College of Computer and Information Sciences
Computer Science Department

CSC 462
Machine learning

HW2: Classification with Logistic Regression

Prepared by:

Name: Sarah Alajlan

Student ID: 442202314

Section: 72917

Dr. Sarab AlMuhaideb

Data Preprocessing:

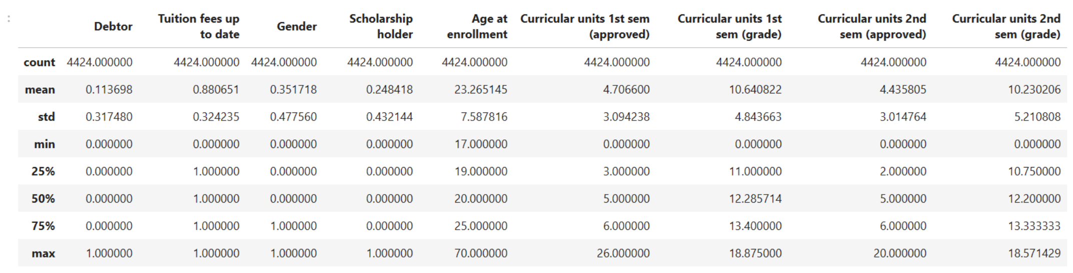
Loading the Dataset: The code loads the dataset from a CSV file named 'HW2-dataset.csv' using the Pandas library. The dataset is stored in a Pandas DataFrame called `data`.

Data Copy: A copy of the original dataset is created and stored in the variable `data_copy`. This copy is made before any modifications to the data, ensuring that the original data is preserved for reference.

Encoding the 'Target' Column: The 'Target' column in the dataset contains categorical values ('Dropout', 'Graduate', and 'Enrolled'). This column is encoded into binary values as follows:
- 'Dropout' is mapped to 1, indicating a positive class.
- 'Graduate' and 'Enrolled' are both mapped to 0, indicating a negative class.

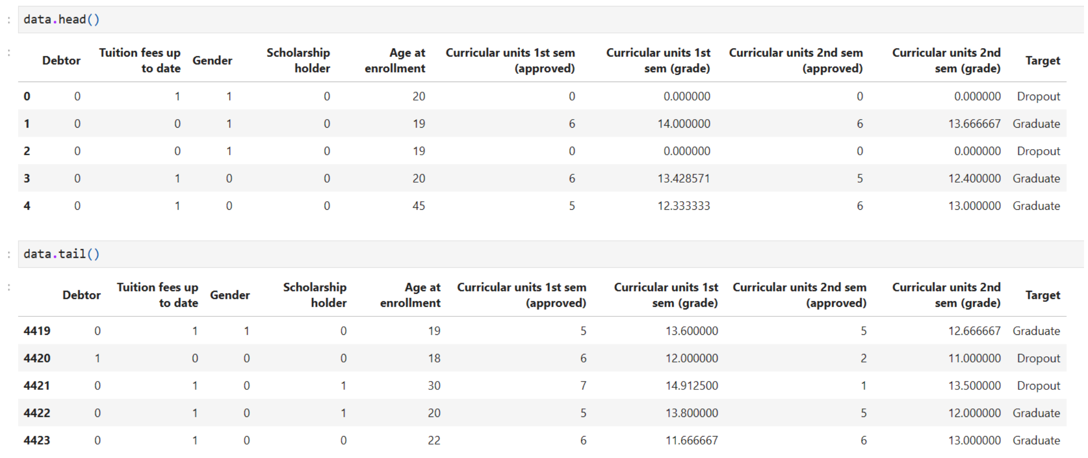
Separating Features and Target: The dataset is separated into features (X) and the target variable (y) for both binary and multinomial classification tasks. The features are extracted from the first 9 columns of the dataset, while the target variable is extracted from the last column.

In figure 1,2 and 3 shows the output of the functions describe ,head ,tail and info.



	Debtor	Tuition fees up to date	Gender	Scholarship holder	Age at enrollment	Curricular units 1st sem (approved)	Curricular units 1st sem (grade)	Curricular units 2nd sem (approved)	Curricular units 2nd sem (grade)
count	4424.000000	4424.000000	4424.000000	4424.000000	4424.000000	4424.000000	4424.000000	4424.000000	4424.000000
mean	0.113698	0.880651	0.351718	0.248418	23.265145	4.706600	10.640822	4.435805	10.230206
std	0.317480	0.324235	0.477560	0.432144	7.587816	3.094238	4.843663	3.014764	5.210808
min	0.000000	0.000000	0.000000	0.000000	17.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	1.000000	0.000000	0.000000	19.000000	3.000000	11.000000	2.000000	10.750000
50%	0.000000	1.000000	0.000000	0.000000	20.000000	5.000000	12.285714	5.000000	12.200000
75%	0.000000	1.000000	1.000000	0.000000	25.000000	6.000000	13.400000	6.000000	13.333333
max	1.000000	1.000000	1.000000	1.000000	70.000000	26.000000	18.875000	20.000000	18.571429

Figure 1: output of describe function



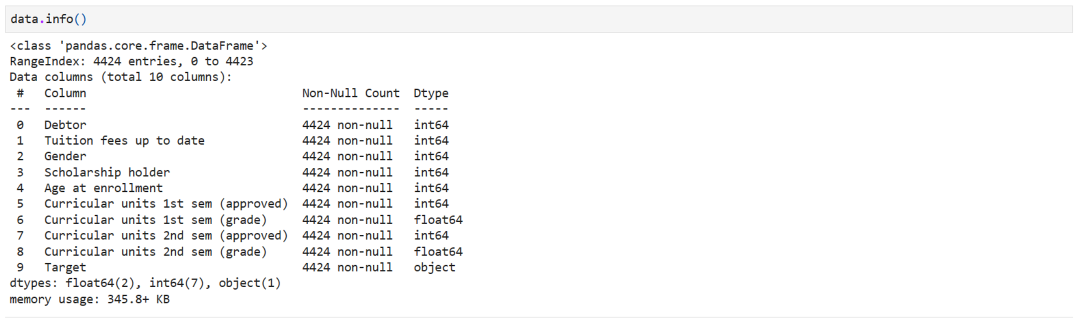
```
data.head()
```

	Debtor	Tuition fees up to date	Gender	Scholarship holder	Age at enrollment	Curricular units 1st sem (approved)	Curricular units 1st sem (grade)	Curricular units 2nd sem (approved)	Curricular units 2nd sem (grade)	Target
0	0	1	1	0	20	0	0.000000	0	0.000000	Dropout
1	0	0	1	0	19	6	14.000000	6	13.666667	Graduate
2	0	0	1	0	19	0	0.000000	0	0.000000	Dropout
3	0	1	0	0	20	6	13.428571	5	12.400000	Graduate
4	0	1	0	0	45	5	12.333333	6	13.000000	Graduate

```
data.tail()
```

	Debtor	Tuition fees up to date	Gender	Scholarship holder	Age at enrollment	Curricular units 1st sem (approved)	Curricular units 1st sem (grade)	Curricular units 2nd sem (approved)	Curricular units 2nd sem (grade)	Target
4419	0	1	1	0	19	5	13.600000	5	12.666667	Graduate
4420	1	0	0	0	18	6	12.000000	2	11.000000	Dropout
4421	0	1	0	1	30	7	14.912500	1	13.500000	Dropout
4422	0	1	0	1	20	5	13.800000	5	12.000000	Graduate
4423	0	1	0	0	22	6	11.666667	6	13.000000	Graduate

Figure 2: output of describe head and tail



```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4424 entries, 0 to 4423
Data columns (total 10 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Debtor                          4424 non-null   int64
 1   Tuition fees up to date         4424 non-null   int64
 2   Gender                          4424 non-null   int64
 3   Scholarship holder              4424 non-null   int64
 4   Age at enrollment               4424 non-null   int64
 5   Curricular units 1st sem (approved)  4424 non-null   int64
 6   Curricular units 1st sem (grade)     4424 non-null   float64
 7   Curricular units 2nd sem (approved)  4424 non-null   int64
 8   Curricular units 2nd sem (grade)     4424 non-null   float64
 9   Target                          4424 non-null   object
dtypes: float64(2), int64(7), object(1)
memory usage: 345.8+ KB
```

Figure 1: output of info function

Hyperparameters:

Learning Rate (Learning_rate):

- The code defines a list of learning rates: [0.01, 0.1, 1].
- Learning rate is a hyperparameter used in the optimization algorithm of logistic regression. It controls the step size during the model's weight updates.
- The code iterates through each learning rate to evaluate the model's performance at different learning rates for both binary and multinomial classification.

Regularization Strength (C):

- Regularization strength, often denoted as C, is another hyperparameter specific to logistic regression.
- It is set when creating the logistic regression models (LogisticRegression instances) for binary and multinomial classification.
- The code uses different values of C for model training (C=i), where i represents the learning rate being tested.
- A smaller C value increases the regularization strength, while a larger C value reduces it.

Parameters:

Number of Folds (n_splits):

- In both binary and multinomial classification, the code uses StratifiedKFold to perform cross-validation with a specified number of folds.
- The number of folds is set to 5 using n_splits=5.

Maximum Number of Iterations (max_iter):

- This parameter sets the maximum number of iterations for the logistic regression models to converge.
- It's set to 1000 for binary classification and 10000 for multinomial classification.

Multi-class Strategy (multi_class):

- For multinomial classification, the code specifies the multi-class strategy as 'multinomial' when creating the logistic regression model.

These hyperparameters are crucial in determining the model's performance and generalization ability. The code systematically explores different learning rates and regularization strengths to assess how they impact the classification performance. It's important to note that, in scikit-learn, the learning rate is not directly adjustable. Instead, regularization, specifically the 'C' parameter, indirectly influences the learning process. The regularization strength controls the model's tendency to fit the training data closely, and by finding the right balance between regularization and learning rate, the code effectively fine-tunes the logistic regression models to achieve the desired level of accuracy and generalization on the given dataset.

Steps that contribute to obtaining the final results other than the data preprocessing steps

1. Binary Classification:
- The code iterates through a list of learning rates (0.01, 0.1, 1).
- For each learning rate, it performs binary classification:
- Utilizes 5-fold cross-validation using `StratifiedKFold` to ensure balanced class distribution in each fold.
- Trains a binary logistic regression model with the specified learning rate (controlled indirectly through the regularization strength C).
- Makes predictions on the test data and calculates performance metrics for each fold.
- Records performance metrics (accuracy, precision, recall) in the `binary_results` dictionary.
- Displays results for each fold.

2. Multinomial Classification:
- Similar to binary classification, multinomial classification is performed for each learning rate in the list.
- For each learning rate:
- 5-fold cross-validation is conducted using `StratifiedKFold`.
- A multinomial logistic regression model is trained with the specified learning rate (controlled indirectly through C).
- Predictions are made, and performance metrics (accuracy, precision, recall, F1-Score) are calculated for each fold.
- Performance metrics are recorded in the `multinomial_results` dictionary.
- Results for each fold are displayed.

3. Print Overall Results:
- After completing binary and multinomial classification for each learning rate, the code prints the overall results.
- It calculates and displays the average accuracy, precision, recall, and F1-Score for both binary and multinomial classification.

4. Confusion Matrix for Multinomial Classification:

- The code accumulates true and predicted labels in `cf1` and `cf2` lists, respectively, for multinomial classification.
- After all folds, it calculates the confusion matrix using `confusion_matrix`.
- It displays the confusion matrix using `ConfusionMatrixDisplay` and adjusts the colormap and title for better visualization.

These steps focus on the evaluation of binary and multinomial classification models, including model training, performance metric calculation, and the presentation of results. The code systematically explores different learning rates and regularization strengths to assess their impact on classification performance and obtain comprehensive results for analysis.

Evaluation results

Binary classification evaluation results for the 5 folds for learning rate 0.01:

Table 1

Fold #	**Accuracy**	**Precision**	**Recall**
Fold 1	0.847	0.844	0.645
Fold 2	0.846	0.821	0.665
Fold 3	0.818	0.770	0.616
Fold 4	0.849	0.838	0.658
Fold 5	0.846	0.833	0.651

Overall Binary Classification Results:
Average Accuracy: 0.8415471534115602
Average Precision: 0.8217198587139011
Average Recall: 0.6474326661724734

Multinomial classification evaluation results for the 5 folds for learning rate 0.01:

Table 2

Fold #	**Accuracy**	**Precision**	**Recall**
Fold 1	0.726	0.686	0.726
Fold 2	0.722	0.691	0.722
Fold 3	0.718	0.672	0.718
Fold 4	0.731	0.712	0.731

Fold 5	0.727	0.685	0.727

Overall Multinomial Classification Results:
Average Accuracy: 0.7251361300713246
Average Precision: 0.689585261419235
Average Recall: 0.7251361300713246
Average F1-Score: 0.672445783398329

Binary classification evaluation results for the 5 folds for learning rate 0.1:

Table 3

Fold #	**Accuracy**	**Precision**	**Recall**
Fold 1	0.856	0.840	0.684
Fold 2	0.848	0.817	0.679
Fold 3	0.831	0.777	0.665
Fold 4	0.856	0.839	0.683
Fold 5	0.852	0.815	0.700

Overall Binary Classification Results:
Average Accuracy: 0.8492323030907277
Average Precision: 0.8182984479290445
Average Recall: 0.6826167531504818

Multinomial classification evaluation results for the 5 folds for learning rate 0.1:

Table 4

Fold #	**Accuracy**	**Precision**	**Recall**
Fold 1	0.729	0.692	0.729
Fold 2	0.728	0.702	0.728
Fold 3	0.722	0.679	0.722
Fold 4	0.737	0.716	0.737
Fold 5	0.729	0.694	0.729

Overall Multinomial Classification Results:
Average Accuracy: 0.7296564153692768
Average Precision: 0.6972690942860733
Average Recall: 0.7296564153692768
Average F1-Score: 0.6879327768215184

Binary classification evaluation results for the 5 folds for learning rate 1:

Table 5

Fold #	Accuracy	Precision	Recall
Fold 1	0.857	0.838	0.691
Fold 2	0.851	0.817	0.693
Fold 3	0.830	0.768	0.676
Fold 4	0.857	0.840	0.686
Fold 5	0.856	0.828	0.697

Overall Binary Classification Results:
Average Accuracy: 0.8508149909246618
Average Precision: 0.818538876491408
Average Recall: 0.6889498393872003

Multinomial classification evaluation results for the 5 folds for learning rate 1:

Table 6

Fold #	Accuracy	Precision	Recall
Fold 1	0.731	0.695	0.731
Fold 2	0.729	0.704	0.729
Fold 3	0.717	0.672	0.717
Fold 4	0.740	0.723	0.740
Fold 5	0.729	0.694	0.729

Overall Multinomial Classification Results:

Average Accuracy: 0.7296564153692768
Average Precision: 0.6980958466806028
Average Recall: 0.7296564153692768
Average F1-Score: 0.6896477401950964

Analysis:

Binary Classification:

Learning Rate 0.01:

Table 1 shows the binary classification results for a learning rate of 0.01.
Across five folds, the model achieves average accuracy of approximately 84.16%, with average precision and recall of around 82.17% and 64.74%, respectively.
Insights: The model demonstrates good accuracy but lower recall, indicating that it correctly classifies non-dropout students but may miss some actual dropouts.

Learning Rate 0.1:

Table 3 displays results for binary classification with a learning rate of 0.1.
The model performs even better with an average accuracy of around 84.92% and improved average precision and recall of approximately 81.83% and 68.26%, respectively.
Insights: The model's overall performance improves with a higher learning rate, maintaining high accuracy and achieving better recall.

Learning Rate 1:

Table 5 shows binary classification results for a learning rate of 1.
The model continues to perform well, with an average accuracy of about 85.08%. It maintains an average precision of around 81.85% and average recall of approximately 68.89%.
Insights: The model performs consistently well across learning rates, with a slight increase in accuracy and precision for the highest learning rate.

Multinomial Classification:

Learning Rate 0.01:

Table 2 presents multinomial classification results for a learning rate of 0.01.
The model achieves an average accuracy of approximately 72.51% and average precision approximately 68.95% and recall of around 72.51%.
Additionally, the average F1-Score is approximately 67.24%.
Insights: The model performs reasonably well in distinguishing between multiple class labels, with balanced precision and recall.

Learning Rate 0.1:

Table 4 displays results for multinomial classification with a learning rate of 0.1.
The model maintains a similar level of performance, with an average accuracy of about 72.97% and precision approximately 69.72% and recall of around 72.96%.

The average F1-Score is around 68.79%.
Insights: The model's performance remains consistent across learning rates, and it effectively handles multinomial classification.

Learning Rate 1:

Table 6 presents multinomial classification results for a learning rate of 1.
The model maintains an average accuracy of approximately 72.97% and precision approximately 69.81% and recall of around 72.96%.

The average F1-Score is approximately 68.96%.
Insights: Similar to previous learning rates, the model consistently performs well in distinguishing between multiple class labels.

Overall Observations:

Binary Classification:

Binary classification shows improved performance with higher learning rates.
The model maintains high accuracy and precision, indicating its ability to correctly classify non-dropout students.
While recall improves with higher learning rates, it's essential to consider the trade-off between precision and recall.

Multinomial Classification:

Multinomial classification results are consistent across learning rates, demonstrating the model's capability to handle multiple class labels effectively.
The model maintains balanced precision and recall, with the F1-Score providing a good balance between the two.
In summary, the model performs well in both binary and multinomial classification tasks, with some improvement in performance observed with higher learning rates. The choice of learning rate should consider the specific trade-offs between precision and recall that align with the problem's objectives and constraints.

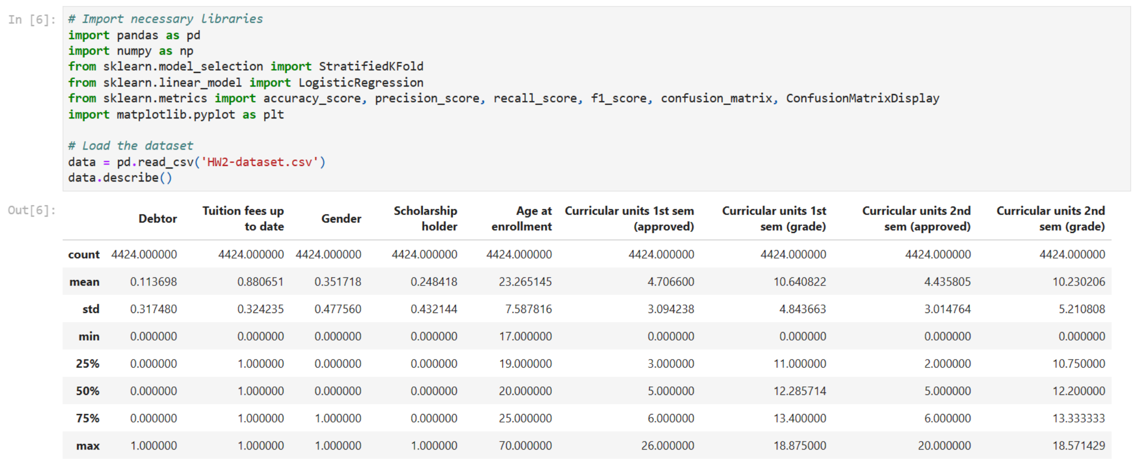
Challenges:

Class Imbalances: Dealing with class imbalances, especially in the context of binary classification, can be challenging. Ensuring the model performs well for both classes can require additional techniques like Encoding the data or using different evaluation metrics.

Implementing the learning rate posed a challenge because there is no direct way to adjust the learning rate using the scikit-learn library. As a solution, regularization was used to indirectly adjust it.

Screenshots of all the code and evaluation results

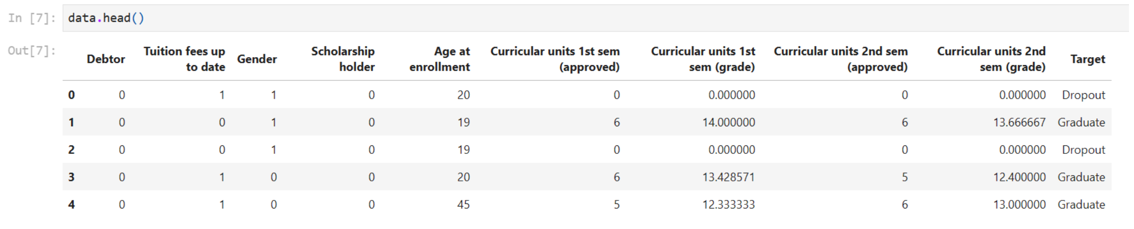
Screenshot of the Code:



In [6]:
```python
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Load the dataset
data = pd.read_csv('HW2-dataset.csv')
data.describe()
```

Out[6]:

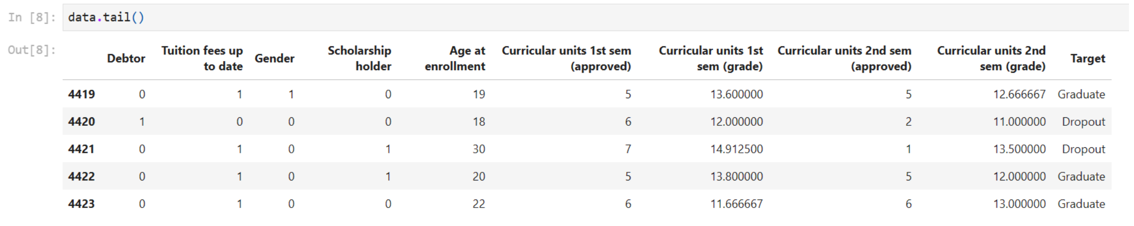
	Debtor	Tuition fees up to date	Gender	Scholarship holder	Age at enrollment	Curricular units 1st sem (approved)	Curricular units 1st sem (grade)	Curricular units 2nd sem (approved)	Curricular units 2nd sem (grade)
count	4424.000000	4424.000000	4424.000000	4424.000000	4424.000000	4424.000000	4424.000000	4424.000000	4424.000000
mean	0.113698	0.880651	0.351718	0.248418	23.265145	4.706600	10.640822	4.435805	10.230206
std	0.317480	0.324235	0.477560	0.432144	7.587816	3.094238	4.843663	3.014764	5.210808
min	0.000000	0.000000	0.000000	0.000000	17.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	1.000000	0.000000	0.000000	19.000000	3.000000	11.000000	2.000000	10.750000
50%	0.000000	1.000000	0.000000	0.000000	20.000000	5.000000	12.285714	5.000000	12.200000
75%	0.000000	1.000000	1.000000	0.000000	25.000000	6.000000	13.400000	6.000000	13.333333
max	1.000000	1.000000	1.000000	1.000000	70.000000	26.000000	18.875000	20.000000	18.571429



In [7]:
```python
data.head()
```

Out[7]:

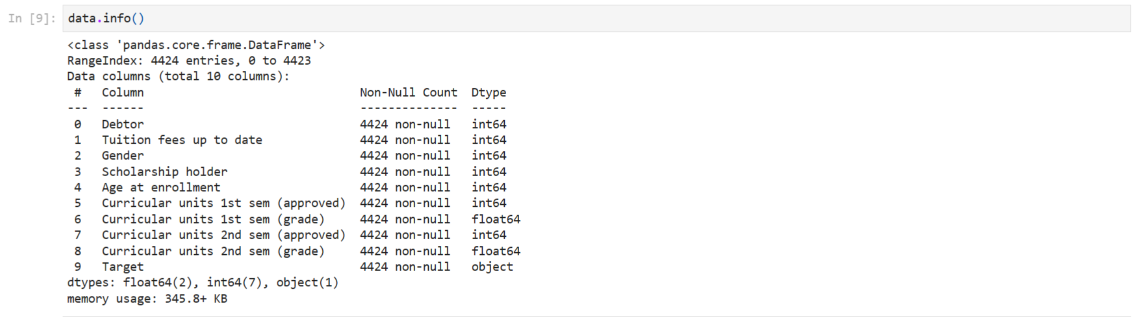
	Debtor	Tuition fees up to date	Gender	Scholarship holder	Age at enrollment	Curricular units 1st sem (approved)	Curricular units 1st sem (grade)	Curricular units 2nd sem (approved)	Curricular units 2nd sem (grade)	Target
0	0	1	1	0	20	0	0.000000	0	0.000000	Dropout
1	0	0	1	0	19	6	14.000000	6	13.666667	Graduate
2	0	0	1	0	19	0	0.000000	0	0.000000	Dropout
3	0	1	0	0	20	6	13.428571	5	12.400000	Graduate
4	0	1	0	0	45	5	12.333333	6	13.000000	Graduate



In [8]:
```python
data.tail()
```

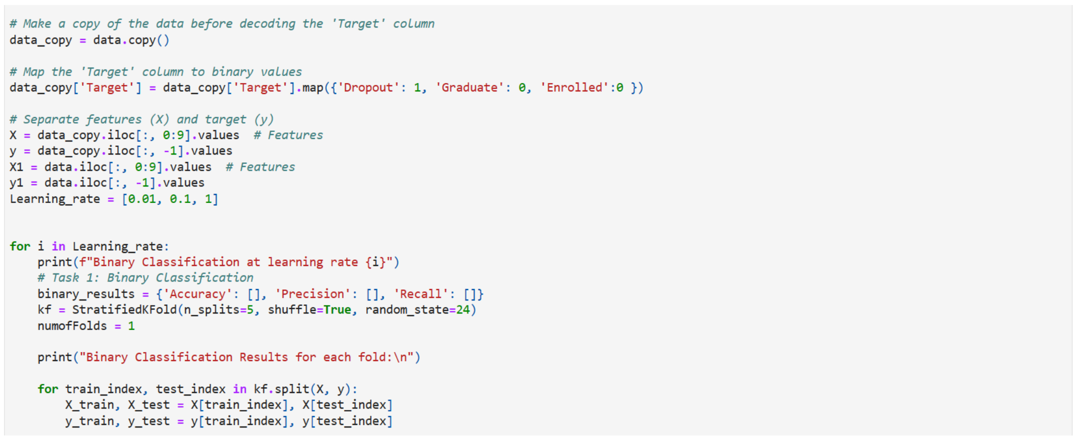
Out[8]:

	Debtor	Tuition fees up to date	Gender	Scholarship holder	Age at enrollment	Curricular units 1st sem (approved)	Curricular units 1st sem (grade)	Curricular units 2nd sem (approved)	Curricular units 2nd sem (grade)	Target
4419	0	1	1	0	19	5	13.600000	5	12.666667	Graduate
4420	1	0	0	0	18	6	12.000000	2	11.000000	Dropout
4421	0	1	0	1	30	7	14.912500	1	13.500000	Dropout
4422	0	1	0	1	20	5	13.800000	5	12.000000	Graduate
4423	0	1	0	0	22	6	11.666667	6	13.000000	Graduate

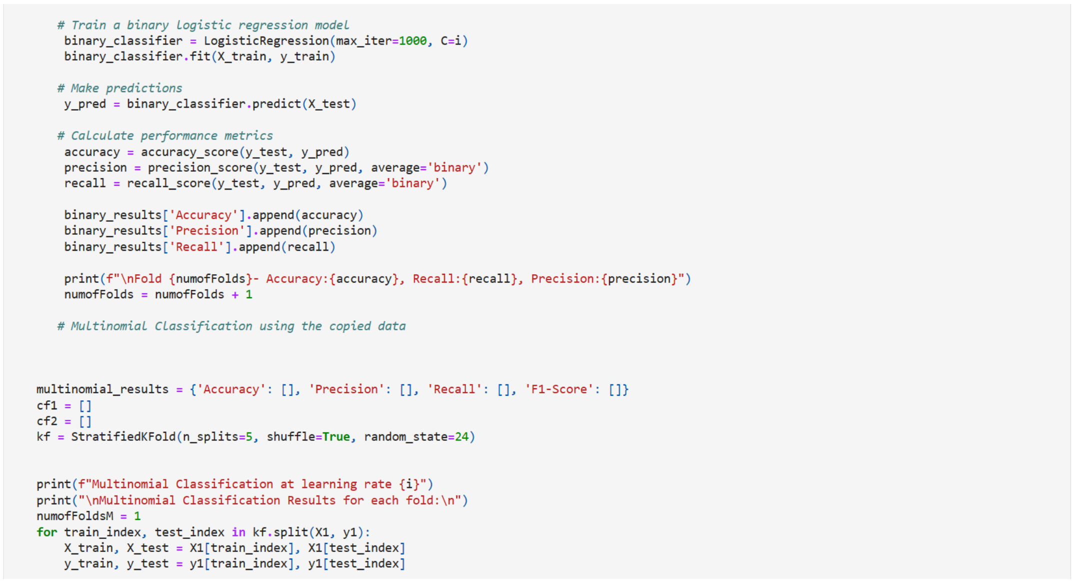


In [9]:
```python
data.info()
```

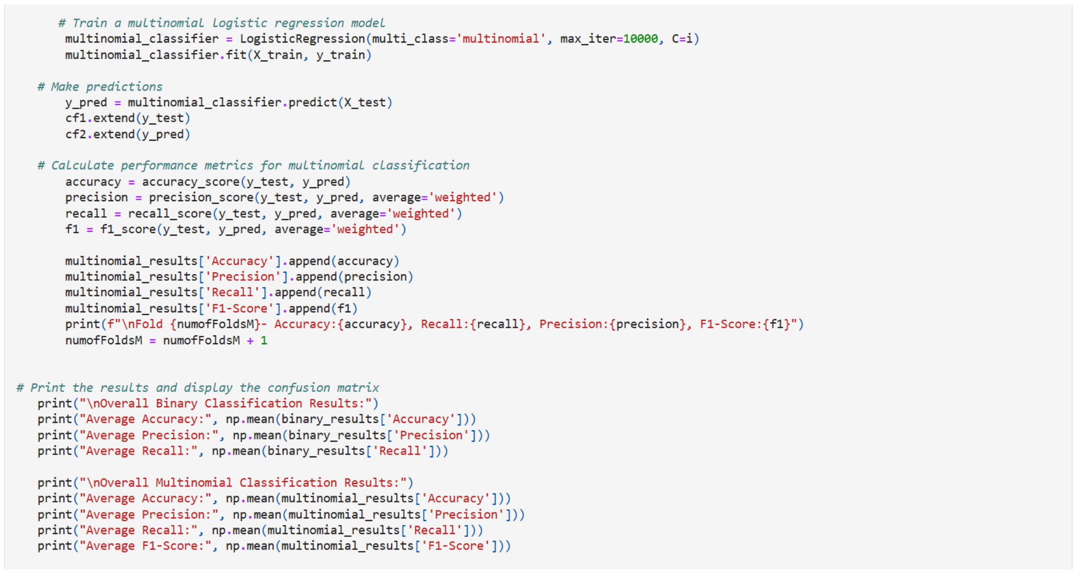
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4424 entries, 0 to 4423
Data columns (total 10 columns):
 #   Column                               Non-Null Count  Dtype  
---  ------                               --------------  -----  
 0   Debtor                               4424 non-null   int64  
 1   Tuition fees up to date              4424 non-null   int64  
 2   Gender                               4424 non-null   int64  
 3   Scholarship holder                   4424 non-null   int64  
 4   Age at enrollment                    4424 non-null   int64  
 5   Curricular units 1st sem (approved)  4424 non-null   int64  
 6   Curricular units 1st sem (grade)     4424 non-null   float64
 7   Curricular units 2nd sem (approved)  4424 non-null   int64  
 8   Curricular units 2nd sem (grade)     4424 non-null   float64
 9   Target                               4424 non-null   object 
dtypes: float64(2), int64(7), object(1)
memory usage: 345.8+ KB
```



```python
# Make a copy of the data before decoding the 'Target' column
data_copy = data.copy()

# Map the 'Target' column to binary values
data_copy['Target'] = data_copy['Target'].map({'Dropout': 1, 'Graduate': 0, 'Enrolled':0 })

# Separate features (X) and target (y)
X = data_copy.iloc[:, 0:9].values  # Features
y = data_copy.iloc[:, -1].values
X1 = data.iloc[:, 0:9].values  # Features
y1 = data.iloc[:, -1].values
Learning_rate = [0.01, 0.1, 1]


for i in Learning_rate:
    print(f"Binary Classification at learning rate {i}")
    # Task 1: Binary Classification
    binary_results = {'Accuracy': [], 'Precision': [], 'Recall': []}
    kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=24)
    numofFolds = 1

    print("Binary Classification Results for each fold:\n")

    for train_index, test_index in kf.split(X, y):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]
```



```python
        # Train a binary logistic regression model
         binary_classifier = LogisticRegression(max_iter=1000, C=i)
         binary_classifier.fit(X_train, y_train)

        # Make predictions
         y_pred = binary_classifier.predict(X_test)

        # Calculate performance metrics
         accuracy = accuracy_score(y_test, y_pred)
         precision = precision_score(y_test, y_pred, average='binary')
         recall = recall_score(y_test, y_pred, average='binary')

         binary_results['Accuracy'].append(accuracy)
         binary_results['Precision'].append(precision)
         binary_results['Recall'].append(recall)

         print(f"\nFold {numofFolds}- Accuracy:{accuracy}, Recall:{recall}, Precision:{precision}")
         numofFolds = numofFolds + 1

        # Multinomial Classification using the copied data



     multinomial_results = {'Accuracy': [], 'Precision': [], 'Recall': [], 'F1-Score': []}
     cf1 = []
     cf2 = []
     kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=24)


     print(f"Multinomial Classification at learning rate {i}")
     print("\nMultinomial Classification Results for each fold:\n")
     numofFoldsM = 1
     for train_index, test_index in kf.split(X1, y1):
         X_train, X_test = X1[train_index], X1[test_index]
         y_train, y_test = y1[train_index], y1[test_index]
```

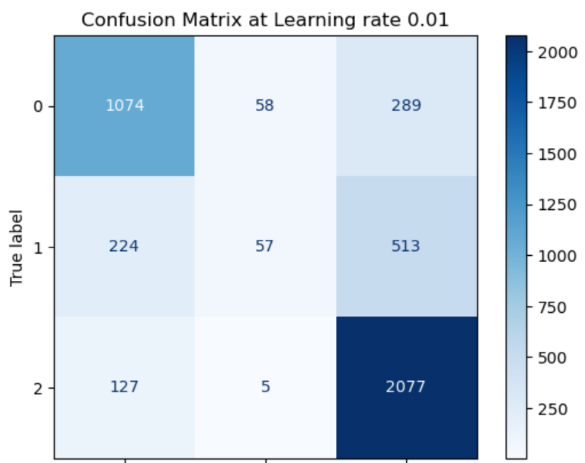


```python
       # Train a multinomial logistic regression model
        multinomial_classifier = LogisticRegression(multi_class='multinomial', max_iter=10000, C=i)
        multinomial_classifier.fit(X_train, y_train)

    # Make predictions
        y_pred = multinomial_classifier.predict(X_test)
        cf1.extend(y_test)
        cf2.extend(y_pred)

    # Calculate performance metrics for multinomial classification
        accuracy = accuracy_score(y_test, y_pred)
        precision = precision_score(y_test, y_pred, average='weighted')
        recall = recall_score(y_test, y_pred, average='weighted')
        f1 = f1_score(y_test, y_pred, average='weighted')

        multinomial_results['Accuracy'].append(accuracy)
        multinomial_results['Precision'].append(precision)
        multinomial_results['Recall'].append(recall)
        multinomial_results['F1-Score'].append(f1)
        print(f"\nFold {numofFoldsM}- Accuracy:{accuracy}, Recall:{recall}, Precision:{precision}, F1-Score:{f1}")
        numofFoldsM = numofFoldsM + 1


 # Print the results and display the confusion matrix
    print("\nOverall Binary Classification Results:")
    print("Average Accuracy:", np.mean(binary_results['Accuracy']))
    print("Average Precision:", np.mean(binary_results['Precision']))
    print("Average Recall:", np.mean(binary_results['Recall']))

    print("\nOverall Multinomial Classification Results:")
    print("Average Accuracy:", np.mean(multinomial_results['Accuracy']))
    print("Average Precision:", np.mean(multinomial_results['Precision']))
    print("Average Recall:", np.mean(multinomial_results['Recall']))
    print("Average F1-Score:", np.mean(multinomial_results['F1-Score']))
```

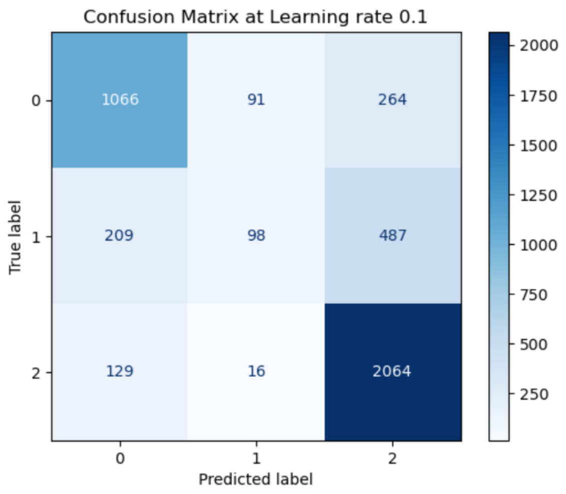
```
# Confusion matrix for Multinomial Classification

    conf = confusion_matrix(cf1, cf2)

    disp = ConfusionMatrixDisplay(conf)

    disp.plot(cmap=plt.cm.Blues)

    plt.title(f"Confusion Matrix at Learning rate {i}")

    plt.show()
```

Evaluation metrics:

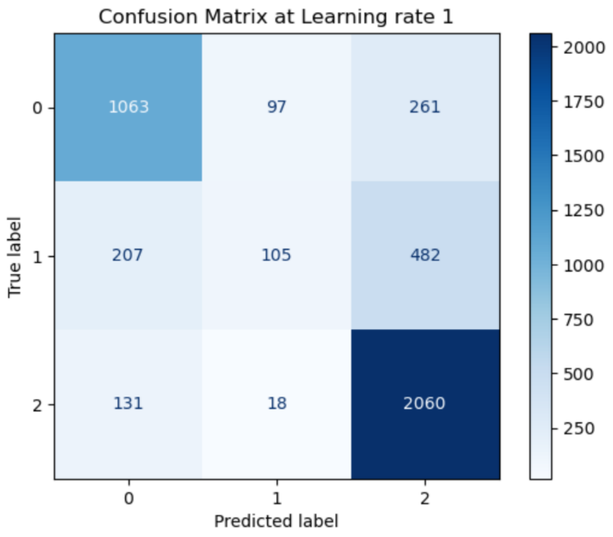
```
Binary Classification at learning rate 0.01
Binary Classification Results for each fold:


Fold 1- Accuracy:0.847457627118644, Recall:0.6456140350877193, Precision:0.8440366972477065

Fold 2- Accuracy:0.8463276836158192, Recall:0.6654929577464789, Precision:0.8217391304347826

Fold 3- Accuracy:0.8180790960451978, Recall:0.6161971830985915, Precision:0.7709251101321586

Fold 4- Accuracy:0.8497175141242937, Recall:0.6584507042253521, Precision:0.8385650224215246

Fold 5- Accuracy:0.8461538461538461, Recall:0.6514084507042254, Precision:0.8333333333333334
Multinomial Classification at learning rate 0.01

Multinomial Classification Results for each fold:


Fold 1- Accuracy:0.7265536723163842, Recall:0.7265536723163842, Precision:0.6860494983592844, F1-Score:0.6797040962366429

Fold 2- Accuracy:0.7220338983050848, Recall:0.7220338983050848, Precision:0.6918848250700694, F1-Score:0.666966039009466

Fold 3- Accuracy:0.7186440677966102, Recall:0.7186440677966102, Precision:0.6720136048380603, F1-Score:0.6637910287136521

Fold 4- Accuracy:0.7310734463276836, Recall:0.7310734463276836, Precision:0.7127898775086244, F1-Score:0.6789793226722844

Fold 5- Accuracy:0.7273755656108597, Recall:0.7273755656108597, Precision:0.6851885013201369, F1-Score:0.6727884303595992

Overall Binary Classification Results:
Average Accuracy: 0.8415471534115602
Average Precision: 0.8217198587139011
Average Recall: 0.6474326661724734

Overall Multinomial Classification Results:
Average Accuracy: 0.7251361300713246
Average Precision: 0.689585261419235
Average Recall: 0.7251361300713246
Average F1-Score: 0.672445783398329
```



```
Binary Classification at learning rate 0.1
Binary Classification Results for each fold:


Fold 1- Accuracy:0.8564971751412429, Recall:0.6842105263157895, Precision:0.8405172413793104

Fold 2- Accuracy:0.848587570621469, Recall:0.6795774647887324, Precision:0.8177966101694916

Fold 3- Accuracy:0.831638418079096, Recall:0.6654929577464789, Precision:0.7777777777777778

Fold 4- Accuracy:0.8564971751412429, Recall:0.6830985915492958, Precision:0.8398268398268398

Fold 5- Accuracy:0.8529411764705882, Recall:0.7007042253521126, Precision:0.8155737704918032
Multinomial Classification at learning rate 0.1

Multinomial Classification Results for each fold:


Fold 1- Accuracy:0.7299435028248588, Recall:0.7299435028248588, Precision:0.6923224029886281, F1-Score:0.6907758539333387

Fold 2- Accuracy:0.7288135593220338, Recall:0.7288135593220338, Precision:0.7029875283616982, F1-Score:0.6840610280464412

Fold 3- Accuracy:0.7220338983050848, Recall:0.7220338983050848, Precision:0.6796672222826845, F1-Score:0.6796680582859368

Fold 4- Accuracy:0.7378531073446327, Recall:0.7378531073446327, Precision:0.7164175985870901, F1-Score:0.6978239665560795

Fold 5- Accuracy:0.7296380090497737, Recall:0.7296380090497737, Precision:0.6949507192102661, F1-Score:0.6873349772857958

Overall Binary Classification Results:
Average Accuracy: 0.8492323030907277
Average Precision: 0.8182984479290445
Average Recall: 0.6826167531504818

Overall Multinomial Classification Results:
Average Accuracy: 0.7296564153692768
Average Precision: 0.6972690942860733
Average Recall: 0.7296564153692768
Average F1-Score: 0.6879327768215184
```



```
Binary Classification at learning rate 1
Binary Classification Results for each fold:


Fold 1- Accuracy:0.8576271186440678, Recall:0.6912280701754386, Precision:0.8382978723404255

Fold 2- Accuracy:0.8519774011299435, Recall:0.6936619718309859, Precision:0.8174273858921162

Fold 3- Accuracy:0.8305084745762712, Recall:0.676056338028169, Precision:0.768

Fold 4- Accuracy:0.8576271186440678, Recall:0.6866197183098591, Precision:0.8405172413793104

Fold 5- Accuracy:0.8563348416289592, Recall:0.6971830985915493, Precision:0.8284518828451883
Multinomial Classification at learning rate 1

Multinomial Classification Results for each fold:


Fold 1- Accuracy:0.7310734463276836, Recall:0.7310734463276836, Precision:0.6950215203269992, F1-Score:0.6941284004616974

Fold 2- Accuracy:0.7299435028248588, Recall:0.7299435028248588, Precision:0.7040475194285398, F1-Score:0.6865120086579501

Fold 3- Accuracy:0.7175141242937854, Recall:0.7175141242937854, Precision:0.6729917827784598, F1-Score:0.6761369598999265

Fold 4- Accuracy:0.7401129943502824, Recall:0.7401129943502824, Precision:0.7235529584845874, F1-Score:0.7016421316895678

Fold 5- Accuracy:0.7296380090497737, Recall:0.7296380090497737, Precision:0.6948654523844278, F1-Score:0.6898192002663398

Overall Binary Classification Results:
Average Accuracy: 0.8508149909246618
Average Precision: 0.818538876491408
Average Recall: 0.6889498393872003

Overall Multinomial Classification Results:
Average Accuracy: 0.7296564153692768
Average Precision: 0.6980958466806028
Average Recall: 0.7296564153692768
Average F1-Score: 0.6896477401950964
```


Confusion Matrix at Learning rate 1
True label
Predicted label