EDS PRACTICAL 3

Name – Mayur Kapgate **Div. – D2**

PRN No. – 202201040065 **Roll No. – 428**

CSV FILE :

RollNo	EDS	SON	DT	ET
428	65	63	64	45
429	73	64	65	66
430	24	86	34	98
431	74	35	56	48
432	88	65	45	56
433	45	45	77	77
434	66	43	66	65
435	75	75	45	87
436	84	86	89	67
437	64	64	76	34

CODE :

```
import numpy as np
import csv

# Read the CSV file and convert it into an array
def read_csv_to_array(file_path):
    with open(file_path, mode='r') as file:
        reader = csv.reader(file)
        next(reader)  # Skip the header row
        data_list = list(reader)
    return np.array(data_list, dtype=np.float64)  # Convert to float
data type

# Perform all matrix operations
def perform_matrix_operations(dataset):
    print("Matrix Operations:")
    print("Transpose:")
    print(np.transpose(dataset))
    print("Matrix multiplication:")
```

```
    print(np.matmul(dataset, np.transpose(dataset)))
    print("Inverse:")

# Horizontal and vertical stacking of arrays
def perform_stacking(dataset):
    a = np.array([1, 2, 3])
    b = np.array([4, 5, 6])
    print("Horizontal stacking:")
    print(np.hstack((a, b)))
    print("Vertical stacking:")
    print(np.vstack((a, b)))
    print()

# Custom sequence generation
def generate_sequences():
    print("Custom sequence generation:")
    print(np.arange(0, 10, 2))  # Generates sequence [0, 2, 4, 6, 8]
    print(np.linspace(0, 1, 5))  # Generates sequence [0.0, 0.25, 0.5,
0.75, 1.0]
    print()

# Arithmetic and statistical operations, mathematical operations,
bitwise operators
def perform_operations(dataset):
    print("Arithmetic and Statistical Operations:")
    print("Sum:")
    print(np.sum(dataset))
    print("Mean:")
    print(np.mean(dataset))
    print("Standard Deviation:")
    print(np.std(dataset))
    print()

    print("Mathematical and Bitwise Operators:")
    print("Square root:")
    print(np.sqrt(dataset))
    print("Element-wise addition:")
    print(dataset + 1)
    print("Bitwise OR:")
    print(np.bitwise_or(dataset, 2))
    print()

# Copying and viewing arrays
def copy_and_view_arrays(dataset):
    print("Copying and Viewing Arrays:")
    copy = np.copy(dataset)
    view = dataset.view()
    print("Original array:")
```

```
    print(dataset)
    print("Copy:")
    print(copy)
    print("View:")
    print(view)
    print()

# Data stacking, searching, sorting, counting, broadcasting
def perform_data_operations():
    arr1 = np.array([1, 2, 3])
    arr2 = np.array([4, 5, 6])
    print("Data Stacking:")
    print(np.column_stack((arr1, arr2)))
    print("Searching:")
    print(np.where(arr1 == 2))
    print("Sorting:")
    print(np.sort(arr1))
    print("Counting:")
    print(np.count_nonzero(arr1))
    print("Broadcasting:")
    print(arr1 + 1)


# Specify the file path of the CSV dataset
file_path = '/content/EDS_PRACTICAL_3.csv'

# Read the CSV file and convert it into an array
dataset = read_csv_to_array(file_path)

# Convert the dataset to integer data type
dataset = dataset.astype(np.int64)

# Perform all matrix operations
perform_matrix_operations(dataset)

# Horizontal and vertical stacking of arrays
perform_stacking(dataset)

# Custom sequence generation
generate_sequences()

# Arithmetic and statistical operations, mathematical operations,
bitwise operators
perform_operations(dataset)

# Copying and viewing arrays
copy_and_view_arrays(dataset)

# Data stacking, searching, sorting, counting, broadcasting
```

```
perform_data_operations()
```

OUTPUT :

```
Matrix Operations:
Transpose:
[[428 429 430 431 432 433 434 435 436 437]
 [ 65  73  24  74  88  45  66  75  84  64]
 [ 63  64  86  35  65  45  43  75  86  64]
 [ 64  65  34  56  45  77  66  45  89  76]
 [ 45  66  98  48  56  77  65  87  67  34]]
Matrix multiplication:
[[197499 199519 197604 197227 200111 199477 199900 202575 206197
201622]
 [199519 202047 200404 199349 202533 202009 202336 205557 208887
203425]
 [197604 200404 203632 196724 200480 201304 200516 205356 206484
200866]
 [197227 199349 196724 197902 200187 199536 200259 202356 205342
201211]
 [200111 202533 200480 200187 203754 201718 202701 206292 209091
203900]
 [199477 202009 201304 199536 201718 203397 202914 205269 208450
203451]
 [199900 202336 200516 200259 202701 202914 203142 205590 208695
203860]
 [202575 205557 205356 202356 206292 205269 205590 210069 212244
206073]
 [206197 208887 206484 205342 209091 208450 208695 212244 216958
210454]
 [201622 203425 200866 201211 203900 203451 203860 206073 210454
206093]]
Inverse:
Horizontal stacking:
[1 2 3 4 5 6]
Vertical stacking:
[[1 2 3]
 [4 5 6]]

Custom sequence generation:
[0 2 4 6 8]
[0.   0.25 0.5  0.75 1.  ]

Arithmetic and Statistical Operations:
Sum:
6869
Mean:
137.38
Standard Deviation:
148.37990295184855

Mathematical and Bitwise Operators:
Square root:
```

```
[[20.68816087  8.06225775  7.93725393  8.          6.70820393]
 [20.71231518  8.54400375  8.          8.06225775  8.1240384 ]
 [20.73644135  4.89897949  9.2736185   5.83095189  9.89949494]
 [20.76053949  8.60232527  5.91607978  7.48331477  6.92820323]
 [20.78460969  9.38083152  8.06225775  6.70820393  7.48331477]
 [20.80865205  6.70820393  6.70820393  8.77496439  8.77496439]
 [20.83266666  8.1240384   6.55743852  8.1240384   8.06225775]
 [20.85665361  8.66025404  8.66025404  6.70820393  9.32737905]
 [20.88061302  9.16515139  9.2736185   9.43398113  8.18535277]
 [20.90454496  8.          8.          8.71779789  5.83095189]]
Element-wise addition:
[[429  66  64  65  46]
 [430  74  65  66  67]
 [431  25  87  35  99]
 [432  75  36  57  49]
 [433  89  66  46  57]
 [434  46  46  78  78]
 [435  67  44  67  66]
 [436  76  76  46  88]
 [437  85  87  90  68]
 [438  65  65  77  35]]
Bitwise OR:
[[430  67  63  66  47]
 [431  75  66  67  66]
 [430  26  86  34  98]
 [431  74  35  58  50]
 [434  90  67  47  58]
 [435  47  47  79  79]
 [434  66  43  66  67]
 [435  75  75  47  87]
 [438  86  86  91  67]
 [439  66  66  78  34]]

Copying and Viewing Arrays:
Original array:
[[428  65  63  64  45]
 [429  73  64  65  66]
 [430  24  86  34  98]
 [431  74  35  56  48]
 [432  88  65  45  56]
 [433  45  45  77  77]
 [434  66  43  66  65]
 [435  75  75  45  87]
 [436  84  86  89  67]
 [437  64  64  76  34]]
Copy:
[[428  65  63  64  45]
 [429  73  64  65  66]
 [430  24  86  34  98]
 [431  74  35  56  48]
 [432  88  65  45  56]
 [433  45  45  77  77]
 [434  66  43  66  65]
 [435  75  75  45  87]
 [436  84  86  89  67]
 [437  64  64  76  34]]
View:
[[428  65  63  64  45]
```

```
 [429  73  64  65  66]
 [430  24  86  34  98]
 [431  74  35  56  48]
 [432  88  65  45  56]
 [433  45  45  77  77]
 [434  66  43  66  65]
 [435  75  75  45  87]
 [436  84  86  89  67]
 [437  64  64  76  34]]

Data Stacking:
[[1 4]
 [2 5]
 [3 6]]
Searching:
(array([1]),)
Sorting:
[1 2 3]
Counting:
3
Broadcasting:
[2 3 4]
```