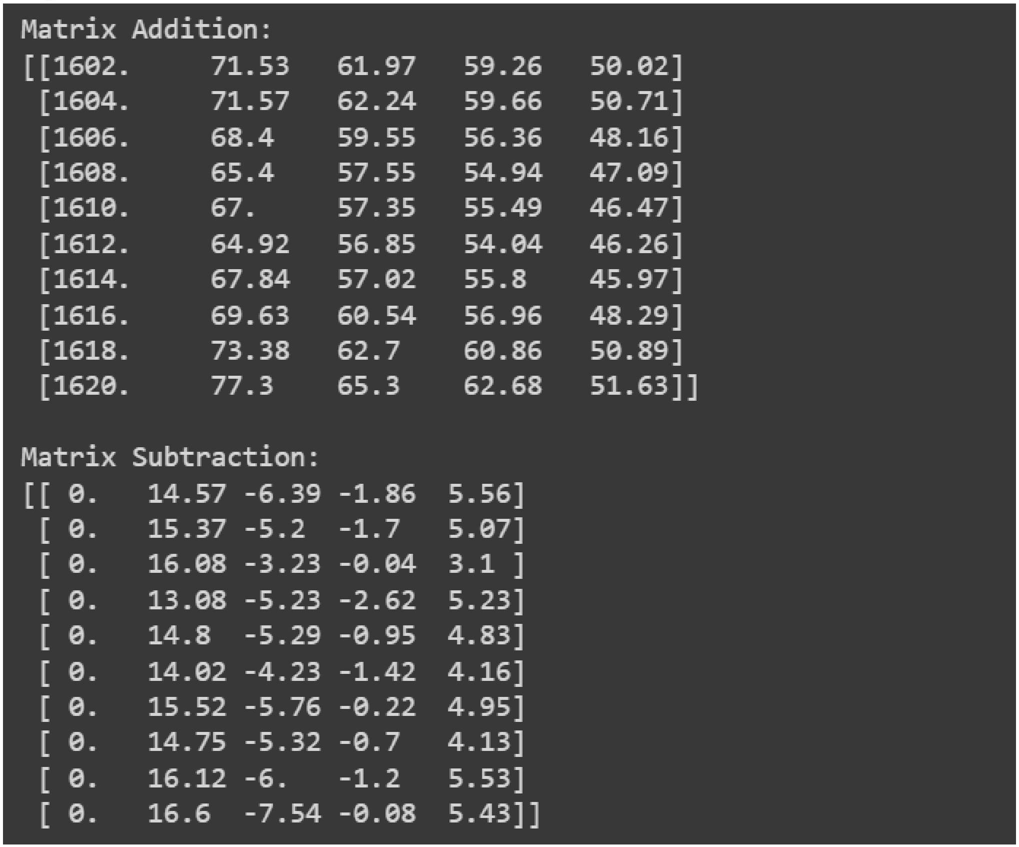
Name   - Ojas Denge
Division - G (4)
Roll No. - 777
PRN No. - 202201070072

EDS ASSIGNMENT-3

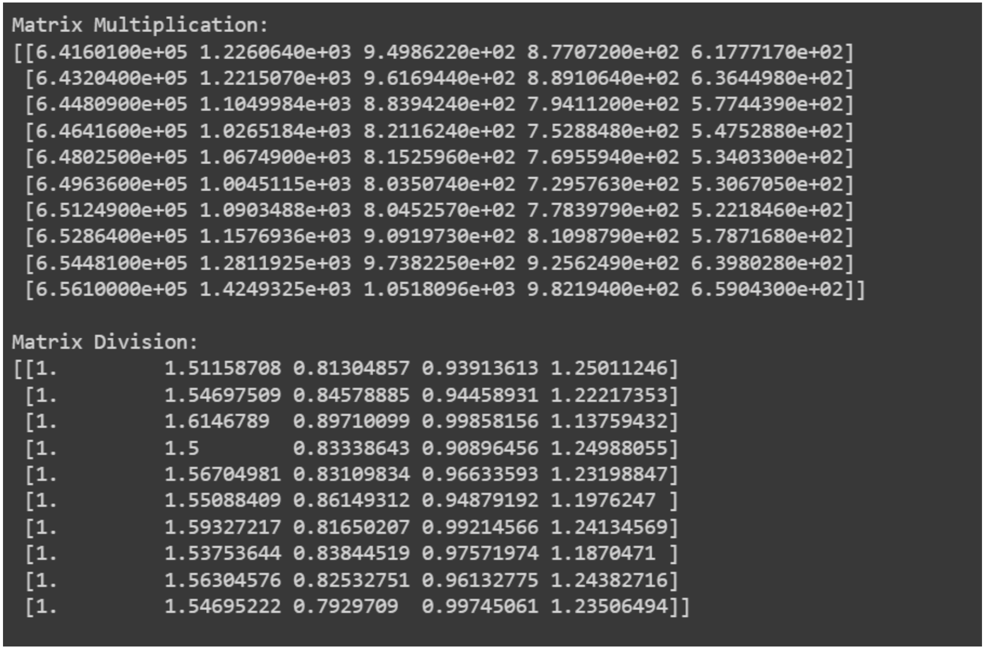
Code:

```
import numpy as np

# Reading a dataset into an array:
'''data1 = np.read_csv("/content/drive/MyDrive/Colab
Notebooks/Files/dataset1.csv")
data2 = np.read_csv("/content/drive/MyDrive/Colab
Notebooks/Files/dataset2.csv")'''

data1 = np.genfromtxt('/content/drive/MyDrive/Colab
Notebooks/Files/dataset1.csv', delimiter=',', skip_header=1)
data2 = np.genfromtxt('/content/drive/MyDrive/Colab
Notebooks/Files/dataset2.csv', delimiter=',', skip_header=1)

# Performing all matrix operations:
# Matrix addition
addition = np.add(data1, data2)
print("Matrix Addition:")
print(addition)
print()
# Matrix subtraction
subtraction = np.subtract(data1, data2)
print("Matrix Subtraction:")
print(subtraction)
print()
# Matrix multiplication
multiplication = np.multiply(data1, data2)
print("Matrix Multiplication:")
print(multiplication)
print()
# Matrix division
division = np.divide(data1, data2)
print("Matrix Division:")
print(division)
print()
# Matrix dot product
dot_product = np.dot(data1, data2.T)
print("Matrix Dot Product:")
print(dot_product)
print()
# Matrix transpose
transpose = data1.T
print("Matrix Transpose:")
print(transpose)
print()
```

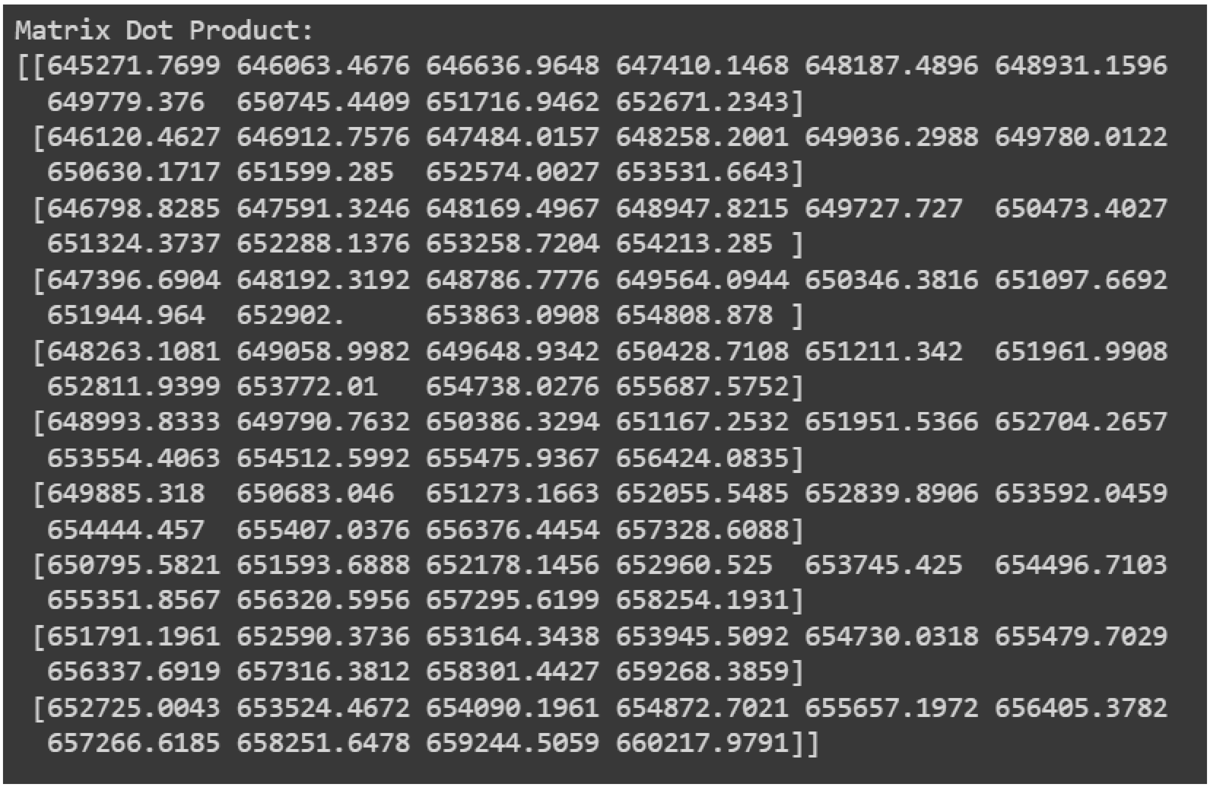
Output:



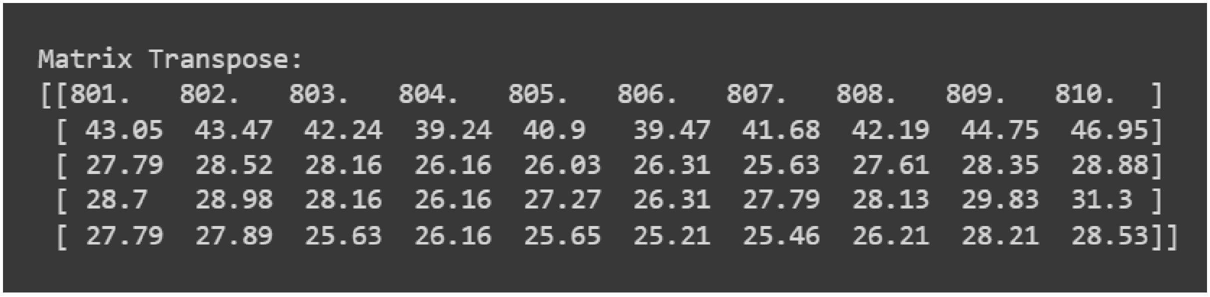
```
Matrix Addition:
[[1602.     71.53   61.97   59.26   50.02]
 [1604.     71.57   62.24   59.66   50.71]
 [1606.     68.4    59.55   56.36   48.16]
 [1608.     65.4    57.55   54.94   47.09]
 [1610.     67.     57.35   55.49   46.47]
 [1612.     64.92   56.85   54.04   46.26]
 [1614.     67.84   57.02   55.8    45.97]
 [1616.     69.63   60.54   56.96   48.29]
 [1618.     73.38   62.7    60.86   50.89]
 [1620.     77.3    65.3    62.68   51.63]]

Matrix Subtraction:
[[ 0.   14.57 -6.39 -1.86  5.56]
 [ 0.   15.37 -5.2  -1.7   5.07]
 [ 0.   16.08 -3.23 -0.04  3.1 ]
 [ 0.   13.08 -5.23 -2.62  5.23]
 [ 0.   14.8  -5.29 -0.95  4.83]
 [ 0.   14.02 -4.23 -1.42  4.16]
 [ 0.   15.52 -5.76 -0.22  4.95]
 [ 0.   14.75 -5.32 -0.7   4.13]
 [ 0.   16.12 -6.   -1.2   5.53]
 [ 0.   16.6  -7.54 -0.08  5.43]]
```



```
Matrix Multiplication:
[[6.4160100e+05 1.2260640e+03 9.4986220e+02 8.7707200e+02 6.1777170e+02]
 [6.4320400e+05 1.2215070e+03 9.6169440e+02 8.8910640e+02 6.3644980e+02]
 [6.4480900e+05 1.1049984e+03 8.8394240e+02 7.9411200e+02 5.7744390e+02]
 [6.4641600e+05 1.0265184e+03 8.2116240e+02 7.5288480e+02 5.4752880e+02]
 [6.4802500e+05 1.0674900e+03 8.1525960e+02 7.6955940e+02 5.3403300e+02]
 [6.4963600e+05 1.0045115e+03 8.0350740e+02 7.2957630e+02 5.3067050e+02]
 [6.5124900e+05 1.0903488e+03 8.0452570e+02 7.7839790e+02 5.2218460e+02]
 [6.5286400e+05 1.1576936e+03 9.0919730e+02 8.1098790e+02 5.7871680e+02]
 [6.5448100e+05 1.2811925e+03 9.7382250e+02 9.2562490e+02 6.3980280e+02]
 [6.5610000e+05 1.4249325e+03 1.0518096e+03 9.8219400e+02 6.5904300e+02]]

Matrix Division:
[[1.         1.51158708 0.81304857 0.93913613 1.25011246]
 [1.         1.54697509 0.84578885 0.94458931 1.22217353]
 [1.         1.6146789  0.89710099 0.99858156 1.13759432]
 [1.         1.5        0.83338643 0.90896456 1.24988055]
 [1.         1.56704981 0.83109834 0.96633593 1.23198847]
 [1.         1.55088409 0.86149312 0.94879192 1.1976247 ]
 [1.         1.59327217 0.81650207 0.99214566 1.24134569]
 [1.         1.53753644 0.83844519 0.97571974 1.1870471 ]
 [1.         1.56304576 0.82532751 0.96132775 1.24382716]
 [1.         1.54695222 0.7929709  0.99745061 1.23506494]]
```

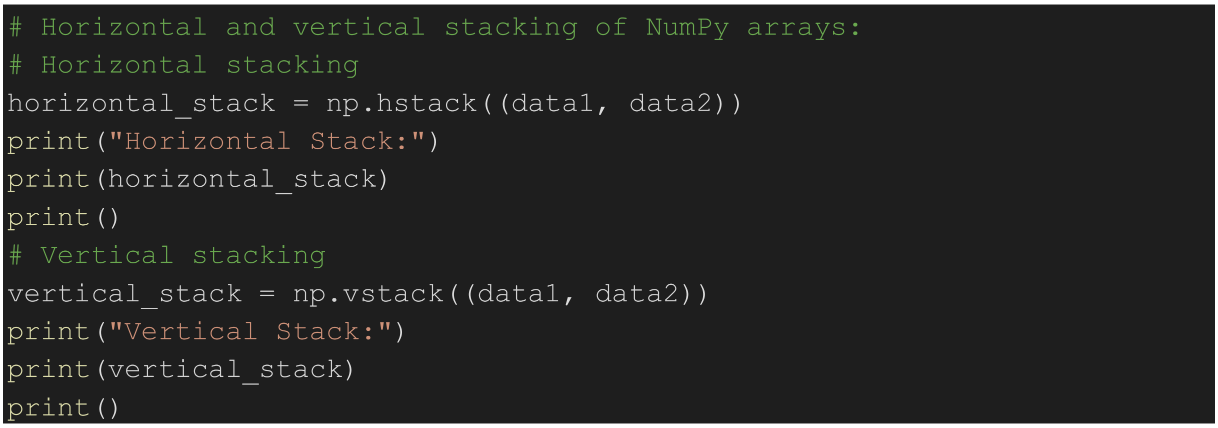


```
Matrix Dot Product:
[[645271.7699 646063.4676 646636.9648 647410.1468 648187.4896 648931.1596
  649779.376  650745.4409 651716.9462 652671.2343]
 [646120.4627 646912.7576 647484.0157 648258.2001 649036.2988 649780.0122
  650630.1717 651599.285  652574.0027 653531.6643]
 [646798.8285 647591.3246 648169.4967 648947.8215 649727.727  650473.4027
  651324.3737 652288.1376 653258.7204 654213.285 ]
 [647396.6904 648192.3192 648786.7776 649564.0944 650346.3816 651097.6692
  651944.964  652902.     653863.0908 654808.878 ]
 [648263.1081 649058.9982 649648.9342 650428.7108 651211.342  651961.9908
  652811.9399 653772.01   654738.0276 655687.5752]
 [648993.8333 649790.7632 650386.3294 651167.2532 651951.5366 652704.2657
  653554.4063 654512.5992 655475.9367 656424.0835]
 [649885.318  650683.046  651273.1663 652055.5485 652839.8906 653592.0459
  654444.457  655407.0376 656376.4454 657328.6088]
 [650795.5821 651593.6888 652178.1456 652960.525  653745.425  654496.7103
  655351.8567 656320.5956 657295.6199 658254.1931]
 [651791.1961 652590.3736 653164.3438 653945.5092 654730.0318 655479.7029
  656337.6919 657316.3812 658301.4427 659268.3859]
 [652725.0043 653524.4672 654090.1961 654872.7021 655657.1972 656405.3782
  657266.6185 658251.6478 659244.5059 660217.9791]]
```



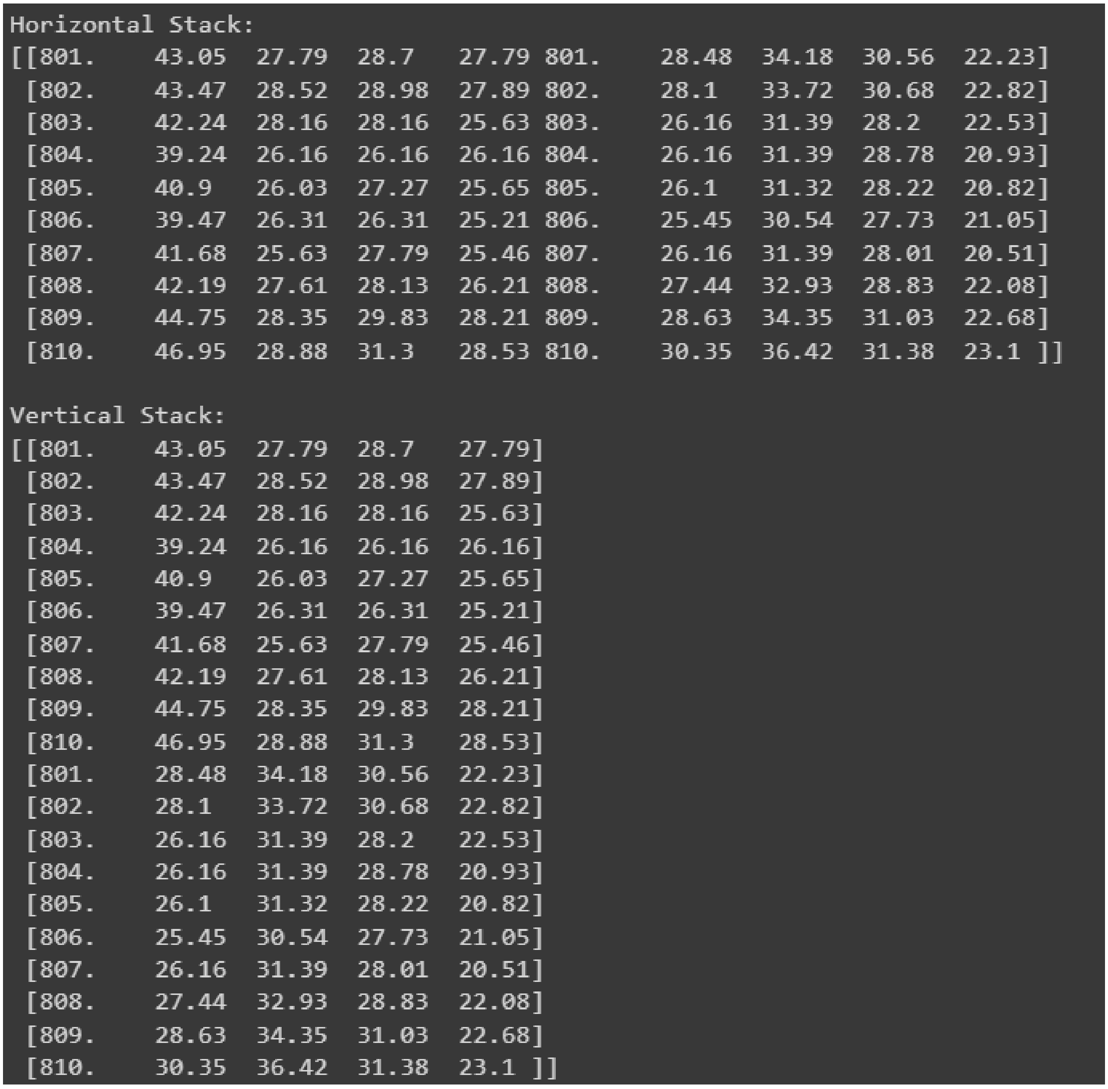
```
Matrix Transpose:
[[801.   802.   803.   804.   805.   806.   807.   808.   809.   810.  ]
 [ 43.05  43.47  42.24  39.24  40.9   39.47  41.68  42.19  44.75  46.95]
 [ 27.79  28.52  28.16  26.16  26.03  26.31  25.63  27.61  28.35  28.88]
 [ 28.7   28.98  28.16  26.16  27.27  26.31  27.79  28.13  29.83  31.3 ]
 [ 27.79  27.89  25.63  26.16  25.65  25.21  25.46  26.21  28.21  28.53]]
```

Code:



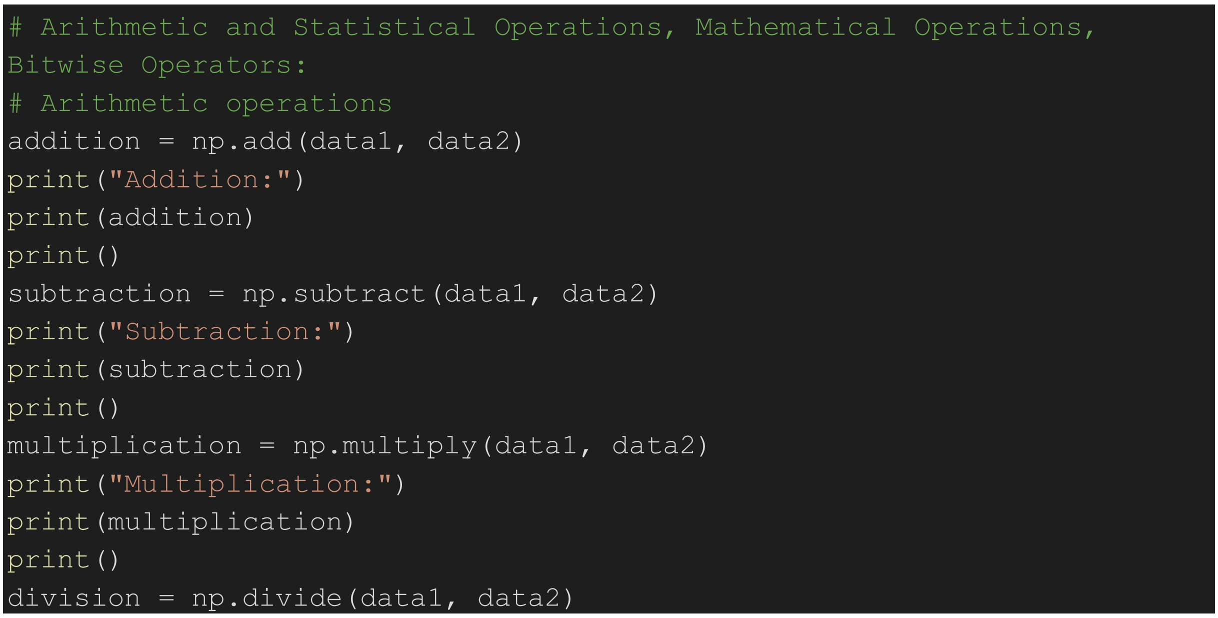
```
# Horizontal and vertical stacking of NumPy arrays:
# Horizontal stacking
horizontal_stack = np.hstack((data1, data2))
print("Horizontal Stack:")
print(horizontal_stack)
print()
# Vertical stacking
vertical_stack = np.vstack((data1, data2))
print("Vertical Stack:")
print(vertical_stack)
print()
```

Output:

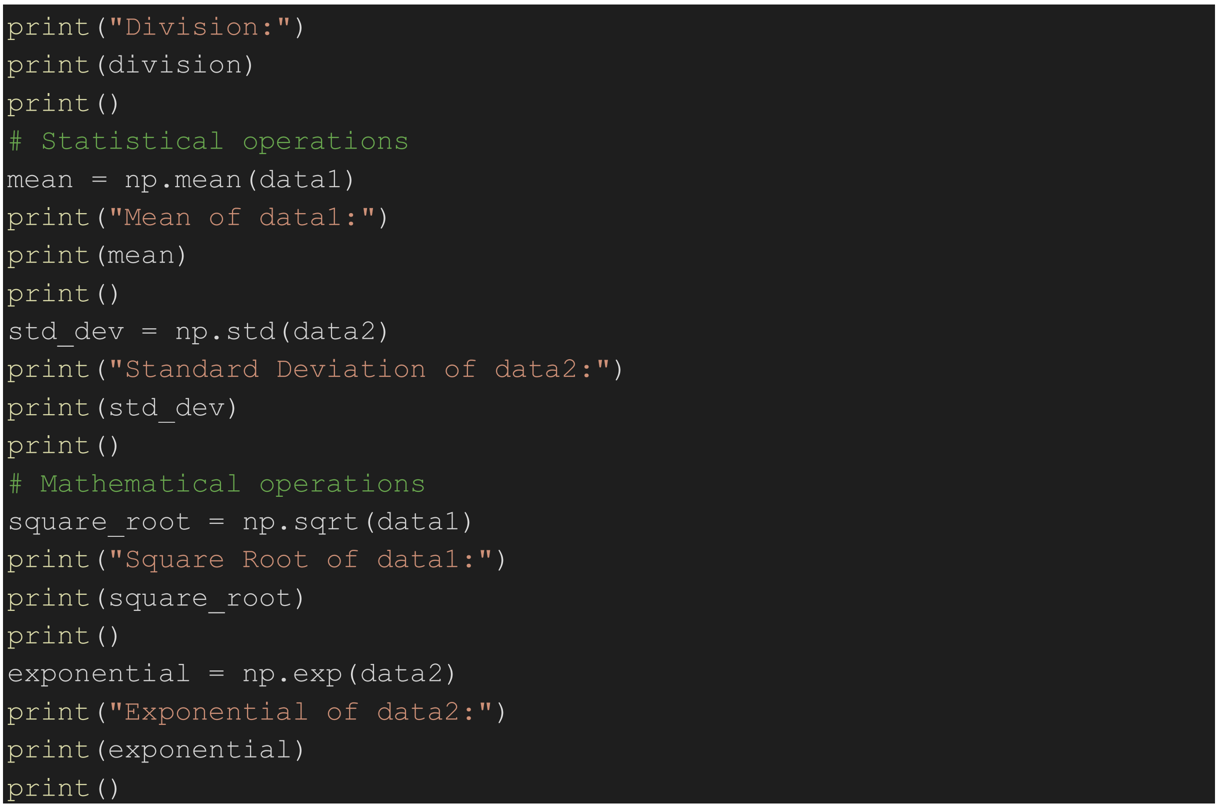


```
Horizontal Stack:
[[801.    43.05  27.79  28.7   27.79 801.    28.48  34.18  30.56  22.23]
 [802.    43.47  28.52  28.98  27.89 802.    28.1   33.72  30.68  22.82]
 [803.    42.24  28.16  28.16  25.63 803.    26.16  31.39  28.2   22.53]
 [804.    39.24  26.16  26.16  26.16 804.    26.16  31.39  28.78  20.93]
 [805.    40.9   26.03  27.27  25.65 805.    26.1   31.32  28.22  20.82]
 [806.    39.47  26.31  26.31  25.21 806.    25.45  30.54  27.73  21.05]
 [807.    41.68  25.63  27.79  25.46 807.    26.16  31.39  28.01  20.51]
 [808.    42.19  27.61  28.13  26.21 808.    27.44  32.93  28.83  22.08]
 [809.    44.75  28.35  29.83  28.21 809.    28.63  34.35  31.03  22.68]
 [810.    46.95  28.88  31.3   28.53 810.    30.35  36.42  31.38  23.1 ]]

Vertical Stack:
[[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]
 [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 ]]
```

Code:

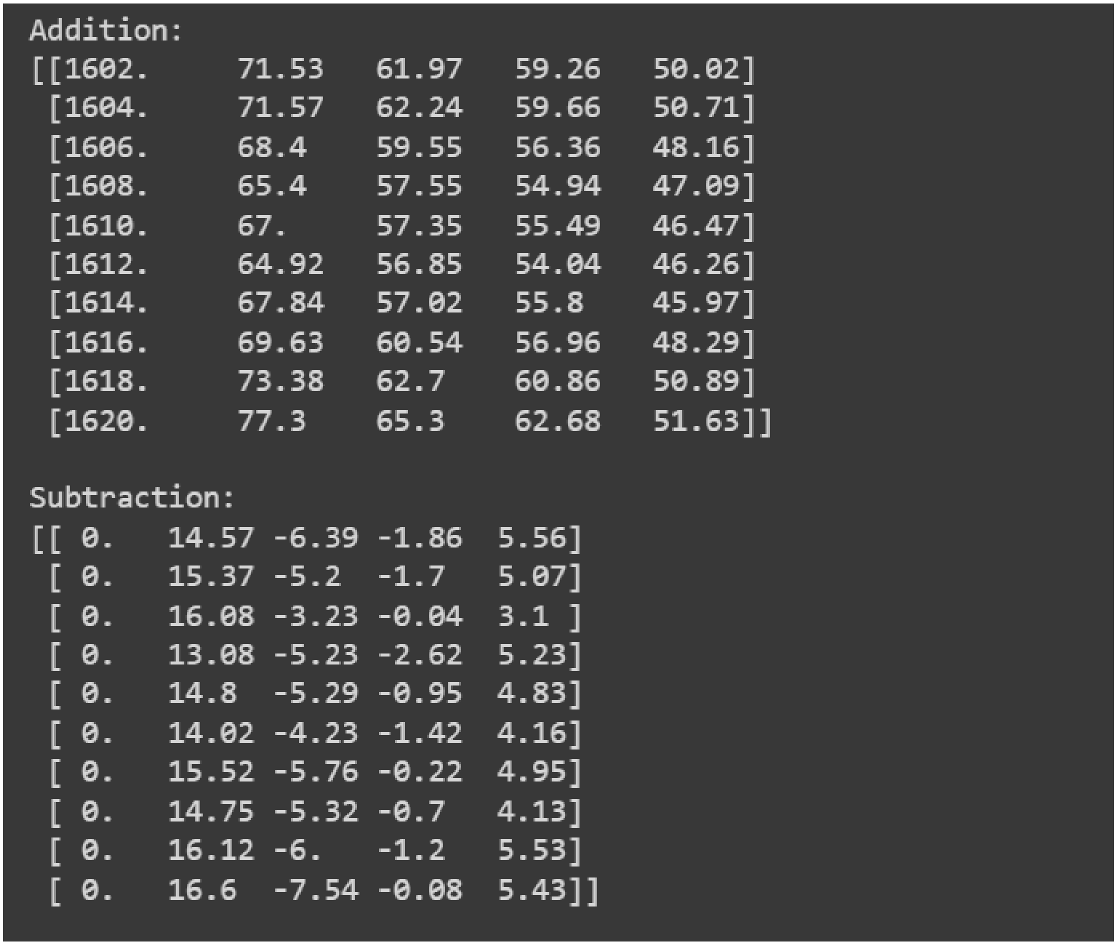


```
# Arithmetic and Statistical Operations, Mathematical Operations,
Bitwise Operators:
# Arithmetic operations
addition = np.add(data1, data2)
print("Addition:")
print(addition)
print()
subtraction = np.subtract(data1, data2)
print("Subtraction:")
print(subtraction)
print()
multiplication = np.multiply(data1, data2)
print("Multiplication:")
print(multiplication)
print()
division = np.divide(data1, data2)
```

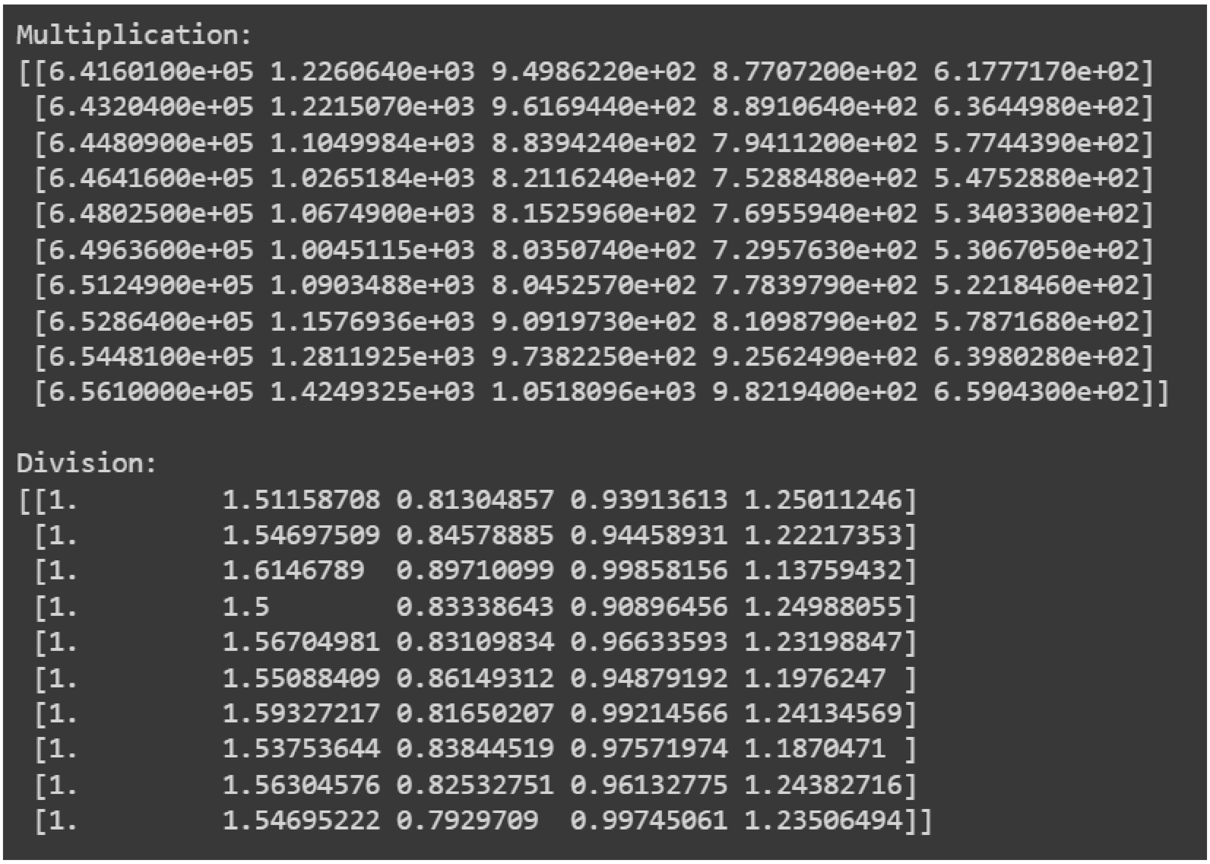


```
print("Division:")
print(division)
print()
# Statistical operations
mean = np.mean(data1)
print("Mean of data1:")
print(mean)
print()
std_dev = np.std(data2)
print("Standard Deviation of data2:")
print(std_dev)
print()
# Mathematical operations
square_root = np.sqrt(data1)
print("Square Root of data1:")
print(square_root)
print()
exponential = np.exp(data2)
print("Exponential of data2:")
print(exponential)
print()
```

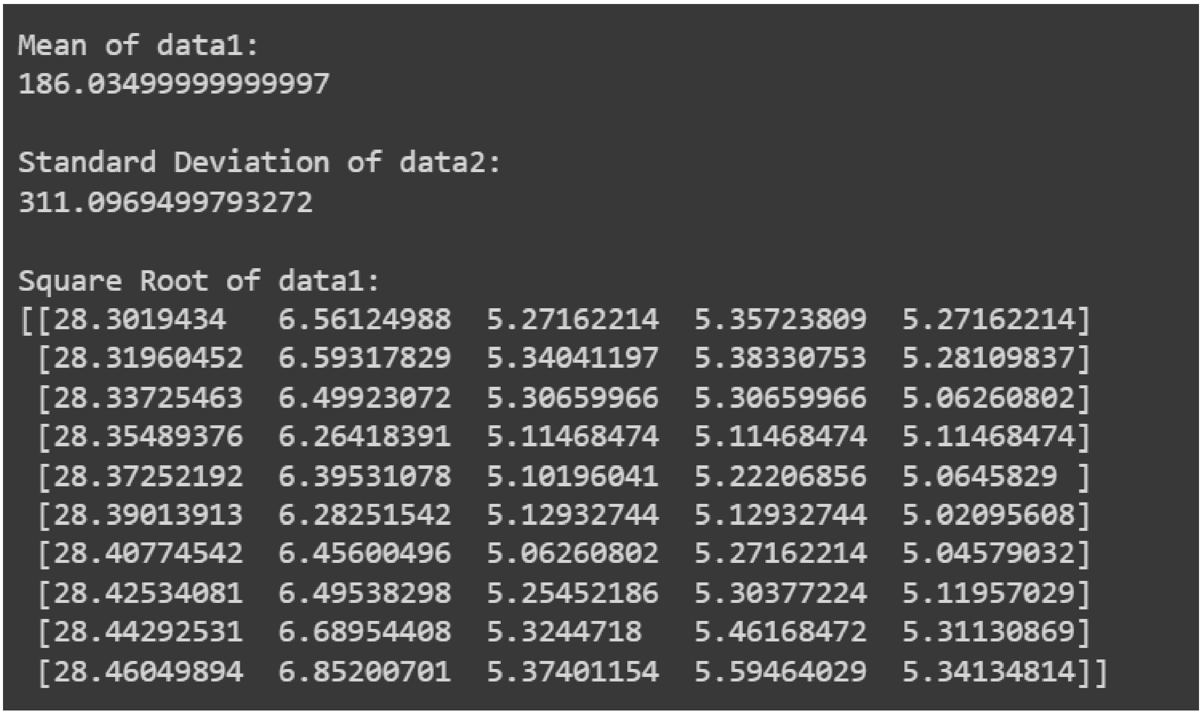
Output:



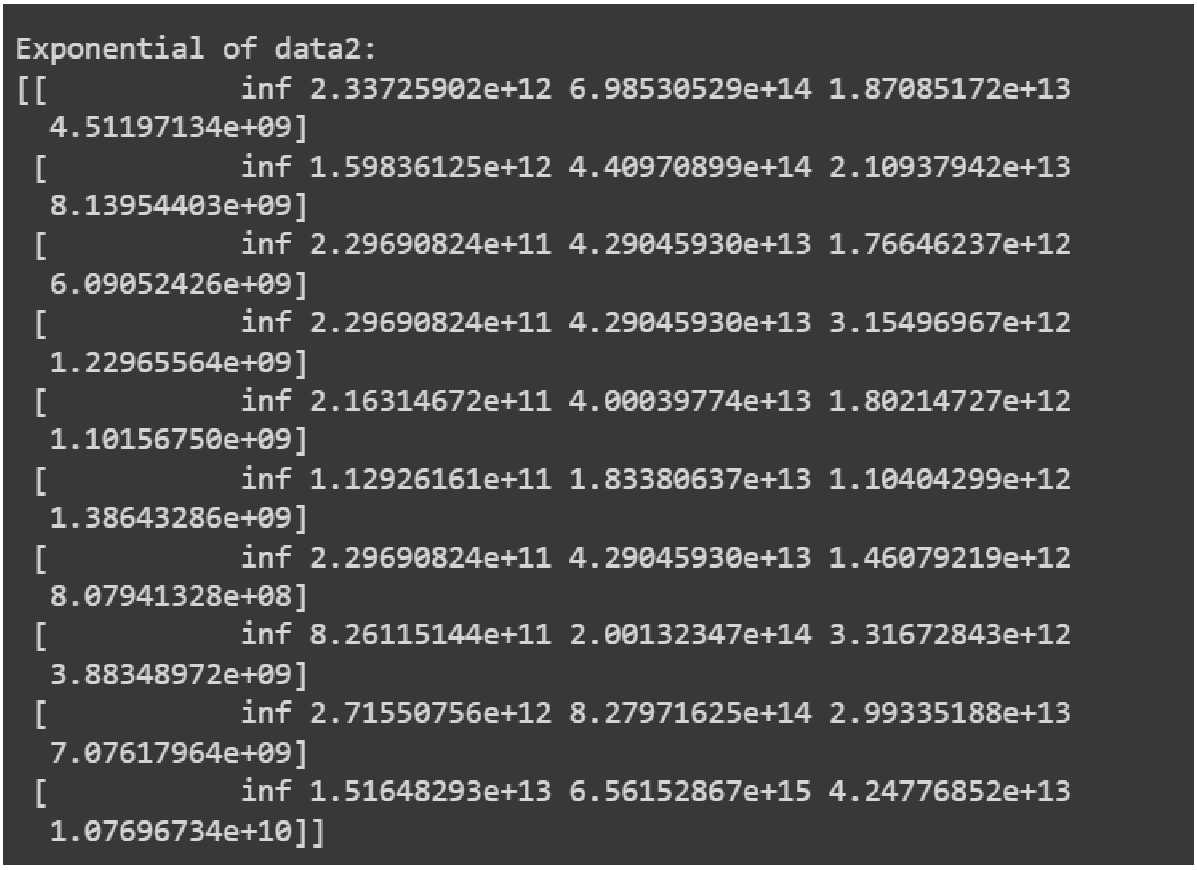
```
Addition:
[[1602.     71.53   61.97   59.26   50.02]
 [1604.     71.57   62.24   59.66   50.71]
 [1606.     68.4    59.55   56.36   48.16]
 [1608.     65.4    57.55   54.94   47.09]
 [1610.     67.     57.35   55.49   46.47]
 [1612.     64.92   56.85   54.04   46.26]
 [1614.     67.84   57.02   55.8    45.97]
 [1616.     69.63   60.54   56.96   48.29]
 [1618.     73.38   62.7    60.86   50.89]
 [1620.     77.3    65.3    62.68   51.63]]

Subtraction:
[[ 0.   14.57 -6.39 -1.86  5.56]
 [ 0.   15.37 -5.2  -1.7   5.07]
 [ 0.   16.08 -3.23 -0.04  3.1 ]
 [ 0.   13.08 -5.23 -2.62  5.23]
 [ 0.   14.8  -5.29 -0.95  4.83]
 [ 0.   14.02 -4.23 -1.42  4.16]
 [ 0.   15.52 -5.76 -0.22  4.95]
 [ 0.   14.75 -5.32 -0.7   4.13]
 [ 0.   16.12 -6.   -1.2   5.53]
 [ 0.   16.6  -7.54 -0.08  5.43]]
```



```
Multiplication:
[[6.4160100e+05 1.2260640e+03 9.4986220e+02 8.7707200e+02 6.1777170e+02]
 [6.4320400e+05 1.2215070e+03 9.6169440e+02 8.8910640e+02 6.3644980e+02]
 [6.4480900e+05 1.1049984e+03 8.8394240e+02 7.9411200e+02 5.7744390e+02]
 [6.4641600e+05 1.0265184e+03 8.2116240e+02 7.5288480e+02 5.4752880e+02]
 [6.4802500e+05 1.0674900e+03 8.1525960e+02 7.6955940e+02 5.3403300e+02]
 [6.4963600e+05 1.0045115e+03 8.0350740e+02 7.2957630e+02 5.3067050e+02]
 [6.5124900e+05 1.0903488e+03 8.0452570e+02 7.7839790e+02 5.2218460e+02]
 [6.5286400e+05 1.1576936e+03 9.0919730e+02 8.1098790e+02 5.7871680e+02]
 [6.5448100e+05 1.2811925e+03 9.7382250e+02 9.2562490e+02 6.3980280e+02]
 [6.5610000e+05 1.4249325e+03 1.0518096e+03 9.8219400e+02 6.5904300e+02]]

Division:
[[1.         1.51158708 0.81304857 0.93913613 1.25011246]
 [1.         1.54697509 0.84578885 0.94458931 1.22217353]
 [1.         1.6146789  0.89710099 0.99858156 1.13759432]
 [1.         1.5        0.83338643 0.90896456 1.24988055]
 [1.         1.56704981 0.83109834 0.96633593 1.23198847]
 [1.         1.55088409 0.86149312 0.94879192 1.1976247 ]
 [1.         1.59327217 0.81650207 0.99214566 1.24134569]
 [1.         1.53753644 0.83844519 0.97571974 1.1870471 ]
 [1.         1.56304576 0.82532751 0.96132775 1.24382716]
 [1.         1.54695222 0.7929709  0.99745061 1.23506494]]
```



```
Mean of data1:
186.03499999999997

Standard Deviation of data2:
311.0969499793272

Square Root of data1:
[[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]]
```



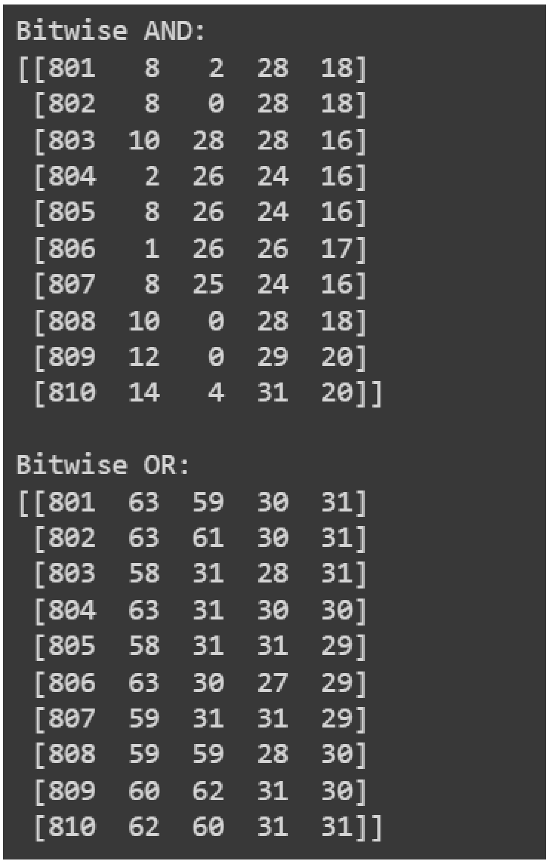
```
Exponential of data2:
[[           inf 2.33725902e+12 6.98530529e+14 1.87085172e+13
  4.51197134e+09]
 [           inf 1.59836125e+12 4.40970899e+14 2.10937942e+13
  8.13954403e+09]
 [           inf 2.29690824e+11 4.29045930e+13 1.76646237e+12
  6.09052426e+09]
 [           inf 2.29690824e+11 4.29045930e+13 3.15496967e+12
  1.22965564e+09]
 [           inf 2.16314672e+11 4.00039774e+13 1.80214727e+12
  1.10156750e+09]
 [           inf 1.12926161e+11 1.83380637e+13 1.10404299e+12
  1.38643286e+09]
 [           inf 2.29690824e+11 4.29045930e+13 1.46079219e+12
  8.07941328e+08]
 [           inf 8.26115144e+11 2.00132347e+14 3.31672843e+12
  3.88348972e+09]
 [           inf 2.71550756e+12 8.27971625e+14 2.99335188e+13
  7.07617964e+09]
 [           inf 1.51648293e+13 6.56152867e+15 4.24776852e+13
  1.07696734e+10]]
```

Code:

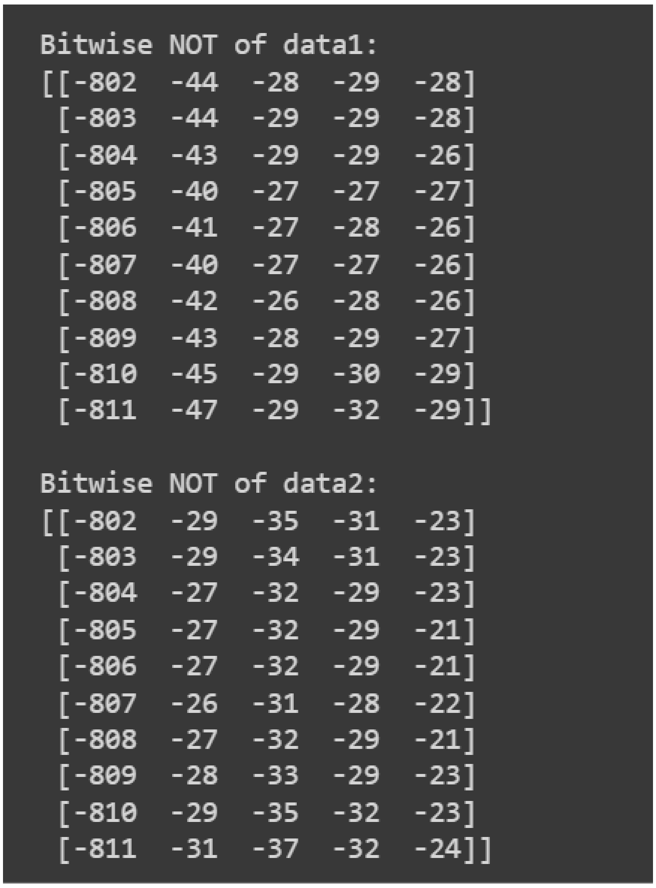
```
# Bitwise operators
# Bitwise AND
bitwise_and = np.bitwise_and(data1.astype(int), data2.astype(int))
print("Bitwise AND:")
print(bitwise_and)
print()
# Bitwise OR
bitwise_or = np.bitwise_or(data1.astype(int), data2.astype(int))
print("Bitwise OR:")
print(bitwise_or)
print()
# Bitwise XOR
bitwise_xor = np.bitwise_xor(data1.astype(int), data2.astype(int))
print("Bitwise XOR:")
print(bitwise_xor)
print()
# Bitwise NOT
bitwise_not_data1 = np.bitwise_not(data1.astype(int))
print("Bitwise NOT of data1:")
print(bitwise_not_data1)
print()
bitwise_not_data2 = np.bitwise_not(data2.astype(int))
```

```
print("Bitwise NOT of data2:")
print(bitwise_not_data2)
print()
```

Output:



```
Bitwise AND:
[[801   8   2  28  18]
 [802   8   0  28  18]
 [803  10  28  28  16]
 [804   2  26  24  16]
 [805   8  26  24  16]
 [806   1  26  26  17]
 [807   8  25  24  16]
 [808  10   0  28  18]
 [809  12   0  29  20]
 [810  14   4  31  20]]

Bitwise OR:
[[801  63  59  30  31]
 [802  63  61  30  31]
 [803  58  31  28  31]
 [804  63  31  30  30]
 [805  58  31  31  29]
 [806  63  30  27  29]
 [807  59  31  31  29]
 [808  59  59  28  30]
 [809  60  62  31  30]
 [810  62  60  31  31]]
```



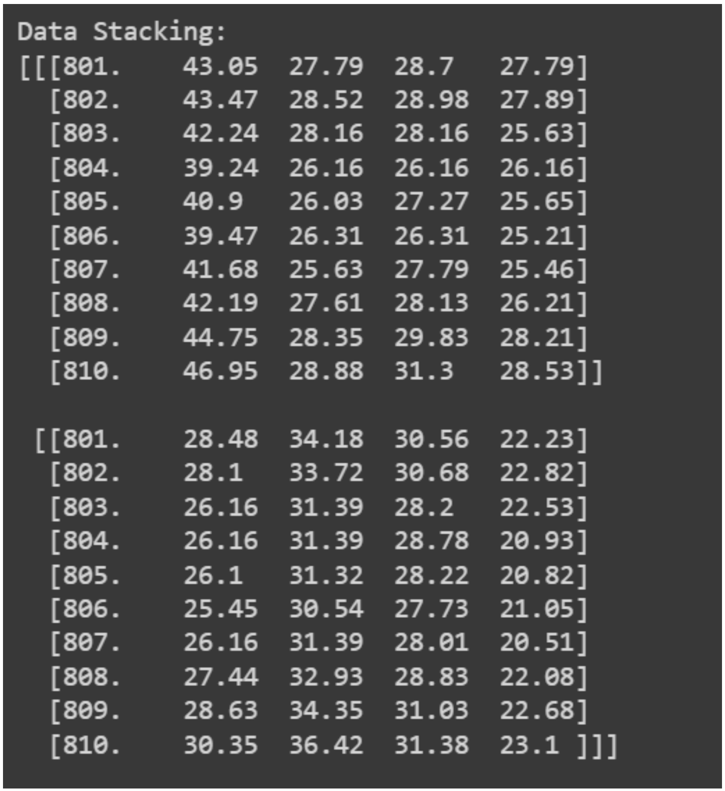
```
Bitwise NOT of data1:
[[-802  -44  -28  -29  -28]
 [-803  -44  -29  -29  -28]
 [-804  -43  -29  -29  -26]
 [-805  -40  -27  -27  -27]
 [-806  -41  -27  -28  -26]
 [-807  -40  -27  -27  -26]
 [-808  -42  -26  -28  -26]
 [-809  -43  -28  -29  -27]
 [-810  -45  -29  -30  -29]
 [-811  -47  -29  -32  -29]]

Bitwise NOT of data2:
[[-802  -29  -35  -31  -23]
 [-803  -29  -34  -31  -23]
 [-804  -27  -32  -29  -23]
 [-805  -27  -32  -29  -21]
 [-806  -27  -32  -29  -21]
 [-807  -26  -31  -28  -22]
 [-808  -27  -32  -29  -21]
 [-809  -28  -33  -29  -23]
 [-810  -29  -35  -32  -23]
 [-811  -31  -37  -32  -24]]
```

Code:

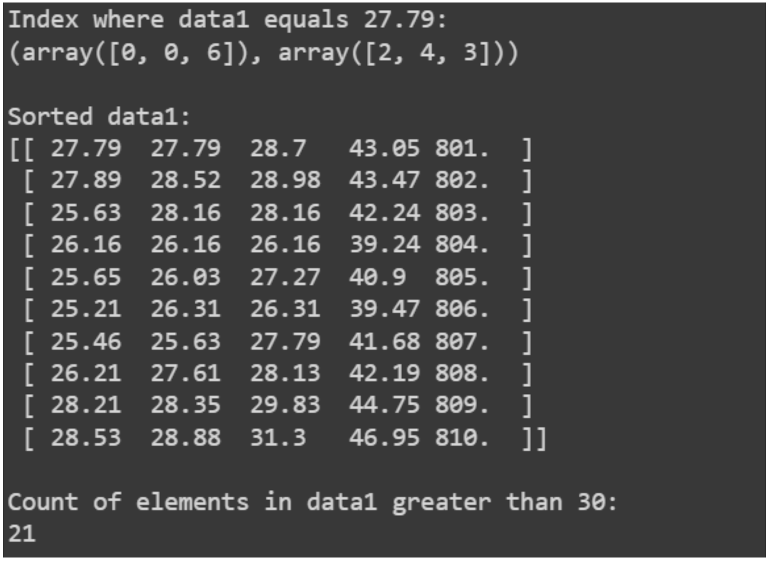
```
# Data Stacking, Searching, Sorting, Counting, Broadcasting:
# Data stacking
stacked_data = np.stack((data1, data2), axis=0)
print("Data Stacking:")
print(stacked_data)
print()
# Searching
index = np.where(data1 == 27.79)
print("Index where data1 equals 27.79:")
print(index)
print()
# Sorting
sorted_data = np.sort(data1)
print("Sorted data1:")
print(sorted_data)
print()
# Counting
```

```
count = np.count_nonzero(data1 > 30)
print("Count of elements in data1 greater than 30:")
print(count)
print()
# Broadcasting
broadcasted_data = data1 * 2
print("Broadcasted data1 (multiplied by 2):")
print(broadcasted_data)
print()
```

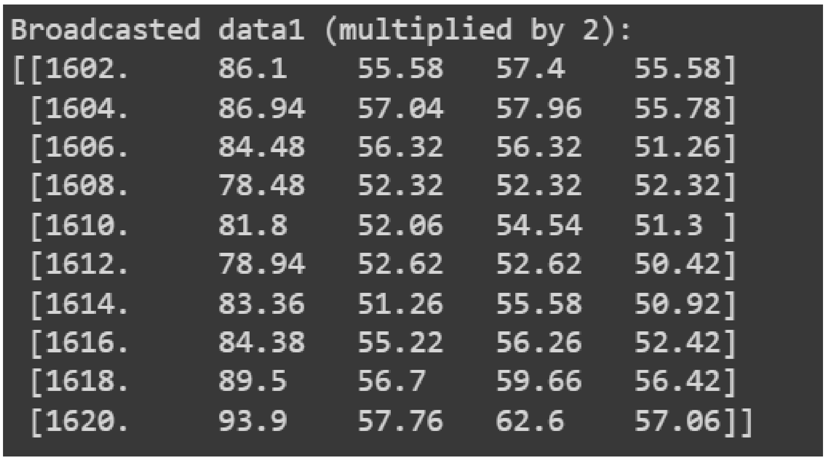
Output:



```
Data Stacking:
[[[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]]

 [[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 ]]]
```



```
Index where data1 equals 27.79:
(array([0, 0, 6]), array([2, 4, 3]))

Sorted data1:
[[ 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 elements in data1 greater than 30:
21
```



```
Broadcasted data1 (multiplied by 2):
[[1602.      86.1     55.58    57.4     55.58]
 [1604.      86.94    57.04    57.96    55.78]
 [1606.      84.48    56.32    56.32    51.26]
 [1608.      78.48    52.32    52.32    52.32]
 [1610.      81.8     52.06    54.54    51.3 ]
 [1612.      78.94    52.62    52.62    50.42]
 [1614.      83.36    51.26    55.58    50.92]
 [1616.      84.38    55.22    56.26    52.42]
 [1618.      89.5     56.7     59.66    56.42]
 [1620.      93.9     57.76    62.6     57.06]]
```

Google Colab Link: https://colab.research.google.com/drive/1rZetINUegCObbBIM3zcBBY-G5tQh9HjU?usp=sharing