Name:Mansi kawade
Rollno: 631
Div:F

TESTMARKS1

```
import numpy as np
t= np.genfromtxt("/content/testmarks1 (1).csv",delimiter=',', dtype=
str,skip_header=1)
print(t)
rollnp =t[:,0]
eds=t[:,1]
son =t[:,2]
dt =t[:,3]
et =t[:,4]
e= eds.astype(float)
d= son.astype(float)
g= dt.astype(float)
h= et.astype(float)
sum =e+d+g+h

print(f"roll number:{rollnp}")
print(f"EDS marks:{e}")
print(f"SON marks:{d}")
print(f"DT marks:{g}")
print(f"ET marks:{h}")
print(f"total marks:{sum}")
print(f"highest marks:{sum.max()}")
print(f"lowest marks:{sum.min()}")
print(f"topper marks:{rollnp}")
```

Output

```
[['801' '43.05' '27.79' '28.7' '27.79']
 ['802' '43.47' '28.52' '28.98' '27.89']
 ['803' '42.24' '28.16' '28.16' '25.63']
 ['804' '39.24' '26.16' '26.16' '26.16']
 ['805' '40.9' '26.03' '27.27' '25.65']
 ['806' '39.47' '26.31' '26.31' '25.21']
 ['807' '41.68' '25.63' '27.79' '25.46']
 ['808' '42.19' '27.61' '28.13' '26.21']
 ['809' '44.75' '28.35' '29.83' '28.21']
 ['810' '46.95' '28.88' '31.3' '28.53']]
roll number:['801' '802' '803' '804' '805' '806' '807' '808' '809' '810']
EDS marks:[43.05 43.47 42.24 39.24 40.9  39.47 41.68 42.19 44.75 46.95]
SON marks:[27.79 28.52 28.16 26.16 26.03 26.31 25.63 27.61 28.35 28.88]
DT marks:[28.7  28.98 28.16 26.16 27.27 26.31 27.79 28.13 29.83 31.3 ]
ET marks:[27.79 27.89 25.63 26.16 25.65 25.21 25.46 26.21 28.21 28.53]
total marks:[127.33 128.86 124.19 117.72 119.85 117.3  120.56 124.14 131.14 135.66]
```

```
highest marks:135.66
lowest marks:117.30000000000001
topper marks:['801' '802' '803' '804' '805' '806' '807' '808' '809' '810']
```

```python
import numpy as np

# Read the dataset from the Excel sheet
# Assuming the dataset is stored in a CSV file named 'data.csv'
data = np.genfromtxt('/content/testmarks1.csv', delimiter=',')

# Perform all matrix operations
# Example: Matrix multiplication
matrix_product = np.dot(data, data.T)
print("Matrix product:")
print(matrix_product)

# Horizontal and vertical stacking of arrays
# Example: Horizontal stacking
stacked_data = np.hstack((data, data))
print("Horizontal stacking:")
print(stacked_data)

# Custom sequence generation
# Example: Generate a sequence of numbers from 0 to 9
sequence = np.arange(10)
print("Sequence:")
print(sequence)

# Arithmetic and Statistical Operations
# Example: Mean of each column
column_means = np.mean(data, axis=0)
print("Column means:")
print(column_means)

# Mathematical Operations
# Example: Square root of each element
sqrt_data = np.sqrt(data)
print("Square root:")
print(sqrt_data)


# Copying and viewing arrays
# Example: Create a view of the original array
data_view = data.view()
print("Original array:")
```

```
print(data)
print("View of the array:")
print(data_view)

# Data Stacking, Searching, Sorting, Counting, Broadcasting
# Example: Stack arrays vertically
stacked_data_v = np.vstack((data, data))
print("Vertical stacking:")
print(stacked_data_v)

# Example: Search for a value in the array
value_index = np.where(data == 27.79)
print("Indices of value 27.79:")
print(value_index)

# Example: Sort the array
sorted_data = np.sort(data)
print("Sorted array:")
print(sorted_data)

# Example: Count the occurrences of a value in the array
value_count = np.count_nonzero(data == 27.79)
print("Count of value 27.79:")
print(value_count)

# Example: Broadcast a scalar value to the array shape
broadcasted_data = np.broadcast_to(2, data.shape)
print("Broadcasted array:")
print(broadcasted_data)
```

Output

```
Matrix product:
[[   nan    nan    nan    nan    nan    nan
     nan    nan    nan    nan    nan]
 [   nan 645822.5607 646672.7434 647324.4481 647898.0468 648784.5812
  649492.0213 650418.6881 651327.2683 652363.4109 653324.9314]
 [   nan 646672.7434 647524.7238 648176.1935 648747.5652 649635.9617
  650343.6928 651272.2308 652183.6408 653223.0748 654187.3498]
 [   nan 647324.4481 648176.1935 648836.0857 649413.3096 650300.9535
  651013.1243 651938.4102 652847.5063 653878.6111 654839.0607]
 [   nan 647898.0468 648747.5652 649413.3096 650008.8144 650890.248
  651608.8356 652527.024  653431.3476 654451.9524 655402.9716]
 [   nan 648784.5812 649635.9617 650300.9535 650890.248  651776.9463
  652493.2825 653417.7432 654323.6509 655350.2761 656307.3469]
 [   nan 649492.0213 650343.6928 651013.1243 651608.8356 652493.2825
  653213.8572 654134.4364 655040.5128 656062.1724 657015.6936]
 [   nan 650418.6881 651272.2308 651938.4102 652527.024  653417.7432
  654134.4364 655063.615  655971.1628 657001.9928 657963.2712]
 [   nan 651327.2683 652183.6408 652847.5063 653431.3476 654323.6509
  655040.5128 655971.1628 656884.5692 657921.248  658886.4376]
```

```
[       nan 652363.4109 653223.0748 653878.6111 654451.9524 655350.2761
  656062.1724 657001.9928 657921.248  658972.918  659948.2708]
 [       nan 653324.9314 654187.3498 654839.0607 655402.9716 656307.3469
  657015.6936 657963.2712 658886.4376 659948.2708 660932.0078]]
Horizontal stacking:
[[  nan   nan   nan   nan   nan   nan   nan   nan   nan   nan]
 [801.   43.05  27.79  28.7   27.79 801.   43.05  27.79  28.7   27.79]
 [802.   43.47  28.52  28.98  27.89 802.   43.47  28.52  28.98  27.89]
 [803.   42.24  28.16  28.16  25.63 803.   42.24  28.16  28.16  25.63]
 [804.   39.24  26.16  26.16  26.16 804.   39.24  26.16  26.16  26.16]
 [805.   40.9   26.03  27.27  25.65 805.   40.9   26.03  27.27  25.65]
 [806.   39.47  26.31  26.31  25.21 806.   39.47  26.31  26.31  25.21]
 [807.   41.68  25.63  27.79  25.46 807.   41.68  25.63  27.79  25.46]
 [808.   42.19  27.61  28.13  26.21 808.   42.19  27.61  28.13  26.21]
 [809.   44.75  28.35  29.83  28.21 809.   44.75  28.35  29.83  28.21]
 [810.   46.95  28.88  31.3   28.53 810.   46.95  28.88  31.3   28.53]]
Sequence:
[0 1 2 3 4 5 6 7 8 9]
Column means:
[nan nan nan nan nan]
Square root:
[[       nan        nan        nan        nan        nan]
 [28.3019434   6.56124988  5.27162214  5.35723809  5.27162214]
 [28.31960452  6.59317829  5.34041197  5.38330753  5.28109837]
 [28.33725463  6.49923072  5.30659966  5.30659966  5.06260802]
 [28.35489376  6.26418391  5.11468474  5.11468474  5.11468474]
 [28.37252192  6.39531078  5.10196041  5.22206856  5.0645829 ]
 [28.39013913  6.28251542  5.12932744  5.12932744  5.02095608]
 [28.40774542  6.45600496  5.06260802  5.27162214  5.04579032]
 [28.42534081  6.49538298  5.25452186  5.30377224  5.11957029]
 [28.44292531  6.68954408  5.3244718   5.46168472  5.31130869]
 [28.46049894  6.85200701  5.37401154  5.59464029  5.34134814]]
Original array:
[[  nan   nan   nan   nan   nan]
 [801.   43.05  27.79  28.7   27.79]
 [802.   43.47  28.52  28.98  27.89]
 [803.   42.24  28.16  28.16  25.63]
 [804.   39.24  26.16  26.16  26.16]
 [805.   40.9   26.03  27.27  25.65]
 [806.   39.47  26.31  26.31  25.21]
 [807.   41.68  25.63  27.79  25.46]
 [808.   42.19  27.61  28.13  26.21]
 [809.   44.75  28.35  29.83  28.21]
 [810.   46.95  28.88  31.3   28.53]]
View of the array:
[[  nan   nan   nan   nan   nan]
 [801.   43.05  27.79  28.7   27.79]
 [802.   43.47  28.52  28.98  27.89]
 [803.   42.24  28.16  28.16  25.63]
 [804.   39.24  26.16  26.16  26.16]
 [805.   40.9   26.03  27.27  25.65]
 [806.   39.47  26.31  26.31  25.21]
 [807.   41.68  25.63  27.79  25.46]
 [808.   42.19  27.61  28.13  26.21]
 [809.   44.75  28.35  29.83  28.21]
 [810.   46.95  28.88  31.3   28.53]]
Vertical stacking:
[[  nan   nan   nan   nan   nan]
 [801.   43.05  27.79  28.7   27.79]
```

```
 [802.   43.47  28.52  28.98  27.89]
 [803.   42.24  28.16  28.16  25.63]
 [804.   39.24  26.16  26.16  26.16]
 [805.   40.9   26.03  27.27  25.65]
 [806.   39.47  26.31  26.31  25.21]
 [807.   41.68  25.63  27.79  25.46]
 [808.   42.19  27.61  28.13  26.21]
 [809.   44.75  28.35  29.83  28.21]
 [810.   46.95  28.88  31.3   28.53]
 [  nan   nan   nan   nan   nan]
 [801.   43.05  27.79  28.7   27.79]
 [802.   43.47  28.52  28.98  27.89]
 [803.   42.24  28.16  28.16  25.63]
 [804.   39.24  26.16  26.16  26.16]
 [805.   40.9   26.03  27.27  25.65]
 [806.   39.47  26.31  26.31  25.21]
 [807.   41.68  25.63  27.79  25.46]
 [808.   42.19  27.61  28.13  26.21]
 [809.   44.75  28.35  29.83  28.21]
 [810.   46.95  28.88  31.3   28.53]]
Indices of value 27.79:
(array([1, 1, 7]), array([2, 4, 3]))
Sorted array:
[[  nan   nan   nan   nan   nan]
 [ 27.79  27.79  28.7   43.05 801.  ]
 [ 27.89  28.52  28.98  43.47 802.  ]
 [ 25.63  28.16  28.16  42.24 803.  ]
 [ 26.16  26.16  26.16  39.24 804.  ]
 [ 25.65  26.03  27.27  40.9  805.  ]
 [ 25.21  26.31  26.31  39.47 806.  ]
 [ 25.46  25.63  27.79  41.68 807.  ]
 [ 26.21  27.61  28.13  42.19 808.  ]
 [ 28.21  28.35  29.83  44.75 809.  ]
 [ 28.53  28.88  31.3   46.95 810.  ]]
Count of value 27.79:
3
Broadcasted array:
[[2 2 2 2 2]
 [2 2 2 2 2]
 [2 2 2 2 2]
 [2 2 2 2 2]
 [2 2 2 2 2]
 [2 2 2 2 2]
 [2 2 2 2 2]
 [2 2 2 2 2]
 [2 2 2 2 2]
 [2 2 2 2 2]
 [2 2 2 2 2]]
```

TESTMARKS2

```
import numpy as np

# Read the dataset from the Excel sheet
# Assuming the dataset is stored in a CSV file named 'data.csv'
data = np.genfromtxt('/content/testmarks2.csv', delimiter=',')

# Perform all matrix operations
# Example: Matrix multiplication
matrix_product = np.dot(data, data.T)
print("Matrix product:")
print(matrix_product)

# Horizontal and vertical stacking of arrays
# Example: Horizontal stacking
stacked_data = np.hstack((data, data))
print("Horizontal stacking:")
print(stacked_data)

# Custom sequence generation
# Example: Generate a sequence of numbers from 0 to 9
sequence = np.arange(10)
print("Sequence:")
print(sequence)

# Arithmetic and Statistical Operations
# Example: Mean of each column
column_means = np.mean(data, axis=0)
print("Column means:")
print(column_means)

# Mathematical Operations
# Example: Square root of each element
sqrt_data = np.sqrt(data)
print("Square root:")
print(sqrt_data)


# Copying and viewing arrays
# Example: Create a view of the original array
data_view = data.view()
```

```
print("Original array:")
print(data)
print("View of the array:")
print(data_view)

# Data Stacking, Searching, Sorting, Counting, Broadcasting
# Example: Stack arrays vertically
stacked_data_v = np.vstack((data, data))
print("Vertical stacking:")
print(stacked_data_v)

# Example: Search for a value in the array
value_index = np.where(data == 27.79)
print("Indices of value 27.79:")
print(value_index)

# Example: Sort the array
sorted_data = np.sort(data)
print("Sorted array:")
print(sorted_data)

# Example: Count the occurrences of a value in the array
value_count = np.count_nonzero(data == 27.79)
print("Count of value 27.79:")
print(value_count)

# Example: Broadcast a scalar value to the array shape
broadcasted_data = np.broadcast_to(2, data.shape)
print("Broadcasted array:")
print(broadcasted_data)
```

Output

```
Matrix product:
[[     nan      nan      nan      nan      nan      nan
       nan      nan      nan      nan      nan]
 [     nan 645008.4693 645799.707  646383.5809 647166.7377 647944.0774
  648690.0435 649536.8699 650486.9218 651450.9186 652391.6894]
 [     nan 645799.707  646592.6632 647178.8774 647962.1598 648740.4224
  649488.0712 650334.9518 651285.8336 652250.343  653190.7978]
 [     nan 646383.5809 647178.8774 647781.5186 648564.8266 649345.7894
  650098.6651 650942.65   651885.9715 652840.2337 653772.5388]
 [     nan 647166.7377 647962.1598 648564.8266 649352.031  650133.845
  650887.0685 651733.0798 652675.3649 653630.9431 654563.7792]
 [     nan 647944.0774 648740.4224 649345.7894 650133.845  650916.9932
  651671.5594 652518.3712 653460.8398 654415.9492 655349.295 ]
 [     nan 648690.0435 649488.0712 650098.6651 650887.0685 651671.5594
  652428.4495 653274.8754 654216.2701 655169.5584 656101.0967]
 [     nan 649536.8699 650334.9518 650942.65   651733.0798 652518.3712
  653274.8754 654123.8979 655067.8922 656024.5244 656959.9146]
 [     nan 650486.9218 651285.8336 651885.9715 652675.3649 653460.8398
```

```
 654216.2701 655067.8922 656020.0338 656984.122  657926.848 ]
 [       nan 651450.9186 652250.343  652840.2337 653630.9431 654415.9492
  655169.5584 656024.5244 656984.122  657957.8427 658907.5769]
 [       nan 652391.6894 653190.7978 653772.5388 654563.7792 655349.295
  656101.0967 656959.9146 657926.848  658907.5769 659865.8533]]
Horizontal stacking:
[[  nan   nan   nan   nan   nan   nan   nan   nan   nan   nan]
 [801.   28.48  34.18  30.56  22.23 801.   28.48  34.18  30.56  22.23]
 [802.   28.1   33.72  30.68  22.82 802.   28.1   33.72  30.68  22.82]
 [803.   26.16  31.39  28.2   22.53 803.   26.16  31.39  28.2   22.53]
 [804.   26.16  31.39  28.78  20.93 804.   26.16  31.39  28.78  20.93]
 [805.   26.1   31.32  28.22  20.82 805.   26.1   31.32  28.22  20.82]
 [806.   25.45  30.54  27.73  21.05 806.   25.45  30.54  27.73  21.05]
 [807.   26.16  31.39  28.01  20.51 807.   26.16  31.39  28.01  20.51]
 [808.   27.44  32.93  28.83  22.08 808.   27.44  32.93  28.83  22.08]
 [809.   28.63  34.35  31.03  22.68 809.   28.63  34.35  31.03  22.68]
 [810.   30.35  36.42  31.38  23.1  810.   30.35  36.42  31.38  23.1 ]]
Sequence:
[0 1 2 3 4 5 6 7 8 9]
Column means:
[nan nan nan nan nan]
Square root:
[[       nan        nan        nan        nan        nan]
 [28.3019434   5.33666563  5.84636639  5.52810998  4.71487009]
 [28.31960452  5.30094331  5.80689246  5.53895297  4.77702836]
 [28.33725463  5.11468474  5.60267793  5.31036722  4.74657771]
 [28.35489376  5.11468474  5.60267793  5.36469943  4.57493169]
 [28.37252192  5.10881591  5.59642743  5.31224999  4.56289382]
 [28.39013913  5.0447993   5.52630075  5.26592822  4.5880279 ]
 [28.40774542  5.11468474  5.60267793  5.29244745  4.52879675]
 [28.42534081  5.23832034  5.73846669  5.3693575   4.69893605]
 [28.44292531  5.35070089  5.8608873   5.57045779  4.76235236]
 [28.46049894  5.50908341  6.03489851  5.60178543  4.80624594]]
Original array:
[[  nan   nan   nan   nan   nan]
 [801.   28.48  34.18  30.56  22.23]
 [802.   28.1   33.72  30.68  22.82]
 [803.   26.16  31.39  28.2   22.53]
 [804.   26.16  31.39  28.78  20.93]
 [805.   26.1   31.32  28.22  20.82]
 [806.   25.45  30.54  27.73  21.05]
 [807.   26.16  31.39  28.01  20.51]
 [808.   27.44  32.93  28.83  22.08]
 [809.   28.63  34.35  31.03  22.68]
 [810.   30.35  36.42  31.38  23.1 ]]
View of the array:
[[  nan   nan   nan   nan   nan]
 [801.   28.48  34.18  30.56  22.23]
 [802.   28.1   33.72  30.68  22.82]
 [803.   26.16  31.39  28.2   22.53]
 [804.   26.16  31.39  28.78  20.93]
 [805.   26.1   31.32  28.22  20.82]
 [806.   25.45  30.54  27.73  21.05]
 [807.   26.16  31.39  28.01  20.51]
 [808.   27.44  32.93  28.83  22.08]
 [809.   28.63  34.35  31.03  22.68]
 [810.   30.35  36.42  31.38  23.1 ]]
Vertical stacking:
[[  nan   nan   nan   nan   nan]
```

```
 [801.   28.48  34.18  30.56  22.23]
 [802.   28.1   33.72  30.68  22.82]
 [803.   26.16  31.39  28.2   22.53]
 [804.   26.16  31.39  28.78  20.93]
 [805.   26.1   31.32  28.22  20.82]
 [806.   25.45  30.54  27.73  21.05]
 [807.   26.16  31.39  28.01  20.51]
 [808.   27.44  32.93  28.83  22.08]
 [809.   28.63  34.35  31.03  22.68]
 [810.   30.35  36.42  31.38  23.1 ]
 [  nan   nan   nan   nan   nan]
 [801.   28.48  34.18  30.56  22.23]
 [802.   28.1   33.72  30.68  22.82]
 [803.   26.16  31.39  28.2   22.53]
 [804.   26.16  31.39  28.78  20.93]
 [805.   26.1   31.32  28.22  20.82]
 [806.   25.45  30.54  27.73  21.05]
 [807.   26.16  31.39  28.01  20.51]
 [808.   27.44  32.93  28.83  22.08]
 [809.   28.63  34.35  31.03  22.68]
 [810.   30.35  36.42  31.38  23.1 ]]
Indices of value 27.79:
(array([], dtype=int64), array([], dtype=int64))
Sorted array:
[[  nan   nan   nan   nan   nan]
 [ 22.23  28.48  30.56  34.18 801.  ]
 [ 22.82  28.1   30.68  33.72 802.  ]
 [ 22.53  26.16  28.2   31.39 803.  ]
 [ 20.93  26.16  28.78  31.39 804.  ]
 [ 20.82  26.1   28.22  31.32 805.  ]
 [ 21.05  25.45  27.73  30.54 806.  ]
 [ 20.51  26.16  28.01  31.39 807.  ]
 [ 22.08  27.44  28.83  32.93 808.  ]
 [ 22.68  28.63  31.03  34.35 809.  ]
 [ 23.1   30.35  31.38  36.42 810.  ]]
Count of value 27.79:
0
Broadcasted array:
[[2 2 2 2 2]
 [2 2 2 2 2]
 [2 2 2 2 2]
 [2 2 2 2 2]
 [2 2 2 2 2]
 [2 2 2 2 2]
 [2 2 2 2 2]
 [2 2 2 2 2]
 [2 2 2 2 2]
 [2 2 2 2 2]
 [2 2 2 2 2]]
```