Institute of Technology of Cambodia

Department Information Technology and Communication

REPORT AI PROJECT: Predict Students' Dropout and Academic Success Using Machine Learning Techniques

LECTURER: UN Lykong

Group: 06

Name	**ID**	**Algorithm**
SOR Sopheak	e20191066	Linear Regression
SUN Vysing	e 20191124	K-Nearest Neighbors
VEN Thon	e20191250	Decision Tree

2023 ~ 2024

Table of Contents

List of Figures

I. Introduction

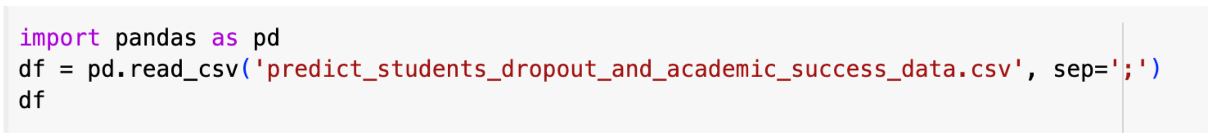
Ensuring quality education is a fundamental right, and governments globally strive to achieve universal enrollment and completion for every child. Despite concerted efforts, persistent dropout rates persist due to social, economic, and demographic factors. The identification of vulnerable groups is crucial for the government to implement targeted interventions effectively.

This project aims to conduct a comprehensive analysis of student dropout rates in school education using the dataset "Predict students' dropout and academic success" We utilized multiple machine learning models, namely KNN (k-Nearest Neighbors), Decision Tree, and Linear Regression, employing diverse methods to make accurate predictions.

II. Data analysis

2.1. Data preparation

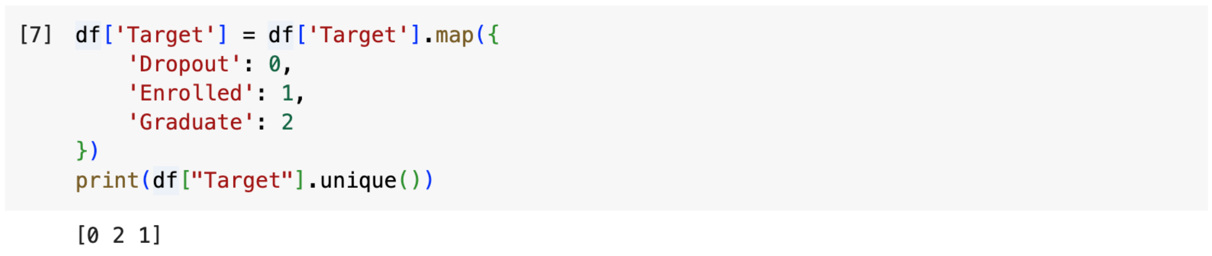
The dataset was imported for further analysis, by using panda as pd



```
import pandas as pd
df = pd.read_csv('predict_students_dropout_and_academic_success_data.csv', sep=';')
df
```

Figure 1: Import dataset

After importing the dataset, we noticed it needed cleaning. So, we made sure there were no missing values and converted the type of object of the column we want to predict (dropout, graduate, or enrolled) from words to numbers (0, 1, and 2).



```
[7]  df['Target'] = df['Target'].map({
         'Dropout': 0,
         'Enrolled': 1,
         'Graduate': 2
     })
     print(df["Target"].unique())

     [0 2 1]
```

Figure 2: Map categorical Target values to numerical values

We adjusted all values within selected columns ('Course', 'Previous qualification (grade)', 'Admission grade') to a similar scale by standardizing them using `scikit-learn's StandardScaler`. This ensures a consistent and optimized data representation for subsequent machine learning algorithms.

```
from sklearn.preprocessing import StandardScaler

column_to_scale = ['Course', 'Previous qualification (grade)', 'Admission grade']
X = df[column_to_scale]

scaler = StandardScaler()

# Fit and transform the data
scaled_X = scaler.fit_transform(X)

#Update DataFrame with scaled values
df[column_to_scale] = scaled_X
df
```

Figure 3: Scaling features 'Course', 'Previous qualification (grade)' and 'Admission grade'

Next, we checked how much the things we want to predict are connected to other info in the dataset.

```
df.corr()['Target']
```

```
Marital status                                    -0.089804
Application mode                                  -0.221747
Application order                                  0.089791
Course                                             0.034219
Daytime/evening attendance\t                       0.075107
Previous qualification                            -0.056039
Previous qualification (grade)                     0.103764
Nacionality                                       -0.014801
Mother's qualification                            -0.043178
Father's qualification                            -0.001393
Mother's occupation                               -0.005629
Father's occupation                               -0.001899
Admission grade                                    0.120889
Displaced                                          0.113986
Educational special needs                         -0.007353
Debtor                                            -0.240999
Tuition fees up to date                            0.409827
Gender                                            -0.229270
Scholarship holder                                 0.297595
Age at enrollment                                 -0.243438
International                                      0.003934
Curricular units 1st sem (credited)                0.048150
Curricular units 1st sem (enrolled)                0.155974
Curricular units 1st sem (evaluations)             0.044362
Curricular units 1st sem (approved)                0.529123
Curricular units 1st sem (grade)                   0.485207
Curricular units 1st sem (without evaluations)    -0.068702
Curricular units 2nd sem (credited)                0.054004
Curricular units 2nd sem (enrolled)                0.175847
Curricular units 2nd sem (evaluations)             0.092721
Curricular units 2nd sem (approved)                0.624157
Curricular units 2nd sem (grade)                   0.566827
Curricular units 2nd sem (without evaluations)    -0.094028
Unemployment rate                                  0.008627
Inflation rate                                    -0.026874
GDP                                                0.044135
Target                                             1.000000
Name: Target, dtype: float64
```

Figure 4: Check Correlation of Target column

Some columns had very little connection (from 0.001 to 0.07), so we removed those. If a column had a stronger connection, we kept it.

```
new_df = df.copy()
new_df = new_df.drop(columns=['Nacionality',
                              'Mother\'s qualification',
                              'Father\'s qualification',
                              'Mother\'s occupation',
                              'Father\'s occupation',
                              'Educational special needs',
                              'International',
                              'Curricular units 1st sem (without evaluations)',
                              'Unemployment rate',
                              'Inflation rate'], axis=1)
new_df.info()
```

Figure 5: Remove other unwanted or irrelevant features from the data

Now that our data is set up, we're ready to start getting it ready for the machine learning algorithms. Since we're predicting dropout, graduation, and enrollment, we split our data into train and test sets.

2.2. Apply machine learning algorithms

2.2.1. Linear Regression:

Linear regression aims to model the linear relationship between a continuous target variable (dependent variable) and one or more predictor variables (independent variables influencing the target). It estimates a line of best fit that minimizes the error between the predicted and actual values of the target variable. The goal is to predict student outcomes, such as dropout, graduation, and enrollment, based on various influencing factors in their daily lives. This involves training a linear regression model, evaluating its performance, and visualizing the results.

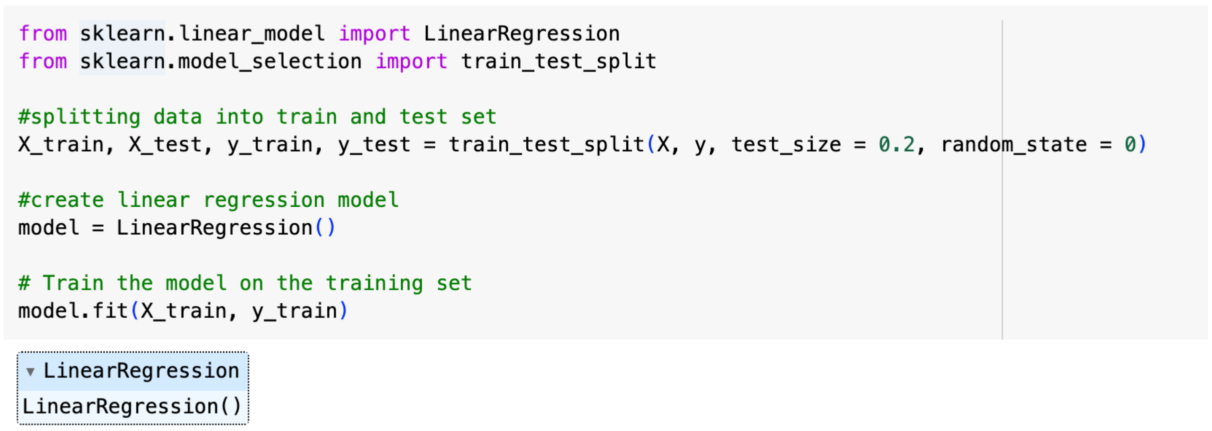
a. Training a Linear Regression Model

These two lines initialize a Linear Regression model and train it on the provided training set, allowing the model to learn patterns and relationships for predicting outcomes.

```
#Seperating Target varible
X = df.iloc[:, :-1].values
y = df['Target'].values
```

Figure 6: Target Variable Separation

Involving the dataset split into training and testing sets using `train_test_split`. Subsequently, a Linear Regression model is created and trained on the designated training set (X_train, y_train). The process visually represents key steps in the "Linear Regression Model Training Process" figure, including dataset splitting, model creation, and training.



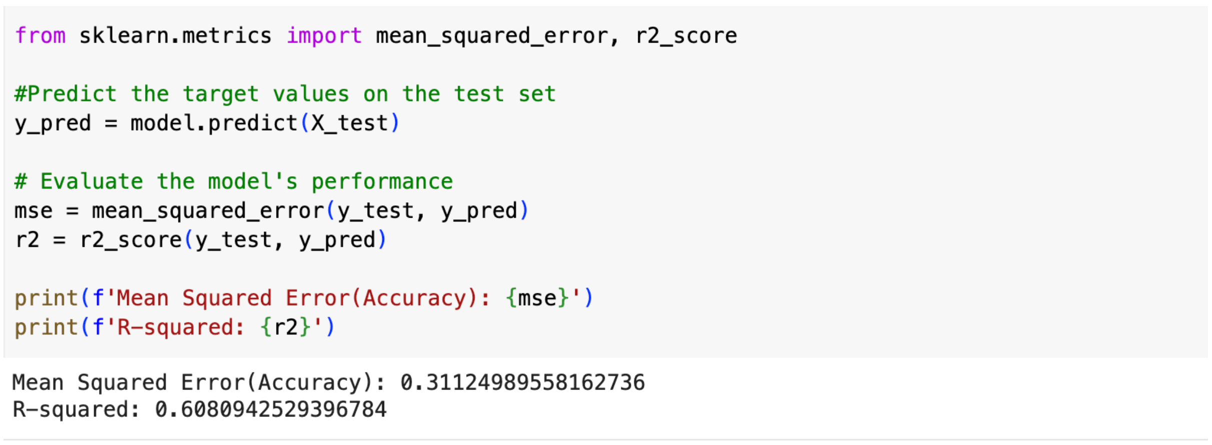
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

#splitting data into train and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)

#create linear regression model
model = LinearRegression()

# Train the model on the training set
model.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

Figure 7: Linear Regression Model Training Process

b. Evaluating Model Performance

After training the linear regression model, it is essential to assess its performance. This involves using a separate set of data (not used during training) to evaluate how well the model generalizes to new, unseen data. Common evaluation metrics include Mean Squared Error (MSE), R-squared, and others.



```
from sklearn.metrics import mean_squared_error, r2_score

#Predict the target values on the test set
y_pred = model.predict(X_test)

# Evaluate the model's performance
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error(Accuracy): {mse}')
print(f'R-squared: {r2}')
```

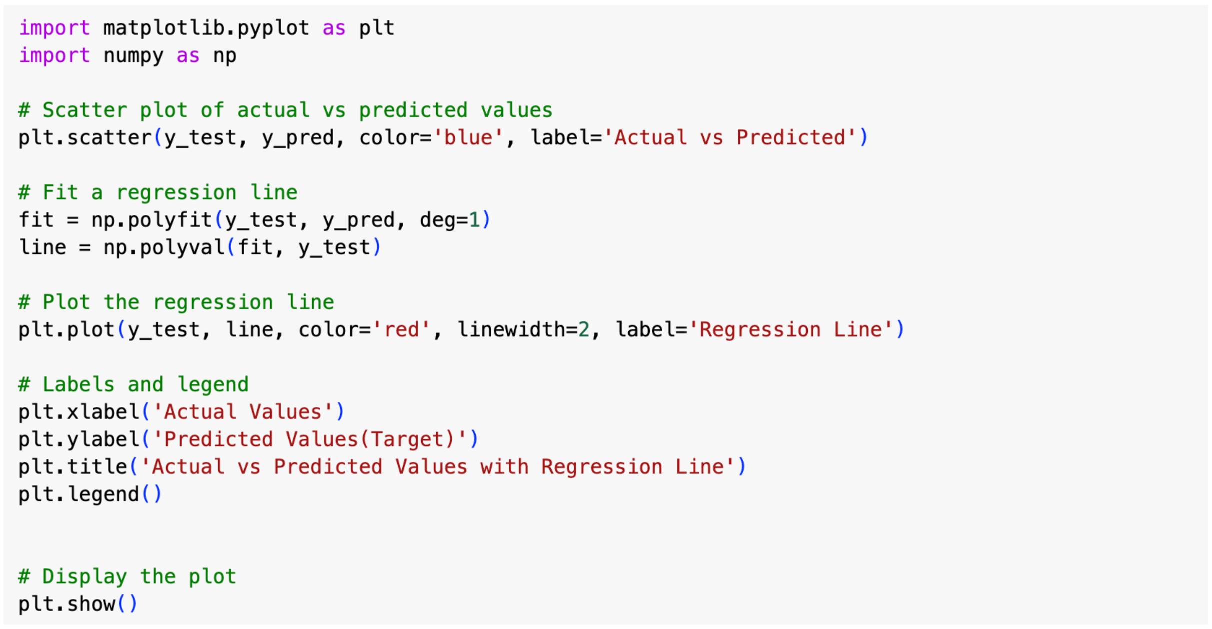
```
Mean Squared Error(Accuracy): 0.31124989558162736
R-squared: 0.6080942529396784
```

Figure 8: Model Performance Evaluation

The results of the Linear Regression model evaluation showcase a Mean Squared Error (MSE) of 0.3112 indicates that, on average, the squared difference between the actual and predicted values is relatively low, suggesting accurate predictions. The R-squared value of 0.6081 implies that approximately 60.81% of the variability in the target variable is explained by the model, reflecting a substantial degree of predictive capability.

c. Visualizing Results

Visualizations play a crucial role in understanding and interpreting the results of a linear regression model. Graphical representations, such as scatter plots and regression lines, help visualize the relationship between predictors and the target variable. Visualization aids in communicating insights and patterns discovered by the model.



```python
import matplotlib.pyplot as plt
import numpy as np

# Scatter plot of actual vs predicted values
plt.scatter(y_test, y_pred, color='blue', label='Actual vs Predicted')

# Fit a regression line
fit = np.polyfit(y_test, y_pred, deg=1)
line = np.polyval(fit, y_test)

# Plot the regression line
plt.plot(y_test, line, color='red', linewidth=2, label='Regression Line')

# Labels and legend
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values(Target)')
plt.title('Actual vs Predicted Values with Regression Line')
plt.legend()


# Display the plot
plt.show()
```

Figure 9: Actual vs Predicted Values with Regression Line

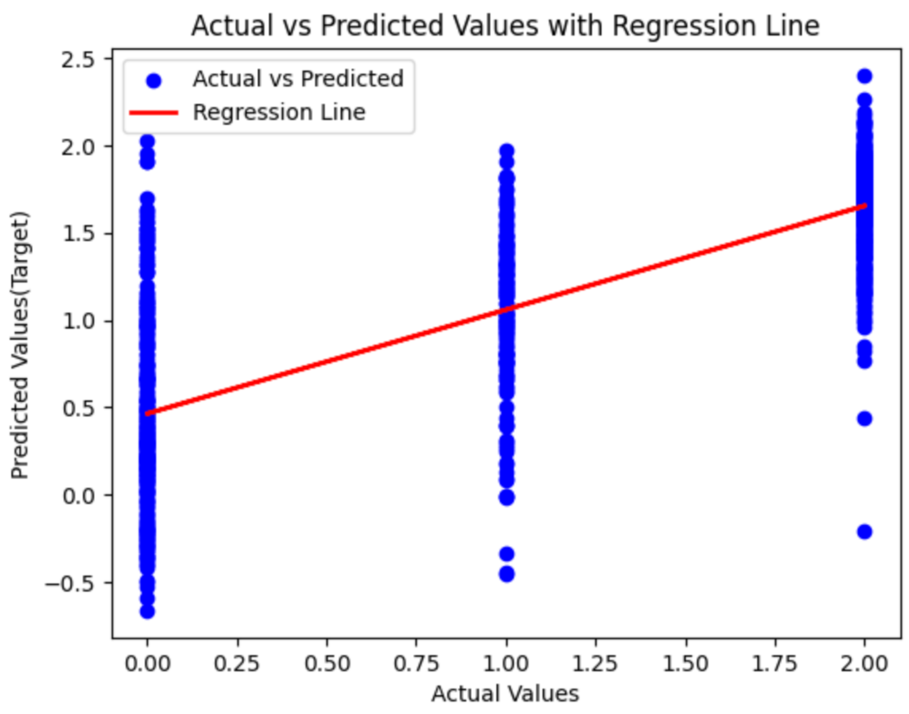


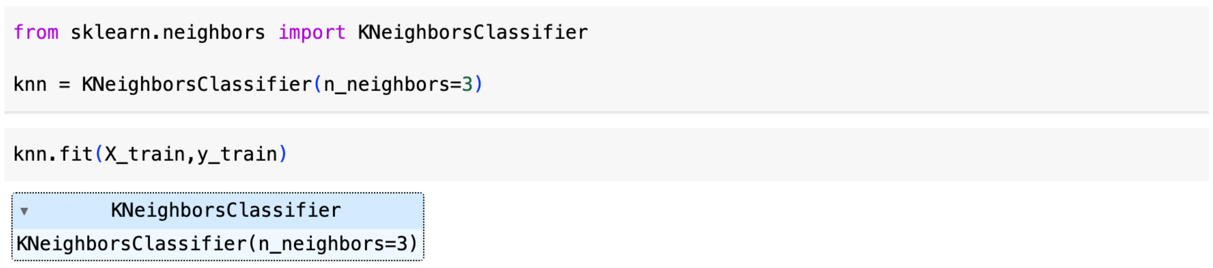
Figure 10: Graph Actual vs Predicted Values with Regression Line

2.2.2. KNN (k-Nearest Neighbors)

KNN, or k-Nearest Neighbors, is a non-parametric machine learning algorithm used for classification and regression tasks. It operates by finding the 'k' training examples closest in distance to a new data point and predicts the target variable based on the majority class (for classification) or average (for regression) of these neighbors.

a. Training a KNN Model

Initialize the KNN model with the chosen 'k' value, which specifies the number of neighbors considered when making predictions. The classifier is initialized with three neighbors (n_neighbors=3), and it is then trained on the training set (X_train, y_train).



```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=3)
```

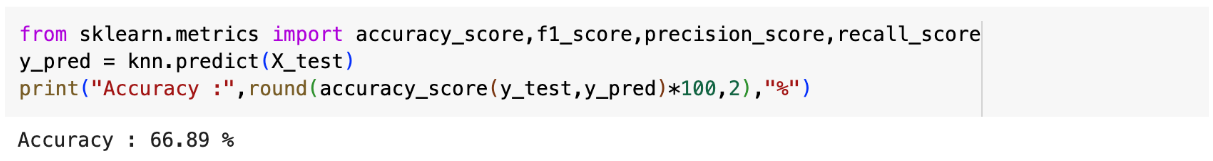
```
knn.fit(X_train,y_train)
```

```
▾       KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

Figure 11: KNN Model Initialization and Training

b. Evaluating Model Performance

The following code assesses the performance of the KNN model on the test set. It calculates accuracy and prints the result.



```
from sklearn.metrics import accuracy_score,f1_score,precision_score,recall_score
y_pred = knn.predict(X_test)
print("Accuracy :",round(accuracy_score(y_test,y_pred)*100,2),"%")
```

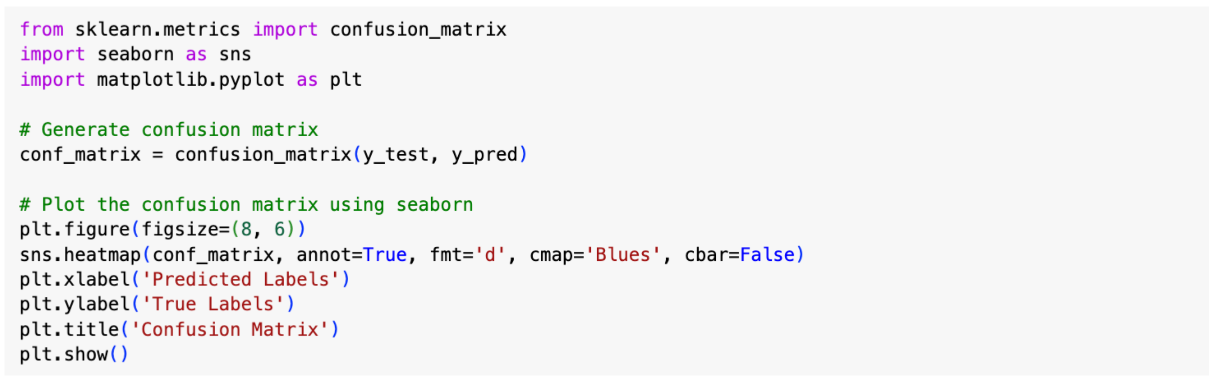
```
Accuracy : 66.89 %
```

Figure 12: KNN Model Evaluation (Accuracy)

The accuracy of the KNN model is a metric that indicates the percentage of correctly classified instances out of the total instances in the test set. In this case, an accuracy of 66.89% suggests that the KNN model correctly predicted the target variable for approximately two-thirds of the test instances.

c. Visualizing Results

This code segment generates a confusion matrix to visualize the performance of the KNN model. It uses seaborn to create a heatmap of the confusion matrix, providing insights into the model's classification performance.



```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Generate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix using seaborn
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

Figure 13: Confusion Matrix for KNN Model

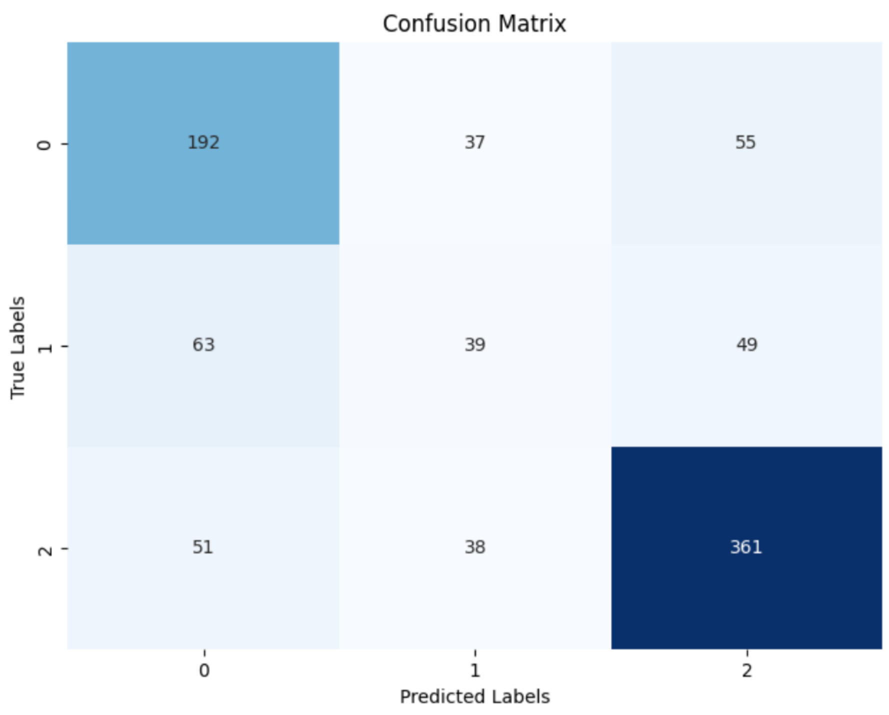


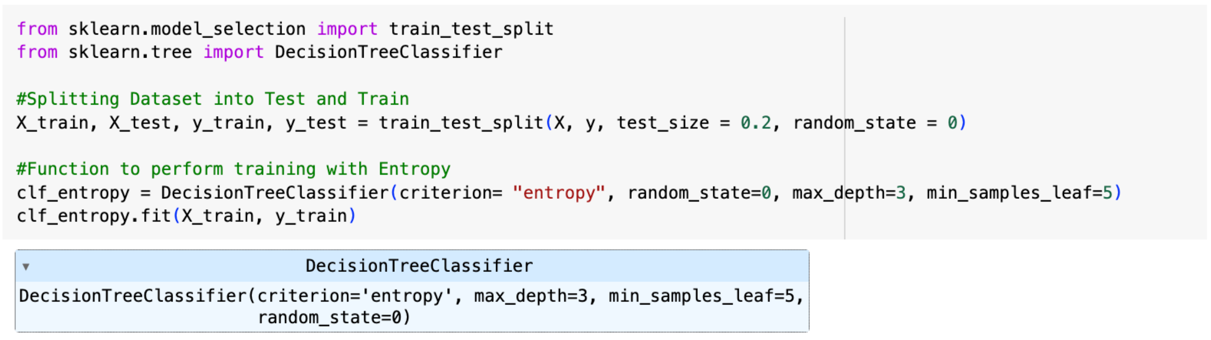
Figure 14: Graph Confusion Matrix for KNN Model

2.2.3. Decision Tree

Decision Trees are versatile and widely used machine learning algorithms that can perform both classification and regression tasks and used to predict the probability of a categorical dependent variable. These trees consist of nodes representing decisions based on input features, leading to outcomes represented by leaves. The tree structure is constructed through a process that selects the best features to split on at each node.

a. Training a Decision Tree Model

This code segment involves splitting the dataset into training and testing sets using the `train_test_split` function. It then initializes a Decision Tree classifier with the criterion set to "entropy" indicating the use of information gain for node splitting. The model is then trained on the training set, configured with a maximum depth of 3 and a minimum of 5 samples per leaf.



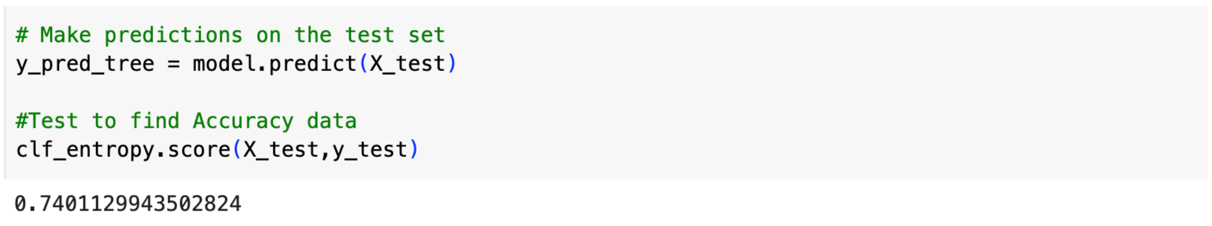
```python
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

#Splitting Dataset into Test and Train
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)

#Function to perform training with Entropy
clf_entropy = DecisionTreeClassifier(criterion= "entropy", random_state=0, max_depth=3, min_samples_leaf=5)
clf_entropy.fit(X_train, y_train)
```

```
▾                         DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=3, min_samples_leaf=5,
                       random_state=0)
```

Figure 15: Decision Tree Model Training Process

b. Evaluating Model Performance

This code section generates predictions (y_pred_tree) using the trained Decision Tree model on the test set. It then calculates the accuracy score of the model on the test data using the score method, providing insights into the model's predictive performance.



```
# Make predictions on the test set
y_pred_tree = model.predict(X_test)

#Test to find Accuracy data
clf_entropy.score(X_test,y_test)
```

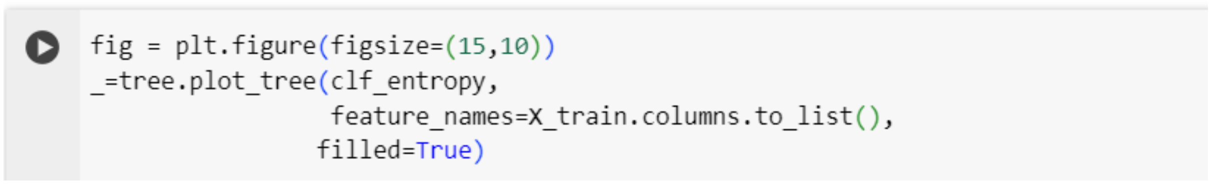
```
0.7401129943502824
```

Figure 16: Decision Tree Model Accuracy

The accuracy score of 0.7401 indicates that the Decision Tree model achieved an accuracy of approximately 74.01% on the test set. This score represents the proportion of correctly predicted outcomes in comparison to the total number of instances in the test data. A higher accuracy score suggests better predictive performance, showcasing the model's ability to make accurate predictions on unseen data.

c. Visualizing Results

Visualizations play a crucial role in understanding and interpreting the results of a Decision Tree model. Overall, this a figure for plotting, then uses tree.plot_tree() to visualize the decision tree classifier clf_entropy using the specified features and a filled color representation for the nodes. This visualization would show the structure of the decision tree, its nodes, splits, and leaf nodes, providing insights into how the classifier is making decisions based on the provided features.



```
fig = plt.figure(figsize=(15,10))
_=tree.plot_tree(clf_entropy,
                 feature_names=X_train.columns.to_list(),
                filled=True)
```

Figure 17: Predicted Values using entropy

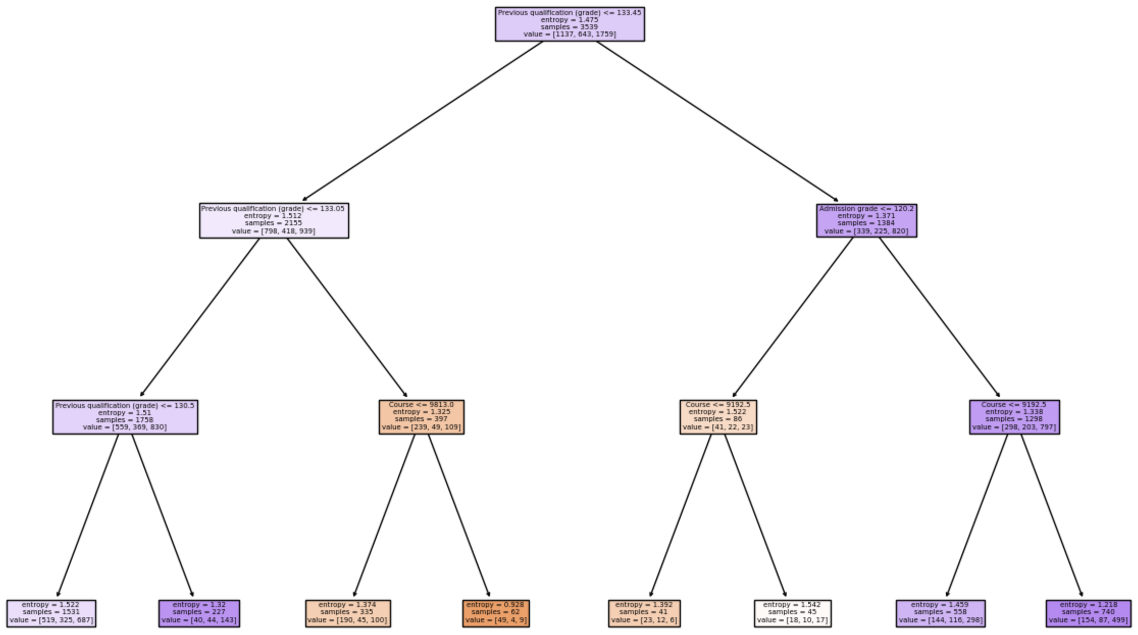


Figure 18: Graph Predicted Values using entropy

III. Conclusion

In conclusion, we applied machine learning models such as Linear Regression, Decision Tree, and K-Nearest Neighbors (KNN to predict student outcomes in the 'Predict students' dropout and academic success' dataset.

Linear Regression achieved accuracy with an MSE of 31.12% and an R-squared value of 60.81%, excelling in capturing linear relationships. Decision Tree, configured with entropy-based splitting, demonstrated accuracy 74.01% and provided insights into feature importance. KNN, with k=3 neighbors, achieved 66.89% accuracy, relying on similarities in feature space.

The choice between these models depends on the project's objectives and dataset characteristics. Linear Regression suits scenarios with prevalent linear relationships, Decision Tree excels in capturing non-linear patterns and showcasing feature importance, while KNN is effective for datasets where instances with similar features tend to have similar outcomes. Considerations such as interpretability, computational efficiency, and project requirements play a crucial role in selecting the most suitable model. Ultimately, the decision should align with the balance of these considerations and the specific goals of the predictive modeling project.