SCHOOL OF ENGINEERING AND TECHNOLOGY

ASSESSMENT FOR THE MASTER OF DATA SCIENCE

SUBJECT CODE AND TITLE:		**MDM5063 DATA MINING**
ASSESSMENT DUE DATE:		**18/10/2024**
NO.	**STUDENT ID**	**STUDENT NAME**
1	**14086334**	**KAN JUN FAI**

IMPORTANT

The University requires students to adhere to submission deadlines for any form of assessment. Penalties are applied in relation to unauthorised late submission of work. Coursework submitted after the deadline will be subjected to the prevailing academic regulations. Please check your respective programme handbook.

Academic Honesty Acknowledgement

"I **KAN JUN FAI** (name) verify that this paper contains entirely my own work. I have not consulted with any outside person or materials other than what was specified (an interviewee, for example) in the assignment or the syllabus requirements. Further, I have not copied or inadvertently copied ideas, sentences, or paragraphs from another student. I realise the penalties *(refer to the student handbook and undergraduate programme handbook)* for any kind of copying or collaboration."

Fai .... (Student' Signature / Initial)

1. Problem Formulation

In modern day education systems, educators subject assessments towards their students to evaluate the understanding of concepts and applications of the subjects. The student performance serves as feedback to gauge the effectiveness of their teaching methods on their students, and adjust accordingly in order to ensure all students comprehend the learning materials. As each and every student is unique in their own pace of learning, the feedback is especially important to educators in order to create materials and teaching methods that are engaging and effective towards each unique group of students. Educators will need to assess their students on several factors – the comprehension in each specific topic, the learning progression, the ability to critically think and apply the knowledge in projects / scenarios – in order to understand in detail the strength and weaknesses of their students, and tailor teaching methods to address the knowledge and teaching gap, ensuring that each student not only possess the ability to understand the materials, but most importantly to retain the knowledge and application skills for the long term.

Educators usually face a class of thirty to hundreds of students at a time, which give rise to the challenge of early identification of students who struggle with the learning content. As each student's capabilities and learning pace differs, educators also require assistance in providing specific time and resources towards different groups of students in order to effectively guide them accordingly. In this case, classification machine learning models can help to analyse student data in order to serve as a prediction system that can aid educators in identifying struggling students so that intervention strategies can be implemented early.

Supervised learning models are traditionally the 'gold standard' of machine learning used to study and accurately predict student who fall within the at-risk category of failing. Among the many supervised learning models, Convolutional Neural Network (CNN) was proven to be the most effective model in accurately predicting student performance. In this study by Mohammed and colleagues, the academic performance – math, reading, and writing skills – and student participation frequency in class were used as features in training and testing the CNN classification model in a (80/20) training-testing split. Results show a staggering 97.85% model accuracy in predicting student performances among their 9 tested datasets (Ahmad Saeed Mohammad et al., 2023). These identified group of at-risk students were then subjected to additional learning support and early intervention, significantly improving the students' performance as well as assisting educators to incorporate different learning strategies with the purpose to improve education quality (Khan et al., 2021).

While supervised learning emphasized on inputs of potential features in order to classify students performance, unsupervised learning models in educational data mining are also developing rapidly, where educators are able to identify groups of students with similar features or behaviour. In this study by that currently an AI model named FIRST had been heavily trained to assist educators using temporal analysis on features that influence student GPA, and identifying patterns exhibited by groups of students that influence their learning ability, and lastly providing insights towards educators in an interactive storytelling manner to better tailor an intervention towards students that are at-risk (Al-Doulat et al., 2020). This study by Mohamed Nafuri and colleagues also highlights the application of unsupervised learning models specifically k-means, BIRCH, and DBSCAN were shown to effectively group students who are at risk among a dataset of Malaysian students based on behavioural patterns exhibited. K-means clustering model had the best overall performance among the three models. Due to a large sample size, optimized parameters, and numerical features, K-means model was able to effectively separate students into distinctive clusters according to specified number of clusters, as evidenced by the study's DB index, and silhouette index values (Mohamed Nafuri et al., 2022). The nature of unsupervised learning models also contributed significantly towards discovery of hidden patterns that contribute to the underperformance of the group of students. In this study by Peach and colleagues, the temporal analysis of the online students uncovered differences in learning strategies through the students' engagement behaviours; Some groups of students completed large portions of learning materials within a short amount of time (massed learning), versus another group of students who have close-to-evenly distributed timing of learning, which contribute to the impact of the learner's performance (Peach et al., 2019).

With the proven efficiency from CNN and K-means clustering respectively, this project aims to compare & contrast two models – CNN alone only, vs. K-means clustering with CNN – in classifying the “high performer” , “underperformer”, and “average performer” categories. Based on the selected dataset of 2,000 students, the best model will be evaluated with the purpose of determining if exploring the future potential benefits of combining CNN together with K-means clustering classification in addressing student performance is feasible and worth it.

2. Problem and Data Understanding

Problem Understanding:

Can this project develop an improved classification machine learning model that can accurately identify respectively the population into three categories of students – the “high performers”, “under performers”, and “average performers” – by comparing the performance of CNN alone only, vs. K-means clustering with CNN? The goal is to compare and contrast, and select the best performing model, by evaluating the performance metrics of accuracy, precision, recall, F1-score, confusion matrix, ROC curve & AUC, and lastly Silhouette Score for K-means clustering. The classification model will be trained using the dataset features such as students having a part-time job, absence days, attending extracurricular activities, weekly self-study hours, and average score. The classification model can be trained and tested for the use of assisting educators to accurately predict students who perform well (high performers), and most importantly students who are struggling and are at risk of failure or dropping out (under performers) in order to apply early intervention strategies. The ideal outcome is that the K-means clustering with CNN classification model performs within an ambitious range of 10 – 20 % significantly more effective than the CNN only classification model (based on the performance metrics mentioned) in distinctively performing clustering .

Several challenges that are expected are that the dataset features (refer to Table below in ‘Data Understanding’), such as the data features contain limited in-depth information on how it may affect students’ focus, engagement, and comprehension on the learning materials. Moreover, there are also no context of the experience level of the students, as well as no temporal data on the students in order to accurately and realistically understand the student background and evaluate progression of their learning. In relation to challenges in the models proposed, the interpretation of the performance may be complex; it may be challenging to pinpoint how K-means clustering affect how the classification model will perform post-clustering. With this additional dimension of complexity, parameter selection for performance tuning of the model in order to improve performance becomes challenging and may cause unexpected trade-offs between the performance metrics.

Data Understanding:

The dataset of 2,000 students titled "Student Scores" from Kaggle were used in this project (Medhat, M. (n.d.)). Among the data features in the Table below, the categorical values – part-time job, extracurricular activities – and numerical attributes – absence days, weekly self-study hours, all individual subject scores, and average score – will be the main features examined for data mining for the student performance classification model.

Features	**Description**
ID	Unique identifier for each student.
First Name	The first name of the student.
Last Name	The last name of the student.
Email	The email address of the student.
Gender	The gender of the student (e.g., male, female).
Part-Time Job	Boolean indicating if the student has a part-time job.
Absence Days	Number of days the student was absent.
Extracurricular Activities	Boolean indicating if the student participates in extracurricular activities.
Weekly Self-Study Hours	Average number of self-study hours per week.
Career Aspiration	The student's career aspirations (e.g., doctor, engineer).
Math Score	Score in Mathematics.
History Score	Score in History.
Physics Score	Score in Physics.
Chemistry Score	Score in Chemistry.
Biology Score	Score in Biology.
English Score	Score in English.
Geography Score	Score in Geography.

Table : Features and its description in the dataset.

3. Data Preparation

The dataset dimension contain 2,000 records and 17 fields. The dataset domain of the mentioned records and fields are shown in image below. There were no null and duplicate values found in this dataset.

```
In [1]: import pandas as pd
        student_score = pd.read_csv(r'C:\Users\Admin\Downloads\Kaggle\student-scores.csv')
```

```
In [2]: print("DATASET DIMENSION -", "Student_score dataset: ", student_score.shape)
        print("")

        print("DATASET DOMAIN",)
        print(student_score.info())
        print("")
        print(student_score.head())
```

```
DATASET DIMENSION - Student_score dataset:  (2000, 17)

DATASET DOMAIN
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 17 columns):
 #   Column                      Non-Null Count  Dtype 
---  ------                      --------------  ----- 
 0   id                          2000 non-null   int64 
 1   first_name                  2000 non-null   object
 2   last_name                   2000 non-null   object
 3   email                       2000 non-null   object
 4   gender                      2000 non-null   object
 5   part_time_job               2000 non-null   bool  
 6   absence_days                2000 non-null   int64 
 7   extracurricular_activities  2000 non-null   bool  
 8   weekly_self_study_hours     2000 non-null   int64 
 9   career_aspiration           2000 non-null   object
 10  math_score                  2000 non-null   int64 
 11  history_score               2000 non-null   int64 
 12  physics_score               2000 non-null   int64 
 13  chemistry_score             2000 non-null   int64 
 14  biology_score               2000 non-null   int64 
 15  english_score               2000 non-null   int64 
 16  geography_score             2000 non-null   int64 
dtypes: bool(2), int64(10), object(5)
memory usage: 238.4+ KB
None

   id first_name last_name                              email  gender  \
0   1       Paul     Casey       paul.casey.1@gslingacademy.com    male   
1   2   Danielle  Sandoval  danielle.sandoval.2@gslingacademy.com  female   
2   3       Tina   Andrews     tina.andrews.3@gslingacademy.com  female   
3   4       Tara     Clark       tara.clark.4@gslingacademy.com  female   
4   5    Anthony    Campos     anthony.campos.5@gslingacademy.com    male   

   part_time_job  absence_days  extracurricular_activities  \
0          False             3                       False   
1          False             2                       False   
2          False             9                        True   
3          False             5                       False   
4          False             5                       False   

   weekly_self_study_hours   career_aspiration  math_score  history_score  \
0                       27              Lawyer          73             81   
1                       47              Doctor          90             86   
2                       13  Government Officer          81             97   
3                        3              Artist          71             74   
4                       10             Unknown          84             77   

   physics_score  chemistry_score  biology_score  english_score  \
0             93               97             63             80   
1             96              100             90             88   
2             95               96             65             77   
3             88               80             89             63   
4             65               65             80             74   

   geography_score  
0               87  
1               90  
2               94  
3               86  
4               76  
```

```
In [3]: # Checking for duplicates

        print(student_score[student_score.duplicated()])        ## no duplicate values found.
```

```
Empty DataFrame
Columns: [id, first_name, last_name, email, gender, part_time_job, absence_days, extracurricular_activities, weekly_self_study_hours, career_aspiration, math_score, history_score, physics_score, chemistry_score, biology_score, english_score, geography_score]
Index: []
```

Data Preprocessing:

The data preparation starts with removing the unnecessary fields of name, email, and career aspiration. Next, the field "average score" was added for further analysis in the following steps.

```
In [4]: ## Drop columns "first_name", "last_name", "email", "career_aspiration" ##

        ss_cleaned = student_score.drop(columns=["first_name", "last_name", "email", "career_aspiration"])

        ## Creating new column for average scores ##

        subjects = ['math_score', 'history_score', 'physics_score', 'chemistry_score', 'biology_score', 'engli

        ss_cleaned['average_score'] = ss_cleaned[subjects].mean(axis=1)

        print(ss_cleaned.head())
        print("")
        print(ss_cleaned.info())
```

```
   id  gender  part_time_job  absence_days  extracurricular_activities  \
0   1    male          False             3                       False
1   2  female          False             2                       False
2   3  female          False             9                        True
3   4  female          False             5                       False
4   5    male          False             5                       False

   weekly_self_study_hours  math_score  history_score  physics_score  \
0                       27          73             81             93
1                       47          90             86             96
2                       13          81             97             95
3                        3          71             74             88
4                       10          84             77             65

   chemistry_score  biology_score  english_score  geography_score  \
0               97             63             80               87
1              100             90             88               90
2               96             65             77               94
3               80             89             63               86
4               65             80             74               76

   average_score
0      82.000000
1      91.428571
2      86.428571
3      78.714286
4      74.428571

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 14 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   id                          2000 non-null   int64
 1   gender                      2000 non-null   object
 2   part_time_job               2000 non-null   bool
 3   absence_days                2000 non-null   int64
 4   extracurricular_activities  2000 non-null   bool
 5   weekly_self_study_hours     2000 non-null   int64
 6   math_score                  2000 non-null   int64
 7   history_score               2000 non-null   int64
 8   physics_score               2000 non-null   int64
 9   chemistry_score             2000 non-null   int64
 10  biology_score               2000 non-null   int64
 11  english_score               2000 non-null   int64
 12  geography_score             2000 non-null   int64
 13  average_score               2000 non-null   float64
dtypes: bool(2), float64(1), int64(10), object(1)
memory usage: 191.5+ KB
None
```

The data statistics for numerical attributes, categorical attributes, skewness and kurtosis are shown below. In the numerical attributes, the students have low mean absence days of 3.66 days, mean self-studying hours of 17.75 hours, and high mean “average_score” of 80.98.

The categorical attributes are also analysed. The number of male and female are relatively similar, indicating no gender bias. In terms of part-time job status, only 316 (15.8%) students have jobs while the remaining 1,684 (84.2%) do not. Similar patterns are shown in extracurricular activities status, only 408 (20.4%) students participate, while 1,592 (79.6%) do not. The takeaway from this information is that most students in this dataset are focused on their studies, rather than juggling between work and studying, as well as participating in extracurricular activities.

```
In [6]: import warnings
        warnings.filterwarnings('ignore')

        # Basic statistics for numerical columns (count, mean, std, min, max, 25%, 50%, 75%)
        print(ss_cleaned.describe())

        # Skewness and kurtosis of particular columns
        print('-----------Skewness-------------')
        print(ss_cleaned.skew())
```

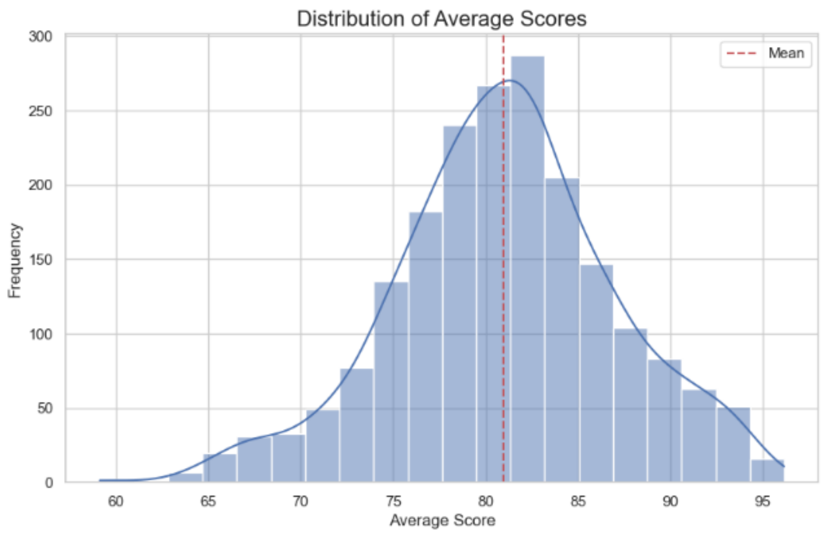
```
                id  absence_days  weekly_self_study_hours   math_score  \
count  2000.000000   2000.000000              2000.000000  2000.000000
mean   1000.500000      3.665500                17.755500    83.452000
std     577.494589      2.629271                12.129604    13.224906
min       1.000000      0.000000                 0.000000    40.000000
25%     500.750000      2.000000                 5.000000    77.000000
50%    1000.500000      3.000000                18.000000    87.000000
75%    1500.250000      5.000000                28.000000    93.000000
max    2000.000000     10.000000                50.000000   100.000000

       history_score  physics_score  chemistry_score  biology_score  \
count    2000.000000    2000.000000      2000.000000     2000.00000
mean       80.332000      81.336500        79.995000       79.58150
std        12.736046      12.539453        12.777895       13.72219
min        50.000000      50.000000        50.000000       30.00000
25%        69.750000      71.000000        69.000000       69.00000
50%        82.000000      83.000000        81.000000       81.00000
75%        91.000000      92.000000        91.000000       91.00000
max       100.000000     100.000000       100.000000      100.00000

       english_score  geography_score  average_score
count    2000.000000      2000.000000    2000.000000
mean       81.277500        80.888000      80.980357
std        12.027087        11.637705       6.042224
min        50.000000        60.000000      59.142857
25%        72.000000        71.000000      77.285714
50%        83.000000        81.000000      81.000000
75%        91.000000        91.000000      84.714286
max        99.000000       100.000000      96.142857
-----------Skewness-------------
id                             0.000000
part_time_job                  1.876711
absence_days                   0.767021
extracurricular_activities     1.470199
weekly_self_study_hours        0.138065
math_score                    -1.090145
history_score                 -0.269966
physics_score                 -0.346301
chemistry_score               -0.201933
biology_score                 -0.529917
english_score                 -0.456268
geography_score               -0.097094
average_score                 -0.142538
dtype: float64
```

```
In [7]: ## Exploring Categorical and Binary Attributes' Data

        print(ss_cleaned['gender'].value_counts())
        print("")
        print(ss_cleaned['part_time_job'].value_counts())
        print("")
        print(ss_cleaned['extracurricular_activities'].value_counts())
```

```
female    1002
male       998
Name: gender, dtype: int64

False    1684
True      316
Name: part_time_job, dtype: int64

False    1592
True      408
Name: extracurricular_activities, dtype: int64
```

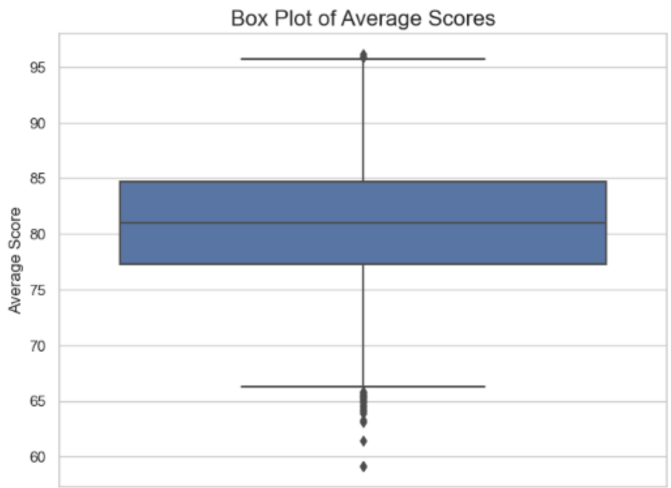
In the histogram below on average scores with the mean line drawn at value 80.98. The shape of the histogram indicates that the dataset is very close to normal distribution, with a slight left skew.

In [8]:
```python
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(style="whitegrid")

# Histogram of average scores
plt.figure(figsize=(10, 6))
sns.histplot(ss_cleaned['average_score'], bins=20, kde=True)
plt.title('Distribution of Average Scores', fontsize=16)
plt.xlabel('Average Score', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.axvline(ss_cleaned['average_score'].mean(), color='r', linestyle='--', label='Mean')
plt.legend()
plt.show()
```



With the boxplot visualisation below, it clearly indicates that there are outlier data points above and below the whisker line, which show that there are high performing students, and underperforming students that are at-risk of failing respectively.

In [9]:
```python
plt.figure(figsize=(8, 6))
sns.boxplot(y='average_score', data=ss_cleaned)
plt.title('Box Plot of Average Scores', fontsize=16)
plt.ylabel('Average Score', fontsize=12)
plt.show()
```



Preparing the Data for Implementation of Machine Learning Models

The categorical attributes of gender, part-time job, and extracurricular activities are converted into numerical values. Performance categories are also created that labels "High Performers" as 2, "Average Performers" as 1, and "Under Performers" as 0. These steps normalize and prepare for the use of these features in the CNN and K-means clustering models.

After creating the performance category, 478 students (23.9%) are high performers, 1,236 students (61.8%) are average performers, and 286 students (14.3%) are under performers.

```
In [10]: ### Data Preparation for Implementing Machine Learning Models
import numpy as np

# GENDER : "1" for Male , "2" for Female
ss_cleaned['gender'] = ss_cleaned['gender'].map({'male': 1, 'female': 2})

## PART_TIME_JOB : "0" for False , "1" for True
ss_cleaned['part_time_job'] = ss_cleaned['part_time_job'].astype(int)

### EXTRACURRICULAR_ACTIVITIES : "0" for False , "1" for True
ss_cleaned['extracurricular_activities'] = ss_cleaned['extracurricular_activities'].astype(int)

# Create the performance categories
#   2 - High Performer    (Average Score >= 85)
#   1 - Average Performer
#   0 - Under Performer   (Average Score <75)

def categorize_performance(row):
    if row['average_score'] >= 85:
        return 2  # High Performer
    elif row['average_score'] < 75:
        return 0  # Under Performer
    else:
        return 1  # Average Performer

# Apply the function to create the performance column
ss_cleaned['performance'] = ss_cleaned.apply(categorize_performance, axis=1)

print(ss_cleaned.head())

   id  gender  part_time_job  absence_days  extracurricular_activities  \
0   1       1              0             3                           0
1   2       2              0             2                           0
2   3       2              0             9                           1
3   4       2              0             5                           0
4   5       1              0             5                           0

   weekly_self_study_hours  math_score  history_score  physics_score  \
0                       27          73             81             93
1                       47          90             86             96
2                       13          81             97             95
3                        3          71             74             88
4                       10          84             77             65

   chemistry_score  biology_score  english_score  geography_score  \
0               97             63             80               87
1              100             90             88               90
2               96             65             77               94
3               80             89             63               86
4               65             80             74               76

   average_score  performance
0      82.000000            1
1      91.428571            2
2      86.428571            2
3      78.714286            1
4      74.428571            0
```

```
In [21]: print(ss_cleaned['performance'].value_counts())

#   1 - Average Performer
#   2 - High Performer   (Average Score >= 85)
#   0 - Under Performer  (Average Score <75)

1    1236
2     478
0     286
Name: performance, dtype: int64
```

4. Rationale for Data Modelling and Evaluation

CNN has been selected due to its best performance among mentioned studies cited. The reason to implement K-means clustering along with CNN is to apply data clustering, and then applying the newly-cluster labelled data into CNN to provide an additional structure that reduces ambiguity among overlapping data, which may improve the classification model's performance metrics. With the improved labelling of K-means clustering, CNN's can learn more efficiently through its mechanism of convolution layers, especially when the dataset has multiple features and combinations, leading to improved classification performance.

5. Data Modelling and Evaluation phases

In [12]:

```
import numpy as np
import pandas as pd
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, roc_curve, auc
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt
```

In [13]:

```
#  Model Building (CNN only)


# Drop features irrelevant for classification
X = ss_cleaned.drop(columns=['id', 'performance', 'average_score']).values
y = ss_cleaned['performance'].values

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

# Convert target label 'performance' to categorical (for 3 categories classification)
y_categorical = to_categorical(y, num_classes=3)

# Reshape data to match 3D format of CNN (samples, time steps, features)
X_scaled = X_scaled.reshape(X_scaled.shape[0], X_scaled.shape[1], 1)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_categorical, test_size=0.2, random_state=10)

# Define the CNN model for 3 categories classification
model = Sequential([
    Conv1D(64, kernel_size=2, activation='relu', input_shape=(X_train.shape[1], 1)),
    MaxPooling1D(pool_size=2),
    Flatten(),
    Dense(100, activation='relu'),
    Dense(3, activation='softmax')
])

# Model Compilation
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Model Training
history = model.fit(X_train, y_train, epochs=20, validation_data=(X_test, y_test), verbose=1)

# Evaluate model on the test set
y_test_pred_probs = model.predict(X_test)
y_test_pred = np.argmax(y_test_pred_probs, axis=1)
y_test_true = np.argmax(y_test, axis=1)

# Model Evaluation
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Accuracy: {test_accuracy}")
```

```
Epoch 1/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.6470 - loss: 0.8075 - val_accuracy: 0.8600 - val_loss: 0.4122
Epoch 2/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8552 - loss: 0.3791 - val_accuracy: 0.8700 - val_loss: 0.3338
Epoch 3/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8805 - loss: 0.3034 - val_accuracy: 0.8600 - val_loss: 0.3171
Epoch 4/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8732 - loss: 0.2807 - val_accuracy: 0.8500 - val_loss: 0.3349
Epoch 5/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8730 - loss: 0.2903 - val_accuracy: 0.8150 - val_loss: 0.3836
Epoch 6/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8838 - loss: 0.2706 - val_accuracy: 0.8575 - val_loss: 0.3125
Epoch 7/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8647 - loss: 0.2945 - val_accuracy: 0.8625 - val_loss: 0.3057
Epoch 8/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8717 - loss: 0.2697 - val_accuracy: 0.8650 - val_loss: 0.3086
Epoch 9/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8836 - loss: 0.2847 - val_accuracy: 0.8475 - val_loss: 0.3172
Epoch 10/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8805 - loss: 0.2715 - val_accuracy: 0.8500 - val_loss: 0.3213
Epoch 11/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8640 - loss: 0.2913 - val_accuracy: 0.8650 - val_loss: 0.3046
Epoch 12/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8894 - loss: 0.2499 - val_accuracy: 0.8600 - val_loss: 0.3187
Epoch 13/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8616 - loss: 0.2841 - val_accuracy: 0.8675 - val_loss: 0.3058
Epoch 14/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8799 - loss: 0.2595 - val_accuracy: 0.8650 - val_loss: 0.2932
Epoch 15/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8783 - loss: 0.2722 - val_accuracy: 0.8650 - val_loss: 0.2922
Epoch 16/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8798 - loss: 0.2597 - val_accuracy: 0.8700 - val_loss: 0.2997
Epoch 17/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8710 - loss: 0.2760 - val_accuracy: 0.8750 - val_loss: 0.3028
Epoch 18/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8903 - loss: 0.2516 - val_accuracy: 0.8700 - val_loss: 0.2954
Epoch 19/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8695 - loss: 0.2840 - val_accuracy: 0.8650 - val_loss: 0.3090
Epoch 20/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9014 - loss: 0.2262 - val_accuracy: 0.8600 - val_loss: 0.3060
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
Test Accuracy: 0.8600000143051147
```

In [14]:

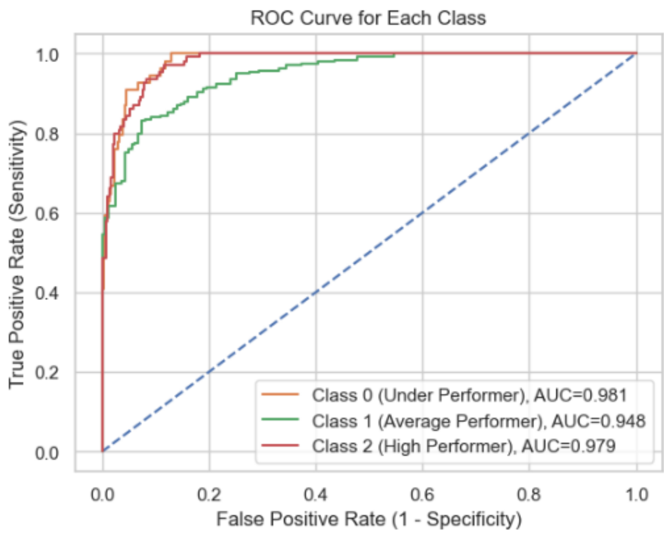
```
#  Accuracy, Precision, Recall, F1-score
accuracy = accuracy_score(y_test_true, y_test_pred)
precision = precision_score(y_test_true, y_test_pred, average='macro')
recall = recall_score(y_test_true, y_test_pred, average='macro')
f1 = f1_score(y_test_true, y_test_pred, average='macro')

print(f'Accuracy: {accuracy:.4f}')
print(f'Precision: {precision:.4f}')
print(f'Recall: {recall:.4f}')
print(f'F1-Score: {f1:.4f}')

# Using metrics' function parameters to derive performance measures
acc = accuracy_score(y_test_true, y_test_pred)
prec = precision_score(y_test_true, y_test_pred, average='macro')
sens = recall_score(y_test_true, y_test_pred, average='macro')
f1 = f1_score(y_test_true, y_test_pred, average='macro')

# Show all performance metrics
print("Accuracy : ", round(acc, 3))
print("Precision : ", round(prec, 3))
print("Sensitivity/Recall) : ", round(sens, 3))
print("F1-Score : ", round(f1, 3))
print("Misclassification : ", round(1 - acc, 3))

print('----------------------------------------')

# Confusion Matrix
conf_matrix = confusion_matrix(y_test_true, y_test_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Accuracy: 0.8600
Precision: 0.8574
Recall: 0.8055
F1-Score: 0.8281
Accuracy :  0.86
Precision :  0.857
Sensitivity/Recall) :  0.806
F1-Score :  0.828
Misclassification :  0.14
----------------------------------------
Confusion Matrix:
[[ 37  17   0]
 [  8 219  10]
 [  0  21  88]]
```

In [15]:

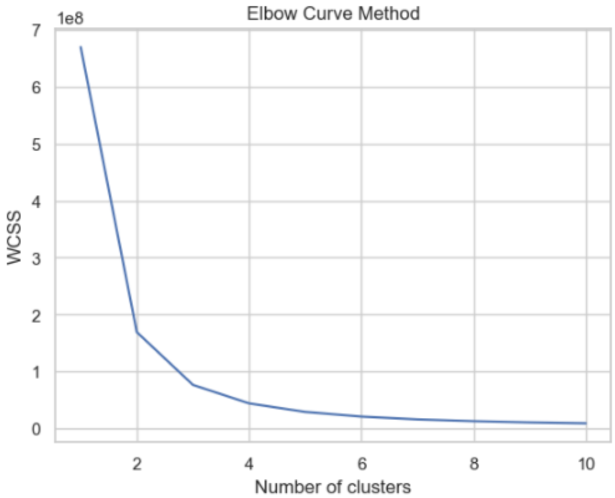
```
y_pred_prob = model.predict(X_test)

# Convert predicted probabilities and true labels to binary for each class (Class 0, Class 1, Class 2)
y_test_class_0 = (np.argmax(y_test, axis=1) == 0).astype(int)
y_test_class_1 = (np.argmax(y_test, axis=1) == 1).astype(int)
y_test_class_2 = (np.argmax(y_test, axis=1) == 2).astype(int)

y_pred_prob_class_0 = y_pred_prob[:, 0]
y_pred_prob_class_1 = y_pred_prob[:, 1]
y_pred_prob_class_2 = y_pred_prob[:, 2]

# Calculate AUC for each class
auc_class_0 = roc_auc_score(y_test_class_0, y_pred_prob_class_0)
auc_class_1 = roc_auc_score(y_test_class_1, y_pred_prob_class_1)
auc_class_2 = roc_auc_score(y_test_class_2, y_pred_prob_class_2)

print(f"AUC for Class 0 (Under Performer): {auc_class_0:.4f}")
print(f"AUC for Class 1 (Average Performer): {auc_class_1:.4f}")
print(f"AUC for Class 2 (High Performer): {auc_class_2:.4f}")

# Plot ROC curves
plt.figure(0).clf()
plt.plot([0, 1], ls="--")

# ROC for Class 0 (Under Performer)
fpr_0, tpr_0, _ = roc_curve(y_test_class_0, y_pred_prob_class_0)
plt.plot(fpr_0, tpr_0, label="Class 0 (Under Performer), AUC=" + str(round(auc_class_0, 3)))

# ROC for Class 1 (Average Performer)
fpr_1, tpr_1, _ = roc_curve(y_test_class_1, y_pred_prob_class_1)
plt.plot(fpr_1, tpr_1, label="Class 1 (Average Performer), AUC=" + str(round(auc_class_1, 3)))

# ROC for Class 2 (High Performer)
fpr_2, tpr_2, _ = roc_curve(y_test_class_2, y_pred_prob_class_2)
plt.plot(fpr_2, tpr_2, label="Class 2 (High Performer), AUC=" + str(round(auc_class_2, 3)))

# Add labels and legend
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.title('ROC Curve for Each Class')
plt.legend(loc="lower right")
plt.show()
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
AUC for Class 0 (Under Performer): 0.9808
AUC for Class 1 (Average Performer): 0.9484
AUC for Class 2 (High Performer): 0.9789
```



In [17]:

```
import numpy as np
import os
from matplotlib import pyplot as plt
from sklearn.cluster import KMeans

# identify optimal k using the Elbow Curve method
wcss = []
for i in range(1, 11):   # specified a range for the Elbow Curve x-axis
    model_elbow = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=10)
    model_elbow.fit(ss_cleaned)
    wcss.append(model_elbow.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('Elbow Curve Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()

###  K = 3 is the ideal number of clusters .
```



```
In [19]: import numpy as np
         import pandas as pd
         from sklearn import metrics
         from sklearn.model_selection import train_test_split
         from sklearn.preprocessing import StandardScaler
         from sklearn.cluster import KMeans
         from tensorflow.keras.utils import to_categorical
         from tensorflow.keras.models import Sequential
         from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense
         from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, roc_curve, auc
         from sklearn.metrics import roc_curve, roc_auc_score
         import matplotlib.pyplot as plt
```

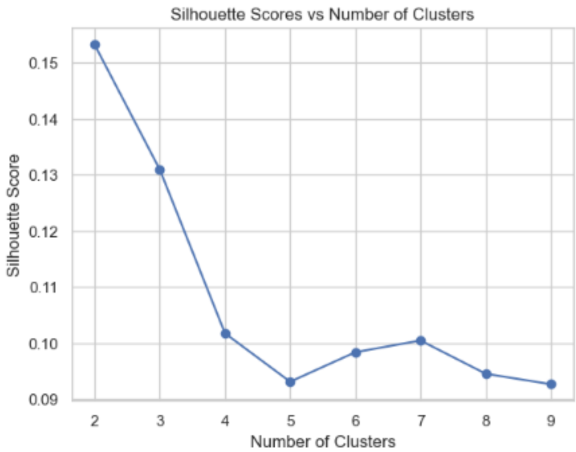
```
In [20]: ##### Model Building (K-means Clustering + CNN) #####

         # Drop features irrelevant for classification
         X = ss_cleaned.drop(columns=['id', 'performance', 'average_score']).values
         y = ss_cleaned['performance'].values

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

         ##### K-Means clustering - 3 clusters #####
         kmeans = KMeans(n_clusters=3, random_state=10)
         kmeans.fit(X_scaled)

         ##### Add cluster labels as a new feature #####
         cluster_labels = kmeans.labels_
         X_with_clusters = np.column_stack((X_scaled, cluster_labels))

         # Convert target label 'performance' to categorical (for 3 categories classification)
         y_categorical = to_categorical(y, num_classes=3)

         # Reshape data to match 3D format of CNN (samples, time steps, features)
         X_with_clusters = X_with_clusters.reshape(X_with_clusters.shape[0], X_with_clusters.shape[1], 1)

         # Split the data into training and testing sets
         X_train, X_test, y_train, y_test = train_test_split(X_with_clusters, y_categorical, test_size=0.2, random_state=10)

         # Define the CNN model for 3 categories classification
         model = Sequential([
             Conv1D(64, kernel_size=2, activation='relu', input_shape=(X_train.shape[1], 1)),
             MaxPooling1D(pool_size=2),
             Flatten(),
             Dense(100, activation='relu'),
             Dense(3, activation='softmax')
         ])

         # Model Compilation
         model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

         # Model training
         history = model.fit(X_train, y_train, epochs=20, validation_data=(X_test, y_test), verbose=1)

         # Evaluate model on the test set
         y_test_pred_probs = model.predict(X_test)
         y_test_pred = np.argmax(y_test_pred_probs, axis=1)
         y_test_true = np.argmax(y_test, axis=1)

         # Model Evaluation
         test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)
         print(f"Test Accuracy: {test_accuracy}")
```

```
Epoch 1/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.6764 - loss: 0.7435 - val_accuracy: 0.8925 - val_loss: 0.2938
Epoch 2/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8908 - loss: 0.2715 - val_accuracy: 0.9350 - val_loss: 0.1859
Epoch 3/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9231 - loss: 0.1913 - val_accuracy: 0.9350 - val_loss: 0.1617
Epoch 4/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9285 - loss: 0.1681 - val_accuracy: 0.9250 - val_loss: 0.1518
Epoch 5/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9539 - loss: 0.1454 - val_accuracy: 0.9400 - val_loss: 0.1453
Epoch 6/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9446 - loss: 0.1372 - val_accuracy: 0.9325 - val_loss: 0.1368
Epoch 7/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9497 - loss: 0.1258 - val_accuracy: 0.9400 - val_loss: 0.1279
Epoch 8/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9460 - loss: 0.1189 - val_accuracy: 0.9425 - val_loss: 0.1253
Epoch 9/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9571 - loss: 0.1091 - val_accuracy: 0.9425 - val_loss: 0.1201
Epoch 10/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9608 - loss: 0.1031 - val_accuracy: 0.9500 - val_loss: 0.1161
Epoch 11/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9644 - loss: 0.1000 - val_accuracy: 0.9450 - val_loss: 0.1164
Epoch 12/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9542 - loss: 0.1095 - val_accuracy: 0.9425 - val_loss: 0.1299
Epoch 13/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9679 - loss: 0.0867 - val_accuracy: 0.9475 - val_loss: 0.1139
Epoch 14/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9526 - loss: 0.1017 - val_accuracy: 0.9475 - val_loss: 0.1163
Epoch 15/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9567 - loss: 0.0964 - val_accuracy: 0.9475 - val_loss: 0.1114
Epoch 16/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9580 - loss: 0.0950 - val_accuracy: 0.9550 - val_loss: 0.0948
Epoch 17/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9576 - loss: 0.0922 - val_accuracy: 0.9625 - val_loss: 0.0894
Epoch 18/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9678 - loss: 0.0880 - val_accuracy: 0.9400 - val_loss: 0.1273
Epoch 19/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9694 - loss: 0.0761 - val_accuracy: 0.9625 - val_loss: 0.0955
Epoch 20/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9661 - loss: 0.0781 - val_accuracy: 0.9600 - val_loss: 0.0980
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
Test Accuracy: 0.9599999785423279
```

In [21]:
```python
from sklearn.metrics import silhouette_score

# Get the cluster labels
cluster_labels = kmeans.labels_

# Compute the silhouette score
sil_score = silhouette_score(X_scaled, cluster_labels)

# Print the silhouette score
print(f"Silhouette Score - K-Means with 3 clusters: {sil_score:.4f}")
```

```
Silhouette Score - K-Means with 3 clusters: 0.1301
```

In [25]:
```python
sil_scores = []
for n_clusters in range(2, 10):
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    cluster_labels = kmeans.fit_predict(X_scaled)
    sil_score = silhouette_score(X_scaled, cluster_labels)
    sil_scores.append(sil_score)

# Plot the silhouette scores for different numbers of clusters
plt.plot(range(2, 10), sil_scores, marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Scores vs Number of Clusters')
plt.show()

## Overall Low Silhouette Scores , from selection 2 to 9 clusters
```

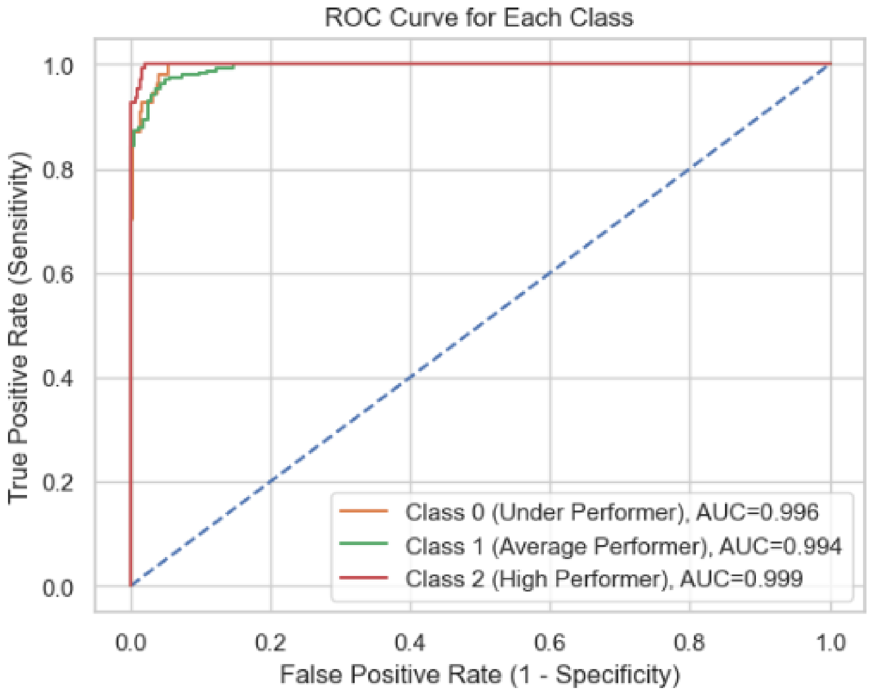


In [22]:
```python
#  Accuracy, Precision, Recall, F1-score
accuracy = accuracy_score(y_test_true, y_test_pred)
precision = precision_score(y_test_true, y_test_pred, average='macro')
recall = recall_score(y_test_true, y_test_pred, average='macro')
f1 = f1_score(y_test_true, y_test_pred, average='macro')

print(f'Accuracy: {accuracy:.4f}')
print(f'Precision: {precision:.4f}')
print(f'Recall: {recall:.4f}')
print(f'F1-Score: {f1:.4f}')

# Using metrics' function parameters to derive performance measures
acc = accuracy_score(y_test_true, y_test_pred)
prec = precision_score(y_test_true, y_test_pred, average='macro')
sens = recall_score(y_test_true, y_test_pred, average='macro')
f1 = f1_score(y_test_true, y_test_pred, average='macro')

# Show all performance metrics
print("Accuracy : ", round(acc, 3))
print("Precision : ", round(prec, 3))
print("Sensitivity/Recall) : ", round(sens, 3))
print("F1-Score : ", round(f1, 3))
print("Misclassification : ", round(1 - acc, 3))

print('----------------------------------------')

# Confusion Matrix
conf_matrix = confusion_matrix(y_test_true, y_test_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Accuracy: 0.9600
Precision: 0.9590
Recall: 0.9408
F1-Score: 0.9491
Accuracy :  0.96
Precision :  0.959
Sensitivity/Recall) :  0.941
F1-Score :  0.949
Misclassification :  0.04
----------------------------------------
Confusion Matrix:
[[ 47   7   0]
 [  2 230   5]
 [  0   2 107]]
```

```
In [23]: y_pred_prob = model.predict(X_test)

         # Convert predicted probabilities and true labels to binary for each class (Class 0, Class 1, Class 2)
         y_test_class_0 = (np.argmax(y_test, axis=1) == 0).astype(int)
         y_test_class_1 = (np.argmax(y_test, axis=1) == 1).astype(int)
         y_test_class_2 = (np.argmax(y_test, axis=1) == 2).astype(int)

         y_pred_prob_class_0 = y_pred_prob[:, 0]
         y_pred_prob_class_1 = y_pred_prob[:, 1]
         y_pred_prob_class_2 = y_pred_prob[:, 2]

         # Calculate AUC for each class
         auc_class_0 = roc_auc_score(y_test_class_0, y_pred_prob_class_0)
         auc_class_1 = roc_auc_score(y_test_class_1, y_pred_prob_class_1)
         auc_class_2 = roc_auc_score(y_test_class_2, y_pred_prob_class_2)

         print(f"AUC for Class 0 (Under Performer): {auc_class_0:.4f}")
         print(f"AUC for Class 1 (Average Performer): {auc_class_1:.4f}")
         print(f"AUC for Class 2 (High Performer): {auc_class_2:.4f}")

         # Plot ROC curves
         plt.figure(0).clf()
         plt.plot([0, 1], ls="--")

         # ROC for Class 0 (Under Performer)
         fpr_0, tpr_0, _ = roc_curve(y_test_class_0, y_pred_prob_class_0)
         plt.plot(fpr_0, tpr_0, label="Class 0 (Under Performer), AUC=" + str(round(auc_class_0, 3)))

         # ROC for Class 1 (Average Performer)
         fpr_1, tpr_1, _ = roc_curve(y_test_class_1, y_pred_prob_class_1)
         plt.plot(fpr_1, tpr_1, label="Class 1 (Average Performer), AUC=" + str(round(auc_class_1, 3)))

         # ROC for Class 2 (High Performer)
         fpr_2, tpr_2, _ = roc_curve(y_test_class_2, y_pred_prob_class_2)
         plt.plot(fpr_2, tpr_2, label="Class 2 (High Performer), AUC=" + str(round(auc_class_2, 3)))

         # Add labels and legend
         plt.xlabel('False Positive Rate (1 - Specificity)')
         plt.ylabel('True Positive Rate (Sensitivity)')
         plt.title('ROC Curve for Each Class')
         plt.legend(loc="lower right")
         plt.show()

         13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
         AUC for Class 0 (Under Performer): 0.9956
         AUC for Class 1 (Average Performer): 0.9938
         AUC for Class 2 (High Performer): 0.9990
```


```
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
AUC for Class 0 (Under Performer): 0.9956
AUC for Class 1 (Average Performer): 0.9938
AUC for Class 2 (High Performer): 0.9990
```

6. Model Evaluation results

The summary of the performance metrics for CNN model only vs. K-means clustering with CNN model is shown in the table below:

Performance Metric	**CNN model only**	**K-means Clustering with CNN**
Accuracy	86.00 %	96.00 %
Precision	85.74 %	95.90 %
Recall	80.55 %	94.84 %
F1-score	82.81 %	94.91 %
AUC (High Performer – 2)	0.9808	0.996
AUC (Average Performer – 1)	0.9484	0.994
AUC (Under Performer – 0)	0.9790	0.999
Silhouette Score		0.1301

In terms of accuracy, K-means clustering with CNN (96.00%) show 10% greater accuracy than CNN model only (86.00%), which proves that the added cluster labels by K-means clustering prior to CNN provided the CNN model a better performance in differentiating between the three performance categories of high performer, average performer and underperformer.

For precision, K-means clustering with CNN (85.74%) performed 10% greater than CNN model only (95.90%), indicating that false-positives are less likely to occur in the hybrid model than CNN alone. The influence of K-means clustering further improved the distinction between three performance categories respectively.

The recall scores in K-means clustering with CNN (80.55%) showed 14.29% greater than CNN alone (94.84%), indicating less false-negatives in the hybrid model, an important factor in ensuring underperforming students at-risk are selected correctly that lead to early intervention.

Since K-means clustering with CNN model performed better in accuracy, precision, and recall, naturally the F1-score in K-means clustering with CNN (82.81%) also showed 12.10% greater score than CNN alone (94.91%), showing powerful balance between precision and recall.

The ROC Curve/AUC is relatively similar performance between the two models, with K-means clustering with CNN having a slightly better score.

Lastly, the silhouette score is at 0.1301, which show that the clustering separation is not distinct, and clusters are poorly defined. Low silhouette scores indicate that clusters overlap each other. However, despite poor silhouette score, the K-means clustering still contributed to the significant improvement of the CNN performance.

As a conclusion, the goal of incorporating K-means clustering together with CNN machine learning model as a means to improve the classification of underperformers and high performers was proven useful when utilized upon this dataset of 2,000 students. The K-means clustering with CNN model has overall improved performance as compared to CNN only model, as evidenced by the performance metrics of accuracy, precision, recall, F1-scores, and ROC Curve/AUC. Although poor performance shown by the Silhouette Score of the K-means clustering, the clustering procedure has improved the ability of the CNN model to recognize and classify between underperformers, average performers, and high performers, likely due to the help of the cluster labels done by the K-means clustering model prior to performing CNN. As future improvements to this project, two suggestions can be done within this dataset to further optimize the performance of this hybrid model, which are experimenting with different unsupervised learning models to combine with CNN – such as DBSCAN which is a suitable model that analyses outliers efficiently. Secondly, proper feature selection and tuning that focuses on lesser, more targeted attributes of students are advised in order to properly train the hybrid model to identify between clusters more distinctly, which may contribute to improved clustering separation and Silhouette Score.

References

1. Ahmad Saeed Mohammad, Musab T. S. Al-Kaltakchi, Jabir Alshehabi Al-Ani, & Chambers, J. A. (2023). Comprehensive Evaluations of Student Performance Estimation via Machine Learning. *Mathematics*, *11*(14), 3153–3153. https://doi.org/10.3390/math11143153 .

2. Khan, I., Ahmad, A. R., Jabeur, N., & Mahdi, M. N. (2021). An artificial intelligence approach to monitor student performance and devise preventive measures. *Smart Learning Environments*, *8*(1). https://doi.org/10.1186/s40561-021-00161-y .

3. Al-Doulat, A., Nur, N., Karduni, A., Benedict, A., Al-Hossami, E., Maher, M. L., Dou, W., Dorodchi, M., & Niu, X. (2020). Making Sense of Student Success and Risk Through Unsupervised Machine Learning and Interactive Storytelling. *Lecture Notes in Computer Science*, 3–15. https://doi.org/10.1007/978-3-030-52237-7_1 .

4. Mohamed Nafuri, A. F., Sani, N. S., Zainudin, N. F. A., Rahman, A. H. A., & Aliff, M. (2022). Clustering Analysis for Classifying Student Academic Performance in Higher Education. *Applied Sciences*, *12*(19), 9467. https://doi.org/10.3390/app12199467 .

5. Peach, R. L., Yaliraki, S. N., Lefevre, D., & Barahona, M. (2019). Data-driven unsupervised clustering of online learner behaviour. *ArXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1902.04047 .

6. Medhat, M. (n.d.). *Student scores* [Data set]. Kaggle. https://www.kaggle.com/datasets/markmedhat/student-scores .