## Advanced Student Performance Analyzer using NumPy and Linear Algebra

**Advanced Student Performance Analyzer using NumPy and Linear Algebra**

**Objective**:

This project aims to build a data-driven student performance analysis tool using NumPy and basic linear algebra techniques. The tool will:

* Evaluate and rank student performance,
* Apply subject-specific weights,
* Normalize scores,
* Predict performance under different grading schemes,
* Perform matrix algebra operations to simulate scoring policy changes.

Dataset:

We have marks (out of 100) of 6 students across 5 subjects:

* Subjects: Math, Physics, Chemistry, English, Computer Science
* Each subject has a specific weight representing its importance in the final score.

|  |
| --- |
| import numpy as np  *# Student Scores (6 students × 5 subjects)* scores = np.array([  [85, 78, 92, 88, 91], *# Student 1* [76, 83, 80, 75, 77], *# Student 2* [90, 88, 94, 91, 89], *# Student 3* [65, 70, 72, 68, 66], *# Student 4* [80, 79, 85, 87, 82], *# Student 5* [72, 69, 74, 70, 76] *# Student 6* ]) |

**Requirements & Tasks**

*Section 1: Basic NumPy Operations*

Task 1: Print basic attributes of the score matrix

|  |
| --- |
| print("Shape:", scores.shape) print("Data Type:", scores.dtype) print("Dimensions:", scores.ndim) |

Task 2: Retrieve Student 3's scores

|  |
| --- |
| print("Student 3 scores:", scores[2]) |

Task 3: Retrieve all Computer Science scores (5th subject)

|  |
| --- |
| print("Computer Science scores:", scores[:, 4]) |

**Section 2: Statistical Analysis**

Task 4: Calculate average score for each student

|  |
| --- |
| student\_avg = np.mean(scores, axis=1) print("Student-wise averages:", student\_avg) |

Task 5: Find highest and lowest marks per subject

|  |
| --- |
| subject\_max = np.max(scores, axis=0) subject\_min = np.min(scores, axis=0) print("Max per subject:", subject\_max) print("Min per subject:", subject\_min) |

Task 6: Rank students based on average scores

|  |
| --- |
| ranking = np.argsort(-student\_avg) *# descending order* print("Ranking (Best to Worst):", ranking + 1) |

Task 7: Find standard deviation per subject

|  |
| --- |
| std\_subject = np.std(scores, axis=0) print("Standard Deviation per Subject:", std\_subject) |

**Section 3: Weighted Scores and Performance Index**

Task 8: Compute weighted score using subject importance

|  |
| --- |
| weights = np.array([0.3, 0.25, 0.2, 0.15, 0.1]) weighted\_scores = np.dot(scores, weights) print("Weighted scores:", weighted\_scores) |

Task 9: Normalize scores (Z-score normalization)

|  |
| --- |
| mean = np.mean(scores, axis=0) std = np.std(scores, axis=0) normalized\_scores = (scores - mean) / std print("Normalized Scores:\n", normalized\_scores) |

**Section 4: Advanced Linear Algebra Applications**

**Task 10: Solve a system of equations – Curved grading example**

*Assume a new grading curve:*

A × x = B  
Where x = new scoring weights to match target average grades

|  |
| --- |
| A = np.array([  [85, 78, 92],  [76, 83, 80],  [90, 88, 94] ])  B = np.array([255, 239, 270]) *# Target total score* x = np.linalg.solve(A, B) print("New Weight Vector (x):", x) |

Task 11: Compute inverse of a 3x3 subject correlation matrix

|  |
| --- |
| B = np.array([  [1, 0.85, 0.78],  [0.85, 1, 0.82],  [0.78, 0.82, 1] ])  B\_inv = np.linalg.inv(B) print("Inverse of correlation matrix:\n", B\_inv) |

Task 12: Covariance matrix of the subject scores

|  |
| --- |
| cov\_matrix = np.cov(scores.T) print("Covariance Matrix:\n", cov\_matrix) |

\* \* \*