {'n_estimators': [50, 100, 200], 'max_depth': [None, 5, 10]}
}
#print('trace4')
# Train models with parameter tuning
best_models, most_performative_model = ternary_classifier.train_model_with_parameter_tuning(X_train, y_train)
#print('trace5')
# Instantiate the ModelEvaluation class
model_evaluation = ModelEvaluation(X_train, y_train, X_test, y_test)
#print ('trace6')

```

```

# Evaluate models
evaluation_results = model_evaluation.evaluate_models(ternary_classifier)
#print('trace7')
# Display evaluation results
display(evaluation_results)
#print('trace8')
# Get the best model for each classifier
logistic_regression_model = best_models['Logistic Regression']
random_forest_model = best_models['Random Forest']
svm_model = best_models['Support Vector Machine']
decision_tree_model = best_models['Decision Trees']
knn_model = best_models['K-Nearest Neighbors']
#print('trace9')
# Get the top 10 coefficients for Logistic Regression
if hasattr(logistic_regression_model, 'coef_'):
    coef_abs = abs(logistic_regression_model.coef_[0])
    top_5_coefficients = sorted(list(zip(ternary_classifier.data.drop('Target', axis=1).columns, coef_abs)), key=lambda x: -x[1])
    print("Top 5 Coefficients for Logistic Regression:")
    for feature, coef in top_5_coefficients:
        print(f"{feature}: {coef}")
    print("\n")
else:
    print("Logistic Regression model does not have coefficients.")
#print('trace10')
# Get the top 5 feature importances for Random Forest, Support Vector Machines, Decision Trees, and K-Nearest Neighbors
models_to_analyze = [random_forest_model, svm_model, decision_tree_model, knn_model]
model_names = ['Random Forest', 'Support Vector Machine', 'Decision Trees', 'K-Nearest Neighbors']
#print('trace11')
for model, model_name in zip(models_to_analyze, model_names):
    if hasattr(model, 'feature_importances_'):
        feature_importances = model.feature_importances_
        top_5_features = sorted(list(zip(ternary_classifier.data.drop('Target', axis=1).columns, feature_importances)), key=lambda x: -x[1])
        print(f"Top 5 Feature Importances for {model_name}:")
        for feature, importance in top_5_features:
            print(f"{feature}: {importance}")
        print("\n")
    else:
        print(f"{model_name} model does not have feature importances.")

```

In []:

```

# Instantiate the TernaryClassifier class with your data
ternary_classifier = TernaryClassifier(cleaned_data)

# Preprocess the data using the TernaryClassifier class
X_scaled, y_encoded = ternary_classifier.preprocess_data()

```

```

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_encoded, test_size=0.2, random_state=42)

# Assuming you have already trained the KNN model in the TernaryClassifier class
knn_model = ternary_classifier.models['K-Nearest Neighbors']

# Perform feature scaling on the training data
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)

# Create a SelectKBest selector and fit it with the training data
k_best_selector = SelectKBest(score_func=chi2, k=5)
X_train_k_best = k_best_selector.fit_transform(X_train_scaled, y_train)

# Get the support (selected) features from SelectKBest
selected_features_indices = k_best_selector.get_support(indices=True)
selected_features = X_train.columns[selected_features_indices]

# Print the top 3 selected features
print("Top 5 selected features by SelectKBest with KNN:")
for feature in selected_features:
    print(feature)

```

In []:

```

# Instantiate the TernaryClassifier class with your data
ternary_classifier = TernaryClassifier(cleaned_data)

# Preprocess the data using the TernaryClassifier class
X_scaled, y_encoded = ternary_classifier.preprocess_data()

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_encoded, test_size=0.2, random_state=42)

# Assuming you have already trained the SVM model in the TernaryClassifier class
svm_model = ternary_classifier.models['Support Vector Machine']

# Perform feature scaling on the training data
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)

# Create an RFE selector and fit it with the trained SVM model
rfe_selector = RFE(estimator=SVC(kernel='linear'), n_features_to_select=5)
rfe_selector.fit(X_train_scaled, y_train)

# Get the support (selected) features from RFE
selected_features_indices = rfe_selector.get_support(indices=True)

```

```
selected_features = X_train.columns[selected_features_indices]

# Print the top 5 selected features
print("Top 5 selected features by RFE with SVM:")
for feature in selected_features:
    print(feature)
```

Insights

The feature importances from the Random Forest model and the coefficients from the Logistic Regression model both highlight the significant role of curricular units in the 2nd semester in predicting academic failure. Students who have more approved units in the 2nd semester demonstrate a higher likelihood of academic success. To effectively reduce the rate of academic failure, the student success team from the Instituto Politécnico de Portalegre should focus on supporting and guiding students to pass more units in the 2nd semester.

The Logistic Regression model emphasizes the importance of being up-to-date with tuition fees as a crucial factor in predicting academic success. Although it may not be among the top 5 feature importances for Random Forest, timely tuition fee payments are still relevant. To mitigate academic failure, the student success team should implement strategies to ensure that students stay on top of their tuition fee payments, such as providing clear payment reminders and offering financial assistance when needed.

Both models underscore the significance of performance-related features, including grades and evaluations in the 1st and 2nd semesters. Students who excel in these evaluations demonstrate a higher likelihood of academic success. To reduce academic failure, the student success team should closely monitor early academic performance and provide additional support or interventions to students who may be struggling in their evaluations or obtaining low grades.

In light of these insights, I recommend the student success team from the Instituto Politécnico de Portalegre to adopt a multifaceted approach. By combining academic support, monitoring tuition fee payments, and providing early interventions based on performance, the team can effectively promote student success and improve overall student outcomes. Leveraging data-driven insights, the student success team can tailor their efforts and allocate resources strategically to better support students and foster a culture of academic achievement.

Insight 1: Curricular Units in 2nd Semester Matter

- Students with more approved units in the 2nd semester have a higher chance of academic success.
- The number of enrolled units in the 2nd semester also positively impacts academic outcomes.
- Encouraging students to pass more units in the 2nd semester can lead to a reduced rate of academic failure.

Focusing on students' progress and success in their 2nd-semester curricular units can significantly contribute to reducing the rate of academic failure.

Insight 2: Monitoring Tuition Fee Payments is Pivotal

Being up-to-date with tuition fees is crucial for predicting academic success. Timely tuition fee payments correlate with better student outcomes. Implementing strategies to ensure prompt tuition fee payments can positively impact academic performance and reduce failure rates.

By closely monitoring and supporting students to stay on track with tuition fee payments, institutions can foster an environment that reduces the rate of academic failure.

Insight 3: Evaluations and Grades are Crucial

Academic performance in the 1st and 2nd semester evaluations significantly influences students' likelihood of success. Students who achieve higher grades tend to have better academic outcomes overall. Early identification and intervention for students struggling with evaluations or grades can lead to improved academic performance and decreased failure rates.

Prioritizing academic support and intervention based on early evaluations and grades can effectively contribute to minimizing the rate of academic failure.

Future Work

I am interested in exploring how to apply XGBoost, a powerful gradient boosting algorithm to a ternary (3 class) classification model. XGBoost handles large datasets and captures complex interactions between features. This could enhance predictive modeling for academic success, improving accuracy and interpretability. Employing a robust hyperparameter tuning process, like grid search or Bayesian optimization, can fine-tune model parameters for better results.

Extending the analysis to include US universities may also provide valuable insights. Identifying common factors influencing student success across diverse contexts is possible. Understanding how these factors vary between institutions. I would like to strengthen both the generalizability and the ability to tailor interventions that are research driven to address academic challenges.

By leveraging advanced techniques and cross-institutional analyses, educational research advances that leverage evidence-based practices, we can systematically improve student outcomes.

I want to craft a set of informed decisions and targeted interventions that reduce the rate of academic failure effectively.