Practical 3

Name: Divya Ganesh Bharsakle

Roll no: 342

Batch: C2

PRN: 202201030041

Problem Statement:

1. Read this dataset into an array
2. Perform all matrix operations on it
3. Horizontal and vertical stacking of numpy arrays
4. Custom sequence generations
5. Arithmetic and statistical operations
6. Mathematical operations
7. Bitwise operations
8. Copying and viewing arrays
9. Data stacking
10. Data Searching
11. Data sorting
12. Data counting
13. Data broadcasting

File: /content/12th_Result.csv

1 to 9 of 9 entries Filter

Roll no	Computer	English	History	Geography	Chemistry	Physics
540	70	80	77	31	44	58
320	60	70	64	41	55	57
460	55	60	61	51	66	56
850	64	50	67	61	77	78
650	78	40	34	71	33	64
314	71	59	80	80	45	36
750	80	67	49	45	56	34
560	67	74	65	64	67	38
468	50	85	34	48	78	37

Show 10 per page

Code:

```
import numpy as np

# Read the dataset into an array
data = np.genfromtxt('/content/12th_Result.csv', delimiter=',',
skip_header=1)
print("Dataset:")
print(data)
print()

transposed_data = data.T
print("Transposed Matrix:")
print(transposed_data)
print()

row_sums = np.sum(data, axis=1)
print("Row Sums:")
print(row_sums)
print()

column_avgs = np.mean(data, axis=0)
print("Column Averages:")
print(column_avgs)
```

```
print()

scaled_data = 2 * data
print("Scaled Matrix:")
print(scaled_data)
print()

elementwise_scaled_data = data * 2
print("Element-wise Scaled Matrix:")
print(elementwise_scaled_data)
print()

matrix_product = np.dot(data, data.T)
print("Matrix Product:")
print(matrix_product)
print()

determinant = np.linalg.det(data)
print("Determinant:")
print(determinant)
print()

print("Horizontal stacking:")
stacked_horizontal = np.hstack((data, data))
print(stacked_horizontal)

print("Vertical stacking:")
stacked_vertical = np.vstack((data, data))
print(stacked_vertical)

custom_sequence = np.arange(0, 10, 2)
print("Custom sequence:", custom_sequence)

print("Sum of each row:", np.sum(data, axis=1))
print("Mean of each column:", np.mean(data, axis=0))

print("Square root of each element:", np.sqrt(data))
print("Exponential of each element:", np.exp(data))

copy_data = np.copy(data)
view_data = data.view()
print("Copied array:\n", copy_data)
print("Viewed array:\n", view_data)

data_stack = np.stack((data, data))
print("Data stacking:\n", data_stack)

indices = np.where(data > 70)
```

```
print("Indices where data > 70:\n", indices)

sorted_data = np.sort(data, axis=0)
print("Sorted data:\n", sorted_data)

unique_elements, counts = np.unique(data, return_counts=True)
print("Unique elements:", unique_elements)
print("Counts:", counts)

broadcasted_data = data + 10
print("Broadcasted data:\n", broadcasted_data)
```

Output:

```
[650. 78. 40. 34. 71. 33. 64. 45. 78.]
 [314. 71. 59. 80. 80. 45. 36. 26. 77.]
 [750. 80. 67. 49. 45. 56. 34. 34. 76.]
 [560. 67. 74. 65. 64. 67. 38. 75. 75.]
 [468. 50. 85. 34. 48. 78. 37. 78. 70.]]
Data stacking:
 [[[540. 70. 80. 77. 31. 44. 58. 64. 74.]
  [320. 60. 70. 64. 41. 55. 57. 45. 80.]
  [460. 55. 60. 61. 51. 66. 56. 64. 73.]
  [850. 64. 50. 67. 61. 77. 78. 80. 72.]
  [650. 78. 40. 34. 71. 33. 64. 45. 78.]
  [314. 71. 59. 80. 80. 45. 36. 26. 77.]
  [750. 80. 67. 49. 45. 56. 34. 34. 76.]
  [560. 67. 74. 65. 64. 67. 38. 75. 75.]
  [468. 50. 85. 34. 48. 78. 37. 78. 70.]]

 [[540. 70. 80. 77. 31. 44. 58. 64. 74.]
  [320. 60. 70. 64. 41. 55. 57. 45. 80.]
  [460. 55. 60. 61. 51. 66. 56. 64. 73.]
  [850. 64. 50. 67. 61. 77. 78. 80. 72.]
  [650. 78. 40. 34. 71. 33. 64. 45. 78.]
  [314. 71. 59. 80. 80. 45. 36. 26. 77.]
  [750. 80. 67. 49. 45. 56. 34. 34. 76.]
  [560. 67. 74. 65. 64. 67. 38. 75. 75.]
  [468. 50. 85. 34. 48. 78. 37. 78. 70.]]]
Indices where data > 70:
 (array([0, 0, 0, 0, 1, 1, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5,
       6, 6, 6, 7, 7, 7, 7, 8, 8, 8, 8]), array([0, 2, 3, 8, 0, 8, 0, 8, 0, 5, 6, 7, 8, 0, 1, 4, 8, 0, 1, 3, 4, 8,
       0, 1, 8, 0, 2, 7, 8, 0, 2, 5, 7]))
Sorted data:
 [[314. 50. 40. 34. 31. 33. 34. 26. 70.]
 [320. 55. 50. 34. 41. 44. 36. 34. 72.]
 [460. 60. 59. 49. 45. 45. 37. 45. 73.]
 [468. 64. 60. 61. 48. 55. 38. 45. 74.]
 [540. 67. 67. 64. 51. 56. 56. 64. 75.]
 [560. 70. 70. 65. 61. 66. 57. 64. 76.]
```

```
 [650. 71. 74. 67. 64. 67. 58. 75. 77.]
 [750. 78. 80. 77. 71. 77. 64. 78. 78.]
 [850. 80. 85. 80. 80. 78. 78. 80. 80.]]
Unique elements: [ 26. 31. 33. 34. 36. 37. 38. 40. 41. 44. 45. 48. 49. 50.
 51. 55. 56. 57. 58. 59. 60. 61. 64. 65. 66. 67. 70. 71.
 72. 73. 74. 75. 76. 77. 78. 80. 85. 314. 320. 460. 468. 540.
 560. 650. 750. 850.]
Counts: [1 1 1 4 1 1 1 1 1 1 4 1 1 2 1 2 2 1 1 1 2 2 6 1 1 4 3 2 1 1 2 2 1 3 5 6 1
 1 1 1 1 1 1 1 1 1]
Broadcasted data:
 [[550. 80. 90. 87. 41. 54. 68. 74. 84.]
 [330. 70. 80. 74. 51. 65. 67. 55. 90.]
 [470. 65. 70. 71. 61. 76. 66. 74. 83.]
 [860. 74. 60. 77. 71. 87. 88. 90. 82.]
 [660. 88. 50. 44. 81. 43. 74. 55. 88.]
 [324. 81. 69. 90. 90. 55. 46. 36. 87.]
 [760. 90. 77. 59. 55. 66. 44. 44. 86.]
 [570. 77. 84. 75. 74. 77. 48. 85. 85.]
 [478. 60. 95. 44. 58. 88. 47. 88. 80.]]
<ipython-input-47-8c34614cd5b4>:59: RuntimeWarning: overflow encountered in exp
  print("Exponential of each element:", np.exp(data))
```